

TargetLink

Adaptive AUTOSAR 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.

© 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	5
Introduction	9
Introduction to Adaptive AUTOSAR.....	9
Introduction to Using TargetLink With Adaptive AUTOSAR.....	10
Overview of Supported Modeling Styles for Adaptive AUTOSAR.....	12
Basics on Service Discovery in TargetLink.....	16
Basics on Code Artifacts generated by TargetLink.....	20
Details on Integrating Adaptive AUTOSAR Code Artifacts.....	24
Basics on SIL Simulation for Adaptive AUTOSAR Models.....	28
Specifying Data Types	31
Introduction to Data Types in Adaptive AUTOSAR.....	32
Introduction to Data Types in Adaptive AUTOSAR.....	32
Defining Scalings and Constrained Range Limits (Adaptive AUTOSAR).....	35
Basics on Scalings and Constrained Range Limits in Adaptive AUTOSAR.....	35
Creating CPP Implementation Data Types From Scratch.....	36
How to Define Scalar Implementation Data Types For Adaptive AUTOSAR.....	36
How to Define 1-D Array Implementation Data Types for Adaptive AUTOSAR.....	37
How to Define 2-D Array Implementation Data Types for Adaptive AUTOSAR.....	39
How to Define Structured Implementation Data Types for Adaptive AUTOSAR.....	40
Creating Application Data Types (Adaptive AUTOSAR).....	43
Creating Application Data Types from Scratch (Adaptive AUTOSAR).....	43
Preparatory Steps	45
How to Start Modeling from Scratch.....	45
How to Create Software Components for Adaptive AUTOSAR.....	46
How to Specify an Adaptive AUTOSAR Function.....	47
How To Create Interfaces (Adaptive AUTOSAR).....	48

How to Create Ports.....	49
How to Specify Service Discovery and Instance Selection.....	51
How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Create Communication Subjects for Adaptive AUTOSAR.....	53
 Modeling Communication as Defined by Adaptive AUTOSAR	 57
Communication as defined by ara::com.....	58
Accessing Fields.....	58
How to Get or Set the Value of a Field As a Service Consumer.....	58
How to Update the Value of a Field as a Service Provider.....	60
Receiving and Sending Events.....	62
How to Receive Events (Adaptive AUTOSAR).....	62
How to Send Events (Adaptive AUTOSAR).....	64
Calling and Implementing Methods.....	66
How to Model a Method Call.....	66
How to Model a Method Implementation.....	68
Communication as defined by ara::per.....	71
Accessing Persistent Memory.....	71
How To Model Access to a Key-Value Pair of a Key-Value Storage.....	71
How To Specify Initial Values for Persistency Data Elements.....	73
 Glossary	 77
 Index	 107

About This Guide

Contents

This guide introduces you to generate code that is compliant with [Adaptive AUTOSAR](#) by using TargetLink.

Note

A separate license is required for generating [Adaptive AUTOSAR](#) software components (SWCs) containing function code for ECUs and for modeling and simulating them.

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](#).

Required knowledge

Standards

- [Adaptive AUTOSAR](#)

Concepts









- Object-oriented programming
- Service-oriented architectures
- Model-based development and code generation

Programming languages

- C++
- C

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

Where to go from here

Information in this section

Introduction to Adaptive AUTOSAR.....	9
Introduction to Using TargetLink With Adaptive AUTOSAR.....	10
Overview of Supported Modeling Styles for Adaptive AUTOSAR.....	12
Basics on Service Discovery in TargetLink.....	16
Basics on Code Artifacts generated by TargetLink.....	20
Details on Integrating Adaptive AUTOSAR Code Artifacts.....	24
Basics on SIL Simulation for Adaptive AUTOSAR Models.....	28

Introduction to Adaptive AUTOSAR

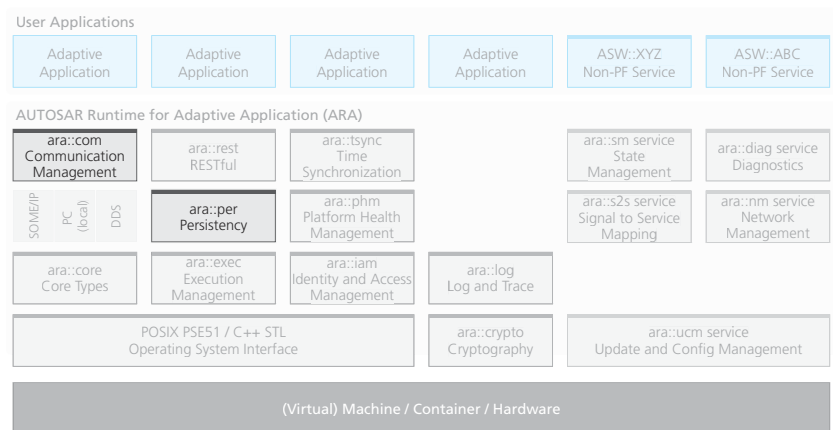
Introduction

AUTOSAR introduced [Adaptive AUTOSAR](#) to complement the well established [Classic AUTOSAR](#). Both provide a three-layer approach that distinguishes between applications, middleware and hardware.

The architecture of [Adaptive AUTOSAR](#) enables service-oriented communication between dynamically changing communication partners. This mainly is achieved during runtime via the middleware.

Middleware and functional clusters

In [Adaptive AUTOSAR](#), the *AUTOSAR Runtime for Adaptive Applications (ARA)* works as the middleware between adaptive applications and the hardware. ARA includes several functional clusters. Modeling with TargetLink mainly revolves around the highlighted clusters, as shown in the following illustration (adapted from figure 3-1 contained in the *AUTOSAR_EXP_PlatformDesign* document):



Adaptive applications

The application layer mainly consists of several adaptive applications. These adaptive applications are collections of executables.

Related topics

Basics

Introduction to Using TargetLink With Adaptive AUTOSAR.....	10
Overview of Supported Modeling Styles for Adaptive AUTOSAR.....	12

Introduction to Using TargetLink With Adaptive AUTOSAR

Development approach

Model-based development TargetLink lets you develop functional parts of adaptive applications with a model-based approach. It provides different model elements to model the functionalities.

These model elements are the following subsystems:

- [Adaptive AUTOSAR Function subsystem](#)
- [Method Behavior subsystem](#)
- [Method Call subsystem](#)

For an overview of the supported modeling styles, refer to [Overview of Supported Modeling Styles for Adaptive AUTOSAR](#) on page 12.

Code generation During code generation, TargetLink generates [Adaptive AUTOSAR functions](#) from Method Behavior subsystems and Adaptive AUTOSAR Function subsystems.

As a code generator that generates code from MATLAB/Simulink® models, TargetLink does not generate the code of entire adaptive applications. This is the case because many parts of this code cannot be conclusively modeled in Simulink.

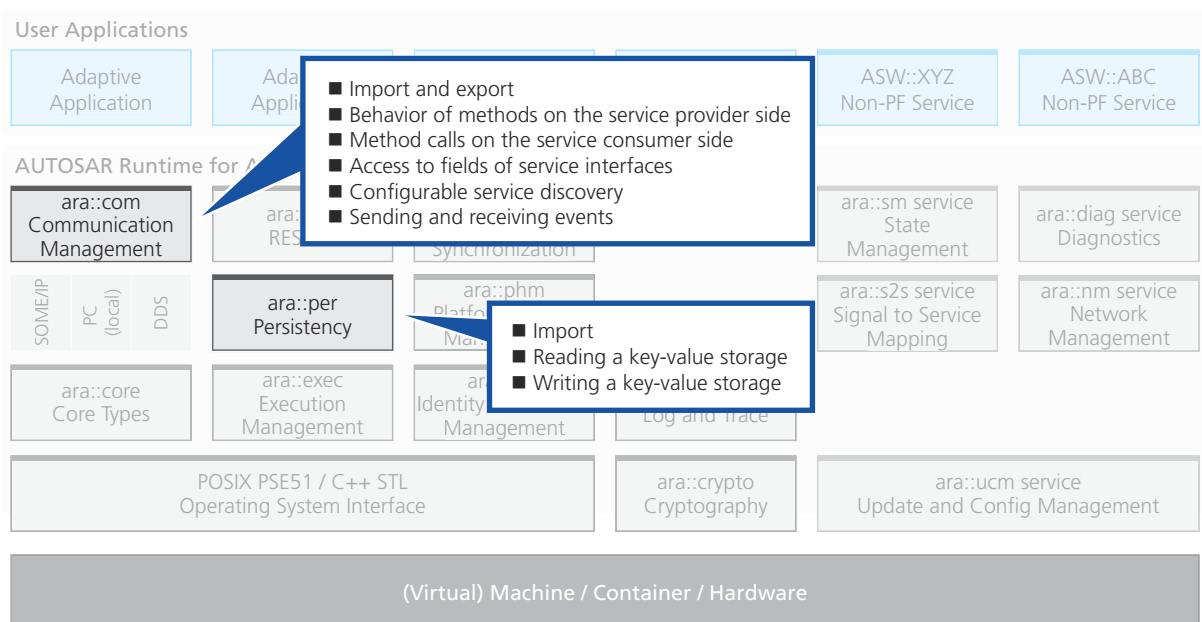
Supported Adaptive AUTOSAR features

TargetLink supports select features of [Adaptive AUTOSAR](#).

The modeling constructs and Data Dictionary workflow for [Adaptive AUTOSAR](#) closely resemble the ones used for [Classic AUTOSAR](#).

The following features are supported:

- Import and export of Adaptive AUTOSAR ARXMLs with elements defined by ara::com.
- Import of Adaptive AUTOSAR ARXMLs with elements defined by ara::per
- Modeling of select parts of a service-based communication as described by ara::com:
 - Behavior of methods on the service provider side via [Method Behavior subsystems](#). Refer to [Calling and Implementing Methods](#) on page 66.
 - Method calls on the service consumer side via [Method Call subsystems](#). Refer to [Calling and Implementing Methods](#) on page 66.
 - Access to fields of service interfaces. Refer to [Accessing Fields](#) on page 58.
 - Configurable service discovery. Refer to [Basics on Service Discovery in TargetLink](#) on page 16.
 - Sending and receiving events as defined in Adaptive AUTOSAR release 18-10. Refer to [Receiving and Sending Events](#) on page 62.
 - SIL simulation of Adaptive AUTOSAR components. Refer to [Basics on SIL Simulation for Adaptive AUTOSAR Models](#) on page 28.
- Modeling of select parts of accessing persistent memory as described by ara::per:
 - Read and write access to key-value pairs with an AdaptivePlatformType from key-value storages via Data Store blocks. Refer to [How To Model Access to a Key-Value Pair of a Key-Value Storage](#) on page 71.



Integrating the generated code

For details on how to integrate code generated by TargetLink into an adaptive application, refer to [Basics on Code Artifacts generated by TargetLink](#) on page 20.

Overview of Supported Modeling Styles for Adaptive AUTOSAR

Support of ara::com and ara::per

ara::com TargetLink lets you model communication as defined by the ara::com API of [Adaptive AUTOSAR](#).

The following use cases are supported:

- [Accessing fields as a service consumer](#) on page 12
- [Accessing fields as a service provider](#) on page 13
- [Sending events](#) on page 13
- [Receiving events](#) on page 14
- [Calling a method](#) on page 14
- [Implementing a method](#) on page 14

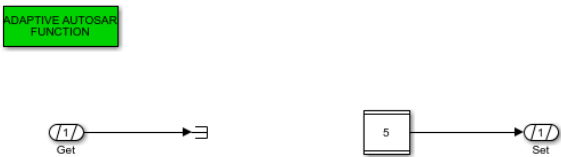
ara::per TargetLink lets you model persistent data access as defined by the ara::per API of Adaptive AUTOSAR.

The following use cases are supported:

- [Reading and writing key-value pairs from a key-value storage](#) on page 15

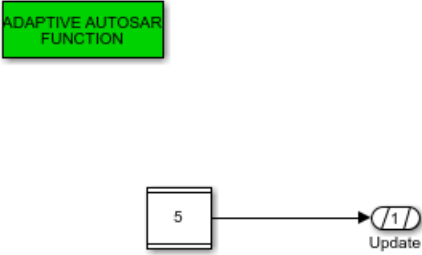
Accessing fields as a service consumer

For this use case, TargetLink lets you use the following model elements:

Modeling Elements	Description
	<p>You use this modeling style to access fields as a service consumer via <code>Get()</code> and <code>Set()</code> methods. You can use it in the following subsystems:</p> <ul style="list-style-type: none"> ▪ Adaptive AUTOSAR Function subsystem ▪ Method Behavior subsystem <p>Refer to How to Get or Set the Value of a Field As a Service Consumer on page 58.</p>

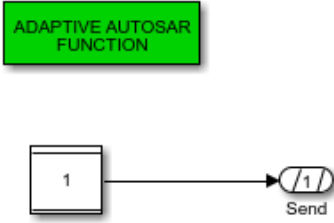
Accessing fields as a service provider

For this use case, TargetLink lets you use the following model elements:

Modeling Elements	Description
	<p>You use this modeling style to access fields as a service provider via the <code>Update()</code> method. You can use it in the following subsystems:</p> <ul style="list-style-type: none"> 🔗 Adaptive AUTOSAR Function subsystem 🔗 Method Behavior subsystem <p>Refer to How to Update the Value of a Field as a Service Provider on page 60.</p>

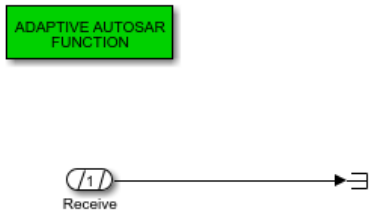
Sending events

For this use case, TargetLink lets you use the following model elements:

Modeling elements	Description
	<p>You use this modeling style to send events via the <code>Send()</code> method. You can use it in the following subsystems:</p> <ul style="list-style-type: none"> 🔗 Adaptive AUTOSAR Function subsystem 🔗 Method Behavior subsystem <p>Refer to How to Send Events (Adaptive AUTOSAR) on page 64.</p>

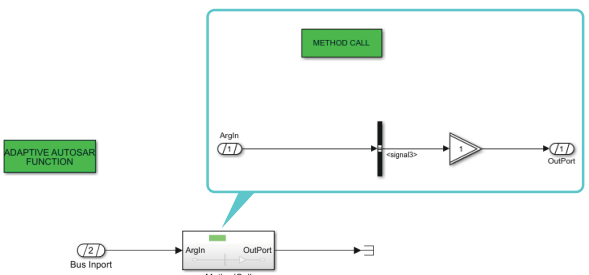
Receiving events

For this use case, TargetLink lets you use the following model elements:

Modeling elements	Description
	<p>You use this modeling style to receive events via the <code>Update()</code> and <code>GetCachedSamples()</code> methods. You can use it in the following subsystems:</p> <ul style="list-style-type: none"> Adaptive AUTOSAR Function subsystem Method Behavior subsystem <p>Refer to How to Receive Events (Adaptive AUTOSAR) on page 62.</p>


Calling a method

For this use case, TargetLink lets you use the following model elements:

Modeling Elements	Description
	<p>You use a Method Call subsystem to model the call of a method that is provided by a service. You can use it in the following subsystems:</p> <ul style="list-style-type: none"> Adaptive AUTOSAR Function subsystem Method Behavior subsystem <p>Refer to How to Model a Method Call on page 66.</p>


Implementing a method

For this use case, TargetLink lets you use the following model elements:

Modeling Elements	Description
	<p>You use a Method Behavior subsystem to model the implementation of a method on the service provider side. Refer to How to Model a Method Implementation on page 68.</p>

Reading and writing key-value pairs from a key-value storage

For this use case, TargetLink lets you use the following model elements:

Modeling Elements	Description
	<p>You use this modeling style to read and write key-value pairs from key-value storages via the <code>GetValue</code> and <code>SetValue</code> methods. You can use it in the following subsystems:</p> <ul style="list-style-type: none">Adaptive AUTOSAR Function subsystemMethod Behavior subsystem <p>Note</p> <p>The data type of the persistency data elements used in this modeling style must be an <code>AdaptivePlatformType</code>.</p> <p>Refer to How To Model Access to a Key-Value Pair of a Key-Value Storage on page 71.</p>

Port blocks carrying AUTOSAR specification

Note

TargetLink port blocks that carry Adaptive AUTOSAR specification for fields or events can be placed in any atomic subsystem in the subsystem hierarchy below Method Behavior subsystems and Adaptive AUTOSAR Function subsystems if they are connected as follows:

- Each of the blocks must be connected to a Simulink port block at the root level of the Adaptive AUTOSAR Function subsystem or Method Behavior subsystem.
- The signal lines must contain only Simulink port blocks.

Related topics

Basics	
Accessing Fields	58
Accessing Persistent Memory	71
Calling and Implementing Methods	66
Introduction to Using TargetLink With Adaptive AUTOSAR	10
Receiving and Sending Events	62

Basics on Service Discovery in TargetLink

Service discovery in TargetLink

TargetLink supports service discovery as described for `ara::com` by [Adaptive AUTOSAR](#) as follows:

- If you generate code for a model that contains model elements that specify service based communication, TargetLink can generate code for the service discovery. This includes code for instantiating proxy objects.
- You can specify which proxy objects are instantiated for a service instance and associate the proxy objects with individual model elements. Refer to [Associating proxy objects and model elements](#) on page 17.
- You can specify how the service instance is selected. Refer to [Service instance selection](#) on page 17.

You can specify the selection of service instances and proxy objects via DD `ServiceDiscovery` objects. These objects are child objects of DD `ServiceRequirePort` objects. A DD `ServiceDiscovery` object must be referenced at the following model elements:

- Function blocks whose Role property is set to `Method Call`.
- Port blocks that reference a DD `ServiceRequirePort` object.

For models that contain references to DD `ServiceDiscovery` objects, TargetLink generates code that combines the discovery of service instances and the instantiation of proxy objects for these service instances. The code looks similar to the following example:

```

/*****
*** FUNCTION:
***     PerformServiceDiscovery_MySoftwareComponent_AARSUBSYSTEM
***
*****/
bool PerformServiceDiscovery_MySoftwareComponent_AARSUBSYSTEM()
{
    bool DiscoveredAllServices = true;

    {
        auto HandleContainer = proxy::MyServiceInterfaceProxy::FindService(ara::com::InstanceIdentifier::Any);

        if (!HandleContainer.empty())
        {
            auto ServiceHandle = *HandleContainer.begin();
            ProxyContainer_MySoftwareComponent_AARSUBSYSTEM_Instance.p_MyServiceInterfaceDiscoveryAProxy =
            std::make_unique<proxy::MyServiceInterfaceProxy>(ServiceHandle);
        }
        else
        {
            DiscoveredAllServices = false;
        }
    }

    return DiscoveredAllServices;
}

```


Service instance selection

You specify how to select a service instance for which a proxy object is instantiated via the `SelectionCriteria` property of the DD `ServiceDiscovery` object:

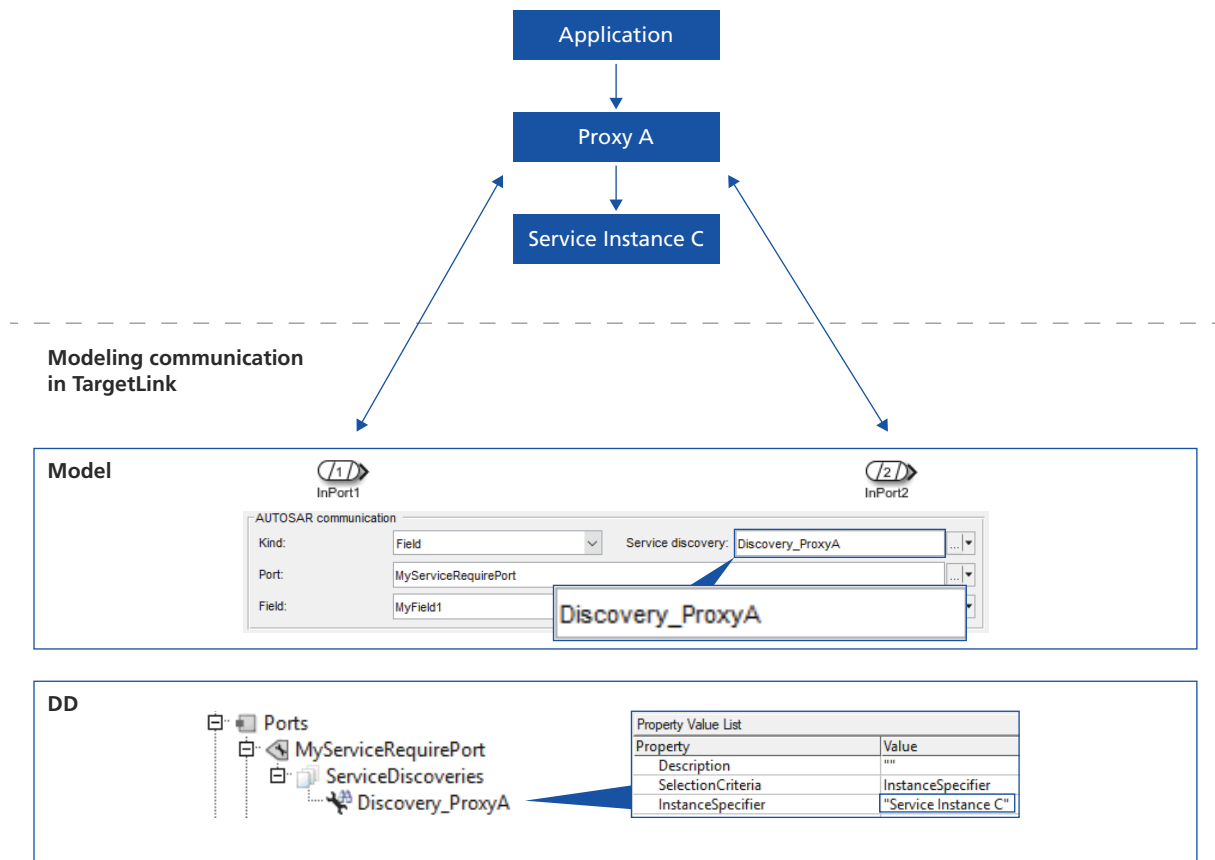
Use Case	SelectionCriteria Value	Description
Use any instance of a service	<code>FirstInContainer</code>	When you generate code, TargetLink instantiates a proxy object for the first instance in the handle container.
Use a fixed instance that is identified via a logical identifier (instance specifier)	<code>InstanceSpecifier</code>	The <code>InstanceSpecifier</code> property lets you specify an instance. When you generate code, TargetLink instantiates a proxy object for this instance.
Use a fixed instance that is identified via a technical identifier (instance identifier)	<code>InstanceIdentifier</code>	The <code>InstanceIdentifier</code> property lets you specify an instance. When you generate code, TargetLink instantiates a proxy object for this instance.
Use an instance from a custom service discovery	<code>None</code>	When you generate code, TargetLink does not instantiate a proxy object. You can implement your own service discovery. Refer to Custom service discovery on page 19.

Associating proxy objects and model elements

You associate a proxy object with an individual model element by referencing a DD `ServiceDiscovery` object at the model element. Different scenarios are possible:

One proxy object for a specific service instance To use the same proxy object at different model elements, you must reference the same DD `ServiceDiscovery` object at these model elements. This scenario is shown in the following example:

Communication diagram

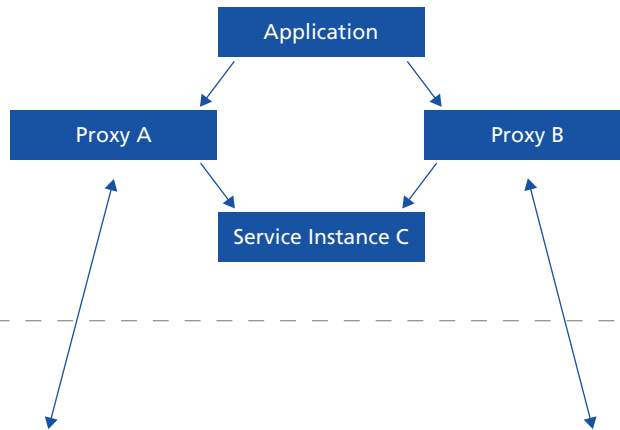


Multiple, different proxy objects for a specific service instance To use different proxy objects for the same service instance, you have to create one DD ServiceDiscovery object for each proxy object you want to use. The DD ServiceDiscovery objects must be configured to be connected with the same service instance. You can reference these DD objects at different model elements. You can do this, for example, via the InstanceSpecifier or InstanceIdentifier properties. In this case, the following properties of the DD ServiceDiscovery objects must be identical:

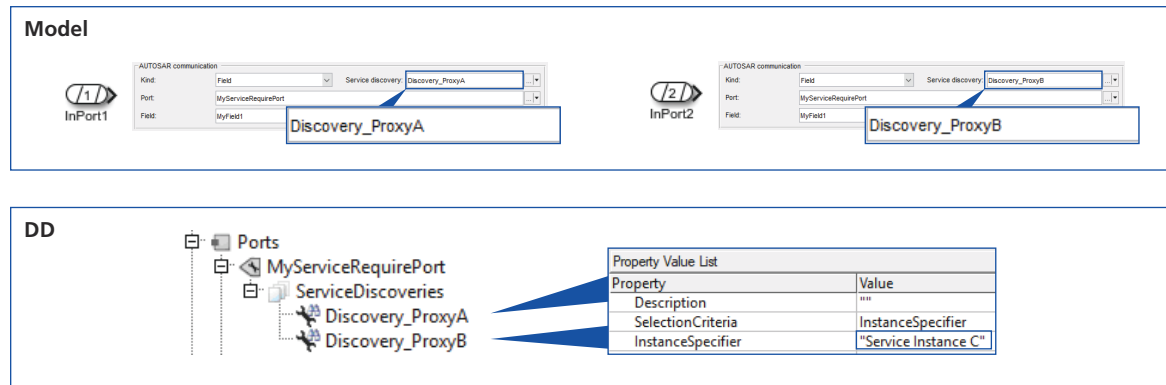
- SelectionCriteria (set to InstanceSpecifier or InstanceIdentifier)
- InstanceSpecifier or InstanceIdentifier

This scenario is shown in the following example:

Communication diagram



Modeling communication in TargetLink



Custom service discovery

You can use custom service discovery code and associate the resulting proxy objects with the TargetLink model as follows:

1. Create one DD ServiceDiscovery object for each custom proxy object to be used and set the DD ServiceDiscovery objects' SelectionCriteria property to **None**. Reference the DD ServiceDiscovery objects at the model elements at which you want to use custom proxy objects. Generate code for the project.
2. In your own custom service discovery code, assign each custom proxy object to a component of the structured proxy container variable. This variable is located inside the <SWC>_<CGU>.cpp module and is named ProxyContainer_<SWC>_<CGU>_Instance. The variable contains one struct component for each DD ServiceDiscovery object. The variable looks similar to the following example:

```

/*-----*\
  VARIABLES
/*-----*/

ProxyContainer_MySWC_MyCGU ProxyContainer_MySWC_MyCGU_Instance = {
    nullptr, /* .p_MyServiceInterfaceMyServiceDiscoveryProxy */
};

```

The type definition of this variable is generated into <SWC>_<CGU>.h and looks similar to the following example:

```

/*-----*\
  TYPEDEFS
/*-----*/

struct ProxyContainer_MySWC_MyCGU {
    std::unique_ptr p_MyServiceInterfaceMyServiceDiscoveryProxy;
};

```

The names of the struct components depend on the names of the DD ServiceDiscovery objects.

After assigning the custom proxy objects to the struct components, individual model elements are associated with a custom proxy object by referencing the corresponding DD ServiceDiscovery object. If you generate code for a model that contains these model elements, TargetLink uses the custom proxy objects in the production code.

Related topics

Basics

[Basics on Code Artifacts generated by TargetLink..... 20](#)

HowTos

[How to Specify Service Discovery and Instance Selection..... 51](#)

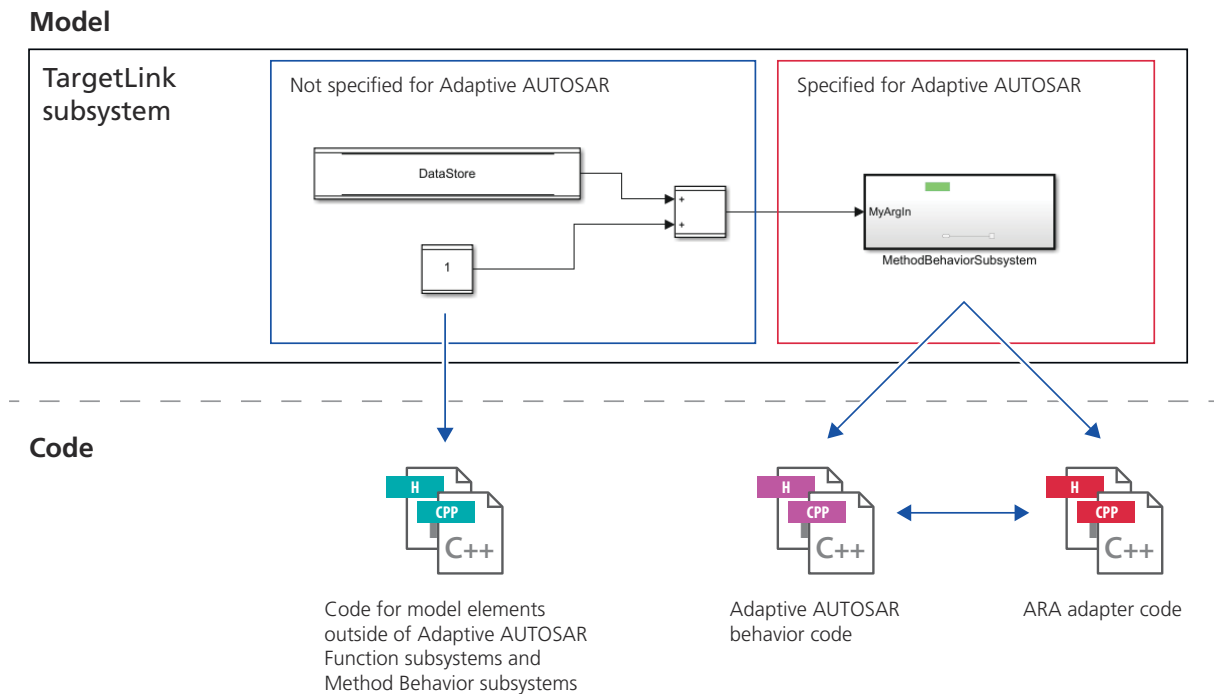
Basics on Code Artifacts generated by TargetLink

Code artifacts generated by TargetLink

If you generate code for models with Adaptive AUTOSAR specification, three different types of code modules are generated:

- Adaptive AUTOSAR behavior code (📄 in the illustration below)
- ARA adapter code (📄)
- Code for model elements outside Adaptive AUTOSAR Function subsystems and Method Behavior subsystems (📄)


The dependencies of these modules on the model are shown in the following illustration:



Adaptive AUTOSAR behavior code



The [Adaptive AUTOSAR behavior code](#) is generated from the model elements inside Adaptive AUTOSAR Function subsystems and Method Behavior subsystems. By default, this code is generated into a module that is referenced by a DD AdaptiveAutosarFunction object or a DD SoftwareComponent object. This code is part of the code for the adaptive application.

Depending on your model, this code can contain calls to functions that are defined in the [ARA adapter code](#) module (.

Code that is generated for model elements inside [Method Call subsystem](#) is not part of the Adaptive AUTOSAR behavior code. Refer to [Basics on SIL Simulation for Adaptive AUTOSAR Models](#) on page 28.

ARA adapter code



TargetLink generates code that connects the behavior code with the Adaptive AUTOSAR API into modules named `<SWC>_<CGU>.h/.cpp`. This code is part of the code for the adaptive application.

Code for model elements outside Adaptive AUTOSAR Function subsystems and Method Behavior subsystems




TargetLink generates code for model elements that do not reside inside or below a subsystem specified for Adaptive AUTOSAR. This code can be used for simulation purposes. This code is not part of the code for the adaptive application.



Exporting files

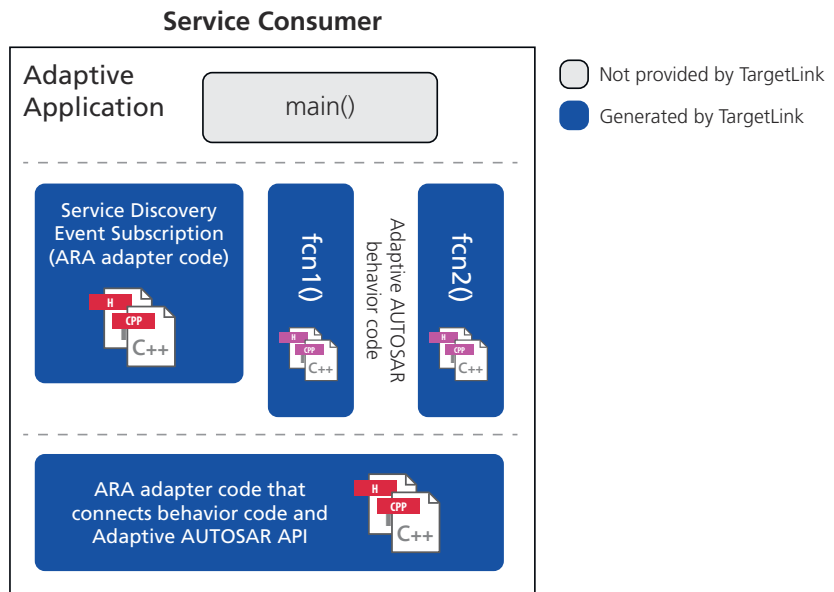
To integrate the modules generated by TargetLink into your adaptive application, it is recommended to export the generated files. To do this, use [t1_export_files](#) ([TargetLink API Reference](#)).

Note

If your model contains model elements outside of [Adaptive AUTOSAR Function subsystems](#) or [Method Behavior subsystems](#), the associated code modules () will be exported by `t1_export_files`. However, these modules are not required to integrate the behavior code into your application.

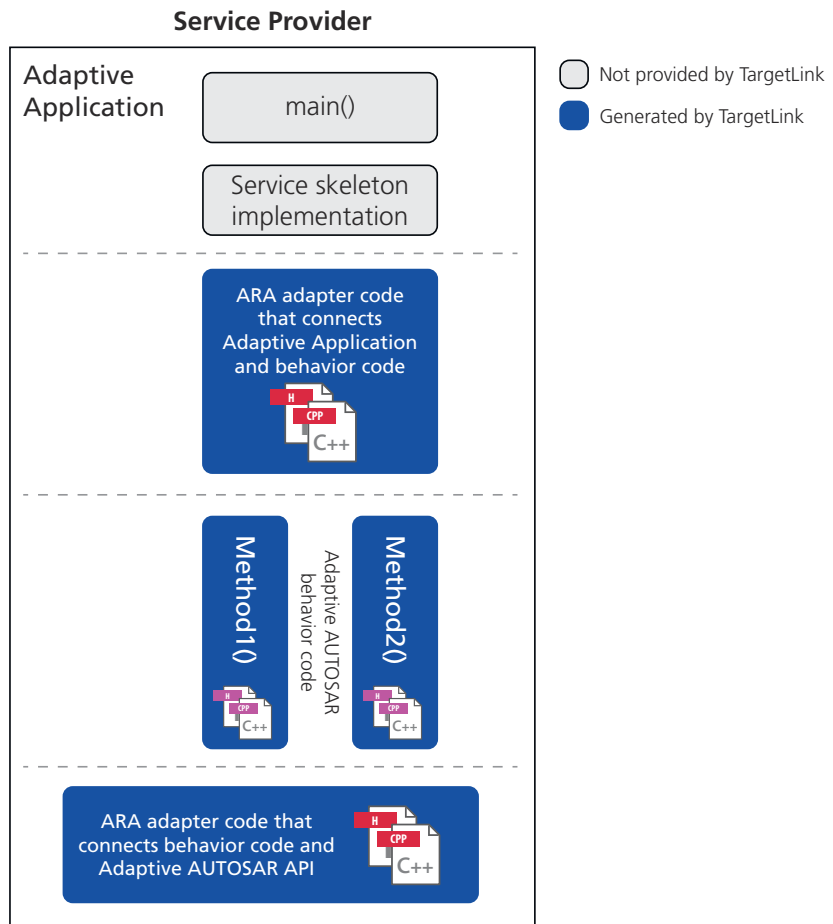
Integrating service consumer code

If your model includes model elements that are specified for the consumer side of service-based communication only, you can integrate the Adaptive AUTOSAR behavior code () directly into the application code you provide. The Adaptive AUTOSAR behavior code contains calls of functions that are defined in the module for the ARA adapter code (). The code for the service discovery and event subscription must be integrated into the application code you provide as well. This is shown schematically in the following illustration:



Integrating service provider code

If your model contains model elements that are specified for providing a service, e.g., the implementation of a method, you must integrate parts of the ARA adapter code (ARA) into the application code you provide. The ARA adapter code will ensure that the behavior code can be called correctly. This is shown schematically in the illustration below:



Related topics

Basics

Basics on Service Discovery in TargetLink.....	16
Basics on SIL Simulation for Adaptive AUTOSAR Models.....	28
Basics on Specifying the Location of Artifacts Generated by TargetLink (📖 TargetLink Customization and Optimization Guide)	

References

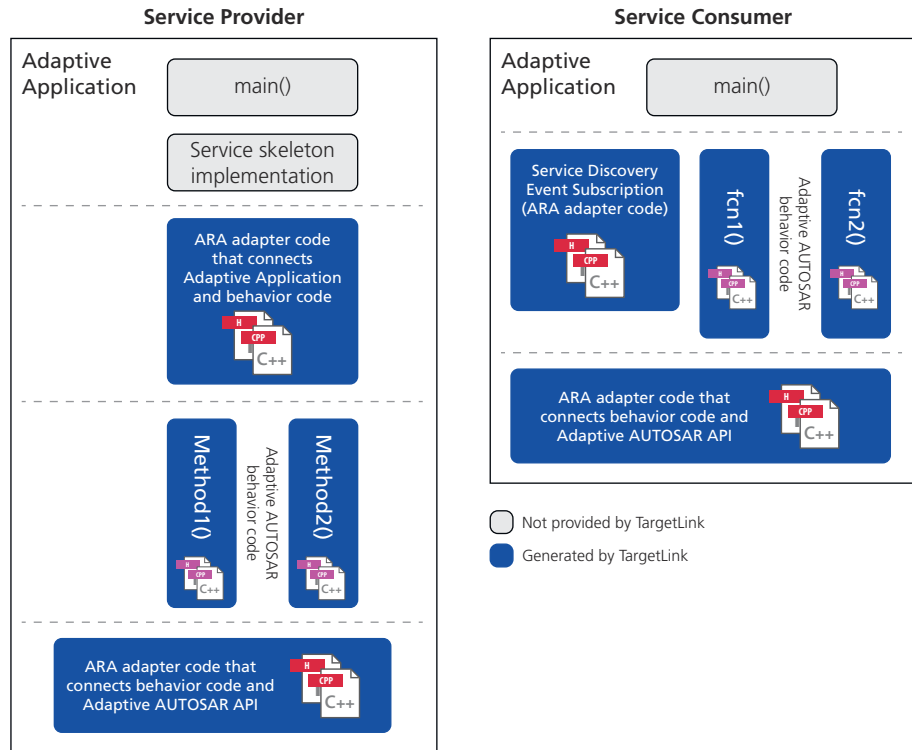
[tl_export_files](#) (📖 TargetLink API Reference)

Details on Integrating Adaptive AUTOSAR Code Artifacts

Using Adaptive AUTOSAR code

When you use [Adaptive AUTOSAR behavior code](#) and [ARA adapter code](#) for your Adaptive Application, you must ensure that certain requirements are met

when you integrate the code into your application. This includes integrating specific parts of the generated code in a certain order into the rest of the code of the Adaptive Application that is provided by you. The code you provide is depicted schematically as a `main()` function in the following simplified models of Adaptive Applications:



The behavior code, except the code for [Method Behavior subsystems](#), must be integrated directly into your Adaptive Application. For code from Method Behavior subsystems see the table below.

Depending on your model and the model elements that were used, you might have to integrate one or more of the following ARA adapter code parts in addition to the behavior code:

Interface - Port	Modeling situation	ARA adapter code
Service - Provide	Providing a method	Set skeleton instance on page 26 Method behavior on page 27
	Sending events	Set skeleton instance on page 26
	Updating fields	Set skeleton instance on page 26
Service - Require	Calling a method	Service discovery on page 26
	Receiving events	Service discovery on page 26 Event subscription on page 27
	Getting and setting fields	Service discovery on page 26

Interface - Port	Modeling situation	ARA adapter code
Persistency - Provide or Provide/Require	Writing the value of a key-value pair	Persistency access on page 27 ¹⁾
Persistency - Require or Provide/Require	Reading the value of a key-value pair	-

¹⁾ In this case, you do not need to integrate ARA adapter code, but you must ensure values are synced to the storage..

Service discovery

The ARA adapter code for the service discovery includes an adapter code function to perform the service discovery itself and a variable you must use if you implement a custom service discovery.

Performing service discovery This function performs the service discovery and instantiates proxy objects according to your specification. Refer to [Basics on Service Discovery in TargetLink](#) on page 16. The return value of this function is a Boolean that is true if all services were discovered and a proxy object could successfully be instantiated.

You can identify this function in the module with the default name `<SWC>_<CGU>.cpp` via its name:

```
bool PerformServiceDiscovery_AdaptiveSwc_S1()
{
    // Code for the service discovery and proxy instantiation
}
```

This function must be integrated into the code you provide. This function must be called before any service-based communication on the service consumer side is carried out.

Implementing custom service discovery If you use TargetLink code in conjunction with custom service discovery code, you must use a proxy container variable provided by TargetLink to associate proxy objects with model elements and their resulting code. Refer to [Basics on Service Discovery in TargetLink](#) on page 16.

You can identify this function in the module with the default name `<SWC>_<CGU>.cpp` via its name:

```
ProxyContainer_AdaptiveSwc_S1 ProxyContainer_AdaptiveSwc_S1_Instance = {
    nullptr, /* .p_IF_ServiceBServiceDiscoveryProxy */
};
```

Set skeleton instance

If you use TargetLink code in an application that provides a service, you must initialize the service skeleton yourself with code that you provide. Additionally, you must associate the skeleton instance with the code provided by TargetLink.

This can be done via the following ARA adapter code function in the module with the default name `<SWC>_<CGU>.cpp`:

```
void SetSkeletonInstance_PP_ServiceA_IF_ServiceA_AdaptiveSwc_S1(skeleton::IF_ServiceASkeleton * p_SkeletonInstance)
{
    SkeletonContainer_AdaptiveSwc_S1_Instance.p_PP_ServiceAIF_ServiceASkeleton = p_SkeletonInstance;
}
```

This function must be invoked before other ARA adapter code functions for the provided service are called.

Event subscription

If you model receiving events, you must integrate the code for the subscription to the associated proxy object.

You can identify the adapter code function provided for this in the module with the default name `<SWC>_<CGU>.cpp` via its name:

```
void Subscribe_AdaptiveSwc_S1()
{
    // Code for event subscription
}
```

This function must be integrated into the code you provide. This function can only be invoked *after* proxy objects are available, i.e., after you invoked the adapter code function for performing the service discovery or after you assigned the result of your custom service discovery to the proxy container components. The **Subscribe** adapter code function must be invoked *before* adapter code functions for receiving events via this proxy object are called.

Method behavior

If you model a functionality in a Method Behavior subsystem, a function that contains the behavior code for the subsystem is generated. Refer to [Basics on Code Artifacts generated by TargetLink](#) on page 20.

Depending on your model, this behavior code can include calls to ARA adapter code functions, for example, to receive events.

In addition, an ARA adapter code function for the Method Behavior subsystem is generated. This adapter code function must be integrated in the code you provide and will in turn call the function that contains the behavior code.

You can identify this adapter code function in the module with the default name `<SWC>_<CGU>.cpp` via its name:

```
ara::core::Future<IF_ServiceA::MethodAOutput> Method_AdaptiveSwc_PP_ServiceA_MethodA(IDT_int32 in)
{
    // adapter code
}
```

When you integrate the adapter code function into your code, you must ensure that the dependencies of the behavior code that is called from this adapter code function are satisfied. For example, if the Method Behavior subsystem models receiving events, the service discovery and event subscription must have been performed before you call the behavior code from the Method Behavior subsystem via its adapter code function.

Persistency access

If you model the access to a key-value pair of a key-value storage, the generated adapter code contains functions to get and/or set values of a key-value pair:

```
void GetValue_AdaptiveSwc_PRP_PersistencyData_Data1_S1(int32_t * OutParam)
{
    auto KVSDb = GetKeyValueStorage_AdaptiveSwc_PRP_PersistencyData_S1();
    auto PerResult = KVSDb->GetValue<int32_t>("Data1");

    if (PerResult)
    {
        *OutParam = PerResult.Value();
    }
}
```

These adapter code functions call another adapter code function that opens a key-value storage the first time it is called:

```
static ara::per::SharedHandle<ara::per::KeyValueStorage>& GetKeyValueStorage_AdaptiveSwc_PRP_PersistencyData_S1()
{
    static auto KVSDb =
ara::per::OpenKeyValueStorage(ara::core::InstanceSpecifier("S1/AdaptiveSwc/PRP_PersistencyData")).ValueOrThrow();
    return KVSDb;
}
```

After writing to a key-value storage via the adapter code functions provided by TargetLink, you must synchronize the changed values with the values in the storage via the `SyncToStorage` method specified by the [Adaptive AUTOSAR](#) standard.

Related topics

Basics

Basics on Code Artifacts generated by TargetLink.....	20
Basics on Service Discovery in TargetLink.....	16
Overview of Supported Modeling Styles for Adaptive AUTOSAR.....	12

Basics on SIL Simulation for Adaptive AUTOSAR Models

Introduction

In general, the [SIL simulation](#) mode of TargetLink lets you simulate your [production code](#) on the host PC. For Adaptive AUTOSAR models, the production code includes [Adaptive AUTOSAR behavior code](#) as well as [ARA adapter code](#). The ARA adapter code connects behavior code with the APIs of functional clusters as defined by Adaptive AUTOSAR.

SIL simulation for Adaptive AUTOSAR models When building the [simulation application](#) for SIL simulation of Adaptive AUTOSAR models, TargetLink uses preprocessor macros and conditional compilation to replace parts of the ARA adapter code. Refer to [Conditional compilation of ARA adapter code](#) on page 29.

To avoid the preprocessor macros in the production code, select the **Clean code** option on the Code Generation page of the TargetLink Main Dialog. Building a SIL simulation application is not possible if this option is active.

Preconditions

During the build process of the simulation application for Adaptive AUTOSAR models, the production code is compiled as C++ code by TargetLink. Thus, the file name extension of all modules, including the modules you created, must be `.cpp`.

Conditional compilation of ARA adapter code

Preprocessor macros are generated in the ARA adapter code to enable conditional compilation during the build process of the simulation application. In general, the conditional compilation replaces the code that relates to the functional cluster with a simulation variable as seen in the following example:

```
void Get_SWC_MyServiceRequirePort_ScalarField_MyDiscovery_Subsystem(ScalarIdt * OutParam)
{
    #ifndef __SIL__ // Code that relates to the ara::com API
        if (ProxyContainer_SWC_Subsystem_Instance.p_MyServiceInterfaceMyDiscoveryProxy != nullptr)
        {
            auto SyncFuture = ProxyContainer_SWC_Subsystem_Instance.p_MyServiceInterfaceMyDiscoveryProxy->ScalarField.Get();
            auto Result = SyncFuture.get();

            *OutParam = Result;
        }
    #else // Simulation variable
        *OutParam = Field_SWC_MyServiceRequirePort_ScalarField_MyDiscovery_Subsystem;
    #endif
}
```

Method Call subsystems For [Method Call subsystems](#), the corresponding ARA adapter code contains a call to a method provided by a service interface. When building the simulation application for SIL simulation, this call is replaced with a call to a simulation function via preprocessor macros and conditional compilation:

```
void Call_SWC_R_Port_MyServiceRequirePort_MyMethod_MyDiscovery_Subsystem(ScalarIdt MyARGIN, ScalarIdt * MyARGOUT)
{
    #ifndef __SIL__ // ARA adapter code for calling the method
        if (ProxyContainer_SWC_R_Port_Subsystem_Instance.p_MyServiceInterfaceMyDiscoveryProxy != nullptr)
        {
            auto SyncFuture =
                ProxyContainer_SWC_R_Port_Subsystem_Instance.p_MyServiceInterfaceMyDiscoveryProxy->MyMethod(MyARGIN);
            auto Result = SyncFuture.get();

            *MyARGOUT = Result.MyARGOUT;
        }
    #else // Call of the simulation function
        Call_SWC_R_Port_MyServiceRequirePort_MyMethod_MyDiscovery_Subsystem_frm(MyARGIN, MyARGOUT);
    #endif
}
```

The simulation function is generated according to the modeling in the Method Call subsystem. This lets you simulate the behavior of a method provided by a service for service consumer modeling.

MIL/SIL differences

Differences between [MIL](#) and SIL simulation can occur because initial values for data elements are not available for Adaptive AUTOSAR models.

Limitations TargetLink supports SIL simulation for Adaptive AUTOSAR components only for a single [code generation unit \(CGU\)](#).

Related topics

Basics	
Basics on Code Artifacts generated by TargetLink.....	20
Basics on Service Discovery in TargetLink.....	16
Overview of Supported Modeling Styles for Adaptive AUTOSAR.....	12

Specifying Data Types

Where to go from here

Information in this section

Introduction to Data Types in Adaptive AUTOSAR.....	32
Defining Scalings and Constrained Range Limits (Adaptive AUTOSAR).....	35
Creating CPP Implementation Data Types From Scratch.....	36
Creating Application Data Types (Adaptive AUTOSAR).....	43

Introduction to Data Types in Adaptive AUTOSAR

Introduction to Data Types in Adaptive AUTOSAR

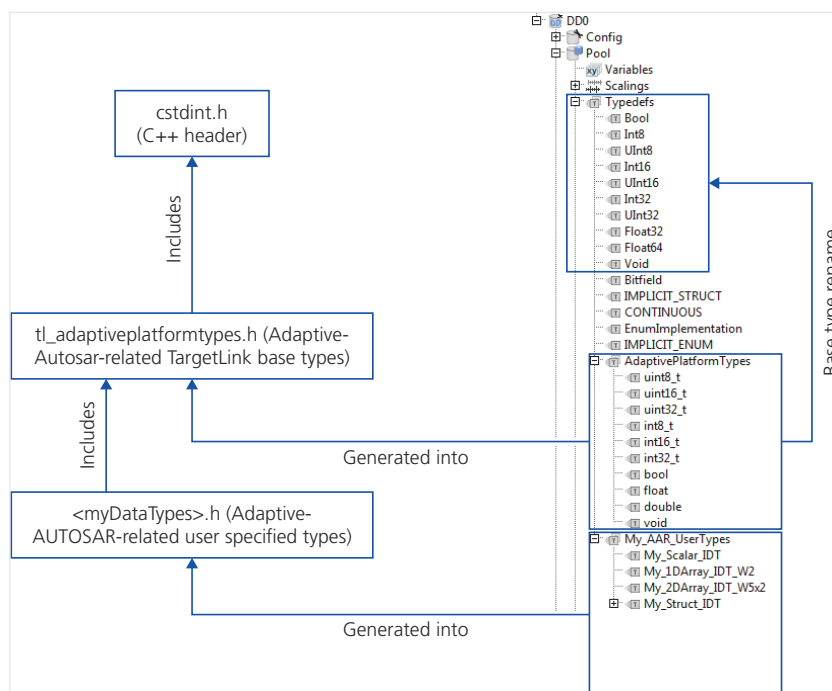
Data types as described by Adaptive AUTOSAR

Adaptive AUTOSAR defines several C++ data types that can be used in service interface descriptions (ARXML files). Within these files, these data types are described via primitive `STD-CPP-IMPLEMENTATION-DATA-TYPE` elements.

Platform data types in TargetLink

TargetLink provides a predefined DD Typedef object for each primitive `StdCppImplementationDataType` it supports. These objects are contained in the `dsdd_master_adaptive_autosar.dd` [System] DD template in the DD TypedefGroup object called `/Pool/Typedefs/AdaptivePlatformTypes`.

The DD Typedef objects are based on TargetLink's base types (via the DD `BaseTypeRename` property) and are generated into a predefined module that is called `tl_adaptiveplatformtypes`. This is shown in the following illustration:



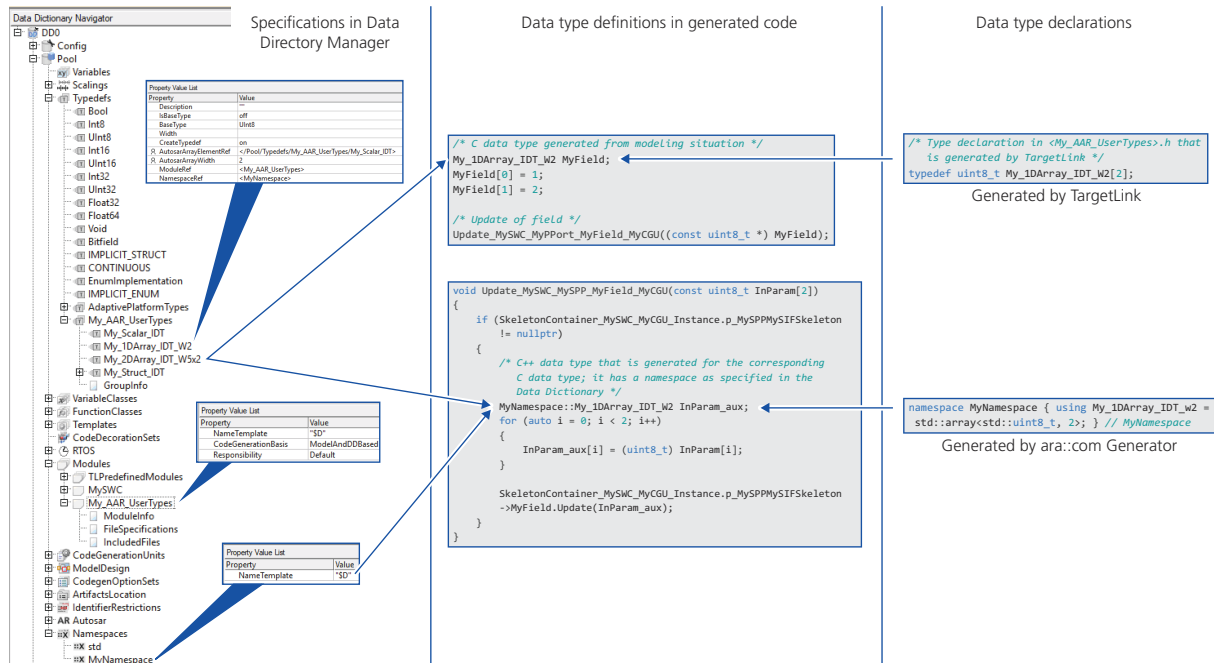
Additionally, the above illustration shows some additional DD Typedef objects used to specify user data types that are generated into a user-specified module `<myDataTypes>.h`.

User-specified data types

You can create additional DD Typedef objects to specify user data types.

During code generation, TargetLink creates two corresponding type definitions for each DD Typedef object:

- A C data type that is used in the [Adaptive AUTOSAR behavior code](#).
- A C++ data type that is used in the C++ functions to be integrated into an adaptive application.



Note

Constraints for DD Typedef objects used at service interfaces

All of the following conditions must be fulfilled for DD Typedef objects that are used for user-specified data types for modeling [Adaptive AUTOSAR](#):

- Their `CreateTypedef` property must be set to `on`.
- Their `BaseType` property must be set to a suitable base type.
- They must have a [module specification](#). The corresponding DD Module object must have its `CodeGenerationBasis` property set to `ModelAndDDBased`. TargetLink uses this module to collect user-specified C data types that are used in the internal behavior.

Specifying namespaces

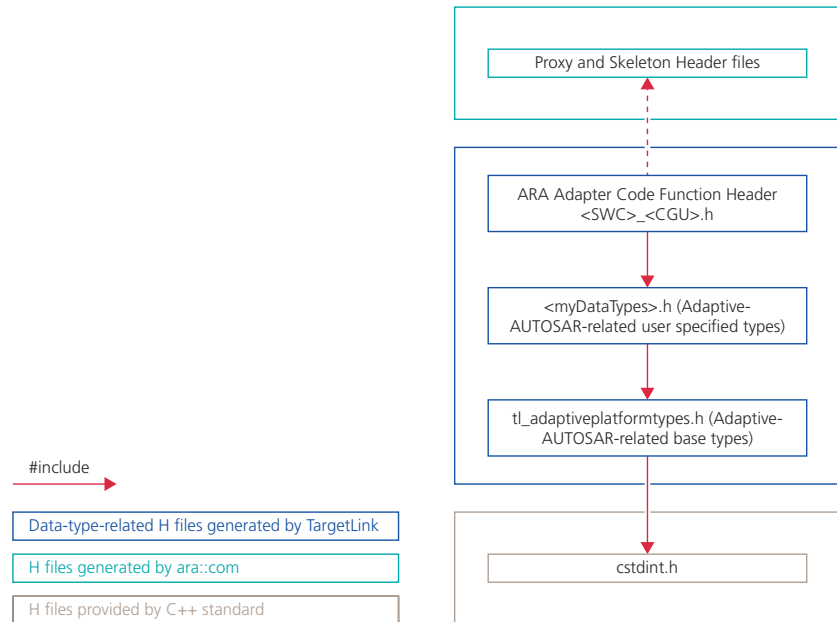
If your user-specified data types are defined within a namespace, you must specify this namespace within the Data Dictionary:

- Create and specify a `/Pool/Namespaces/<Namespace>` object.
- Reference it at the desired DD Typedef objects.

During code generation, TargetLink uses this namespace in the definitions of the C++ data types.

Dependencies of data-type-specific header files

TargetLink makes sure that the dependencies of the data-type-specific header files are correct. A typical dependency-relationship looks as follows:



Limitations

For limitations concerning data types when modeling for [Adaptive AUTOSAR](#), refer to [AUTOSAR Import/Export of data types](#) ([TargetLink Limitation Reference](#)).

Related topics


HowTos

[How to Define Scalar Implementation Data Types For Adaptive AUTOSAR](#)..... 36

Defining Scalings and Constrained Range Limits (Adaptive AUTOSAR)


Basics on Scalings and Constrained Range Limits in Adaptive AUTOSAR

Introduction

Creating scalings and constrained range limits in Adaptive AUTOSAR works the same way as in Classic AUTOSAR. Refer to [Basics on Scalings and Constrained Range Limits \(Classic AUTOSAR\)](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#)).

Related topics

Basics

[Basics on Scalings and Constrained Range Limits \(Classic AUTOSAR\)](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#))

Creating CPP Implementation Data Types From Scratch

Where to go from here

Information in this section

How to Define Scalar Implementation Data Types For Adaptive AUTOSAR.....	36
How to Define 1-D Array Implementation Data Types for Adaptive AUTOSAR.....	37
How to Define 2-D Array Implementation Data Types for Adaptive AUTOSAR.....	39
How to Define Structured Implementation Data Types for Adaptive AUTOSAR.....	40

How to Define Scalar Implementation Data Types For Adaptive AUTOSAR

Objective

To define scalar implementation data types for [Adaptive AUTOSAR](#), you must create a DD Typedef object and specify it as described below.

Note

In TargetLink, STD-CPP-IMPLEMENTATION-DATA-TYPE elements are represented as DD Typedef objects.

Preconditions

- DD project files that are used for modeling according to Adaptive AUTOSAR must be based on the dsdd_master_adaptive_autosar.dd [System] DD system template.
- You created a DD Module object whose CodeGenerationBasis property is set to **ModelAndDDBased**. For details, refer to [Introduction to Data Types in Adaptive AUTOSAR](#) on page 32.

Method

To define a scalar implementation data type for Adaptive AUTOSAR

- 1 Create a DD TypedefGroup object in /Pool1/Typedef and name it as required. For example, My_AAR_UserTypes.
- 2 Add the GroupInfo subtree to the DD TypedefGroup object and specify a package, such as SharedElements/CPPIImplementationDataTypes, via its Package property.

- 3 Add a DD Typedef object to the DD TypedefGroup object and name it as required, e.g., `My_Scalar_IDT`.
- 4 In the DD Typedef object's Property Value List, specify the following settings:

Property	Value
IsBaseType	off
BaseType	Select a base type from the list, e.g., <code>UInt8</code> .
CreateTypedef	on
ModuleRef	Reference to the DD Module object to which you want to generate your type definitions, e.g., <code>MY_AAR_UserTypes</code> .
NamespaceRef	If your implementation data type belongs to a namespace, reference a suitable DD Namespace object. TargetLink uses this namespace in the definition of the C++ data type.

Result

You created a DD Typedef object to define a scalar implementation data type for Adaptive AUTOSAR.

TargetLink generates a suitable definition in the header file called `MY_AAR_UserTypes`:

```
typedef uint8_t My_Scalar_IDT;
```

Related topics

Basics

[Introduction to Data Types in Adaptive AUTOSAR](#)..... 32

How to Define 1-D Array Implementation Data Types for Adaptive AUTOSAR

Objective

To define 1-D array implementation data types for [Adaptive AUTOSAR](#), you must create a DD Typedef object and specify it as described below.

Note

In TargetLink, `STD-CPP-IMPLEMENTATION-DATA-TYPE` elements are represented as DD Typedef objects.

Precondition

- You created a scalar implementation data type as described in [How to Define Scalar Implementation Data Types For Adaptive AUTOSAR](#) on page 36.

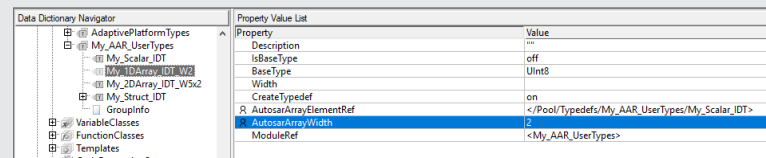
Method**To define a 1-D array implementation data type for Adaptive AUTOSAR**

- 1 Open the context menu of the DD TypedefGroup object that contains your implementation data types for [Adaptive AUTOSAR](#).
- 2 From the AUTOSAR submenu, choose **Create AUTOSAR Array Type** and rename the resulting implementation data type as required, e.g., **My_Array_IDT_W2**.
- 3 In its Property Value List, specify the following settings:

Property	Value
IsBaseType	off
BaseType	Select a base type from the list, e.g., UInt8 .
CreateTypedef	on
AutosarArrayElementRef	Select a scalar implementation data type for the array elements that is compatible with the selected base type.
AutosarArrayWidth	Specify the width of the array.
ModuleRef	Reference to the DD Module object to which you want to generate your type definitions, e.g., MY_AAR_UserTypes .
NamespaceRef	If your implementation data type belongs to a namespace, reference a suitable DD Namespace object. TargetLink uses this namespace in the definition of the C++ data type.

Note

You have to specify a consistent width for interface elements and internal behavior elements that are of the defined array type. For example, if you want to create a field of the **My_1DArray_IDT_W2** type, you have to specify 2 for the DD ArField object's Width property.

**Result**

You created a DD Typedef object to define a 1-D array implementation data type for Adaptive AUTOSAR.

TargetLink generates a suitable definition in the header file called **MY_AAR_UserTypes**:

```
typedef uint8_t My_1DArray_IDT_W2[2];
```

Related topics

Basics

Introduction to Data Types in Adaptive AUTOSAR..... 32

How to Define 2-D Array Implementation Data Types for Adaptive AUTOSAR

Objective

To define 2-D array implementation data types for [Adaptive AUTOSAR](#), you must create a DD Typedef object and specify it as described below.

Note

In TargetLink, STD-CPP-IMPLEMENTATION-DATA-TYPE elements are represented as DD Typedef objects.

Precondition

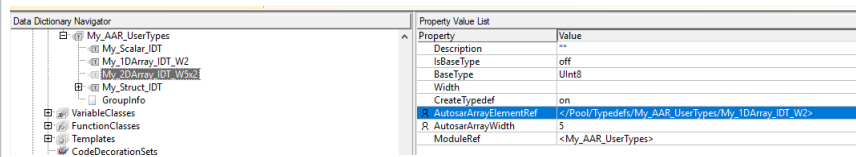
- You created an array data type as described in [How to Define 1-D Array Implementation Data Types for Adaptive AUTOSAR](#) on page 37.

Method

To create a 2-D array implementation data type for Adaptive AUTOSAR

- Open the context menu of the DD TypedefGroup object that contains your implementation data types for [Adaptive AUTOSAR](#).
- From the AUTOSAR submenu, choose Create AUTOSAR Array Type and rename the resulting CPP implementation data type as required, e.g., `My_2DArray_IDT_W5x2`.
- In its Property Value List, specify the following settings:

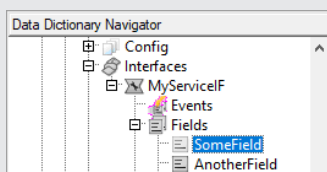
Property	Value
Is BaseType	off
Base Type	Select a base type from the list.
Create Typedef	on
AutosarArrayElementRef	Select an 1-D array implementation data type for the array elements that is compatible with the selected base type. This DD Typedef object's AutosarArrayWidth property determines the array's second dimension.
AutosarArrayWidth	Specify the dimensionality of the array's first dimension as a scalar value. The second dimension results from the DD Typedef object's settings that is referenced via the AutosarArrayElementRef property.



Property	Value
ModuleRef	Reference to the DD Module object to which you want to generate your type definitions, e.g., MY_AAR_UserTypes.
NamespaceRef	If your implementation data type belongs to a namespace, reference a suitable DD Namespace object. TargetLink uses this namespace in the definition of the C++ data type.

Note

You have to specify a consistent width for interface elements and internal behavior elements that are of the defined array type. For example, if you want to create a field of the My_2DArray_IDT_W5x2 type, you have to specify [5 2] for the DD ArField object's Width property.

Data Dictionary Navigator	Property Value List														
	<table> <tr> <th>Property</th><th>Value</th></tr> <tr> <td>Description</td><td>""</td></tr> <tr> <td>HasGetter</td><td>on</td></tr> <tr> <td>HasNotifier</td><td>off</td></tr> <tr> <td>HasSetter</td><td>on</td></tr> <tr> <td>Type</td><td><My_AAR_UserTypes/My_2DArray_IDT_W5x2></td></tr> <tr> <td>Width</td><td>[5 2]</td></tr> </table>	Property	Value	Description	""	HasGetter	on	HasNotifier	off	HasSetter	on	Type	<My_AAR_UserTypes/My_2DArray_IDT_W5x2>	Width	[5 2]
Property	Value														
Description	""														
HasGetter	on														
HasNotifier	off														
HasSetter	on														
Type	<My_AAR_UserTypes/My_2DArray_IDT_W5x2>														
Width	[5 2]														

Result

You created a DD Typedef object to define a 2-D array implementation data type for Adaptive AUTOSAR.

TargetLink generates a suitable definition in the header file called MY_AAR_UserTypes:

```
typedef uint8_t My_2DArray_IDT_W5x2[5][2];
```

Related topics**Basics**

[Introduction to Data Types in Adaptive AUTOSAR](#)..... 32

How to Define Structured Implementation Data Types for Adaptive AUTOSAR

Objective

To define structured implementation data types for [Adaptive AUTOSAR](#), you must create a DD Typedef object and specify it as described below.

Note

In TargetLink, STD-CPP-IMPLEMENTATION-DATA-TYPE elements are represented as DD Typedef objects.

Preconditions

- DD project files that are used for modeling according to Adaptive AUTOSAR must be based on the dsdd_master_adaptive_autosar.dd [System] DD system template.
- You created a DD Module object whose CodeGenerationBasis property is set to **ModelAndDDBased**. For details, refer to [Introduction to Data Types in Adaptive AUTOSAR](#) on page 32.

Method**To define a structured implementation data type for Adaptive AUTOSAR**

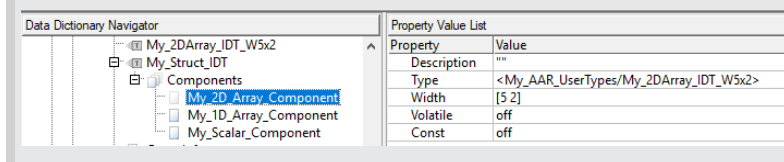
- 1 Create a DD TypedefGroup object in /Pool1/Typedef and name it as required. For example, **My_AAR_UserTypes**.
- 2 Add the GroupInfo subtree to the DD TypedefGroup object and specify a package, such as **SharedElements/CPPIImplementationDataTypes**, via its Package property.
- 3 Add a DD Typedef object to the DD TypedefGroup object and name it as required, e.g., **My_Struct_IDT**.
- 4 In its Property Value List, specify the following settings:

Property	Value
IsBaseType	off
BaseType	Struct
CreateTypedef	on
ModuleRef	Reference to the DD Module object to which you want to generate your type definitions, e.g., MY_AAR_UserTypes .
NamespaceRef	If your implementation data type belongs to a namespace, reference a suitable DD Namespace object. TargetLink uses this namespace in the definition of the C++ data type.

- 5 From the context menu of the DD Typedef object, select **Create Components**.
- 6 From the context menu of the Components subtree, select **Create Component**.
- 7 In the component's Property Value List, select an adaptive platform type or a user-specified type.

Note

If you select an array type as a component of the struct type, you have to specify a consistent Width for the component.



Result

You created a DD Typedef object to define a structured implementation data type for Adaptive AUTOSAR.

TargetLink generates a suitable definition in the header file called **MY_AAR_UserTypes**:

```
typedef struct My_Struct_IDT_tag {
    My_2DArray_IDT_W5x2 My_2D_Array_Component;
    My_1DArray_IDT_W2 My_1D_Array_Component;
    My_Scalar_IDT My_Scalar_Component;
} My_Struct_IDT;
```

Related topics

Basics

[Introduction to Data Types in Adaptive AUTOSAR..... 32](#)

Creating Application Data Types (Adaptive AUTOSAR)

Creating Application Data Types from Scratch (Adaptive AUTOSAR)

Introduction

Just like [Classic AUTOSAR](#), [Adaptive AUTOSAR](#) distinguishes between [implementation data types](#) and [application data types](#).

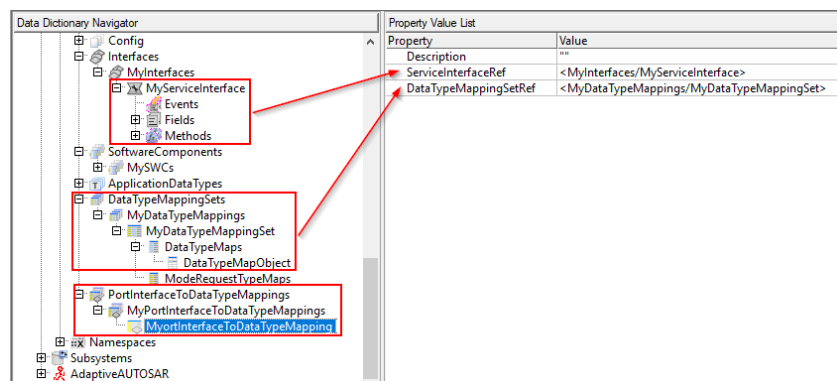
Creating application data types

You create [application data types](#) for [Adaptive AUTOSAR](#) in the same manner as application data types for [Classic AUTOSAR](#). Refer to [Creating Application Data Types from Scratch \(Classic AUTOSAR\)](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

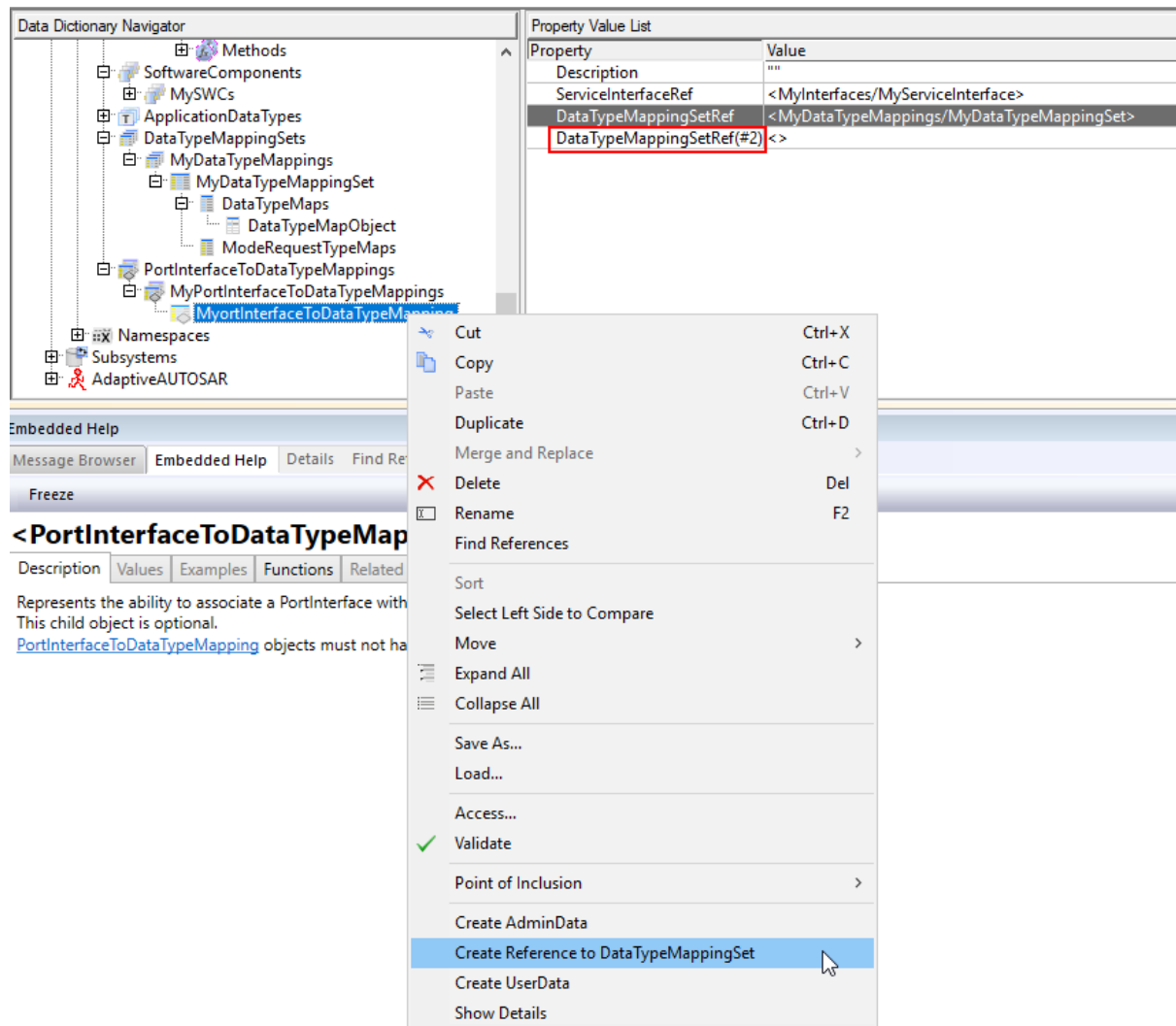
Mapping IDTs, ADTs and interfaces

[Implementation data types](#) and [application data types](#) must be mapped. As in [Classic AUTOSAR](#), this works via DD `DataTypeMappingSet` and DD `DataTypeMap` objects.

In [Adaptive AUTOSAR](#), data type mapping sets are mapped to service interfaces via `PORT-INTERFACE-TO-DATA-TYPE-MAPPING` elements. In the Data Dictionary, this is represented by DD `PortInterfaceToDataTypeMapping` objects that can be created in the `/Pool/Autosar/PortInterfaceToDataTypeMappings` subtree. Each DD `PortInterfaceToDataTypeMapping` object can reference exactly one DD `ServiceInterface` object via the `ServiceInterfaceRef` property and one or more DD `DataTypeMappingSet` objects via the `DataTypeMappingSetRef(<n>)` property:



If you want to reference several DD `DataTypeMappingSet` objects, you can add additionally reference properties via the context menu:



Related topics

Basics

Basics on Implementation and Application Data Types (Classic AUTOSAR) (📖 TargetLink Classic AUTOSAR Modeling Guide)
 Basics on Working with Implementation and Application Data Types (Classic AUTOSAR) (📖 TargetLink Classic AUTOSAR Modeling Guide)

Preparatory Steps

Where to go from here

Information in this section

How to Start Modeling from Scratch.....	45
How to Create Software Components for Adaptive AUTOSAR.....	46
How to Specify an Adaptive AUTOSAR Function.....	47
How To Create Interfaces (Adaptive AUTOSAR).....	48
How to Create Ports.....	49
How to Specify Service Discovery and Instance Selection.....	51
How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Create Communication Subjects for Adaptive AUTOSAR.....	53


How to Start Modeling from Scratch

Objective

To start modeling according to Adaptive AUTOSAR from scratch, you must create a TargetLink model and a proper DD project file.


Preconditions

The following preconditions must be fulfilled:

- You have installed TargetLink including the TargetLink Adaptive AUTOSAR Module.
- You created a TargetLink model as described in [How to Create a Model from Scratch](#) ( [TargetLink Orientation and Overview Guide](#)).

Method**To start modeling from scratch**

- 1 From the model's TargetLink menu, choose Data Dictionary Manager.
The TargetLink Data Dictionary Manager opens.
- 2 From the File menu of the TargetLink Data Dictionary Manager, choose New - Use Current DD Workspace and create a DD project file based on the dsdd_master_adaptive_autosar.dd [System] system template.
- 3 From the File menu of the Data Dictionary Manager, choose Save As ... and save the DD project file.
- 4 Open the TargetLink Main Dialog and specify the following settings:

Property	Value
CodeGenerationMode (Model element) ( TargetLink Model Element Reference) (on the Code Generation page of the TargetLink Main Dialog Block)	Adaptive AUTOSAR
Associate model with fixed DD project file (on the Options page of the TargetLink Main Dialog Block)	Specify the DD project file created and saved above.

Result

You have created a DD project file and specified a previously created subsystem for modeling according to Adaptive AUTOSAR.

Related topics**References**

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

How to Create Software Components for Adaptive AUTOSAR

Objective

To model a functionality, you must define a software component.

Preconditions

The following preconditions must be fulfilled:

- You have opened a TargetLink model with a Data Dictionary for the Adaptive AUTOSAR use case. Refer to [How to Start Modeling from Scratch](#) on page 45.

Method**To create software components**

- 1 In the Data Dictionary Navigator, right-click the /Pool/Autosar/SoftwareComponents/ subtree.

- 2 Choose **Create SoftwareComponentGroup** from the context menu.
- 3 Enter a name for the software component group, for example, **ComponentType**.
- 4 Right-click the software component group and choose **Create GroupInfo** from the context menu.
- 5 In the **Property Value List**, enter a package name, for example, **ComponentType**. This lets you export software components in the group to the specified package. Software components in AUTOSAR files located in the package can be imported to that location.
- 6 Right-click the software component group and choose **Create SoftwareComponent** from the context menu.
- 7 Enter a name for the new software component, for example, **my_SWC**.
- 8 In the **Property Value List**, specify the following settings that directly influence code generation:

Property	Value
ComponentType	Set the value to <code>AdaptiveApplicationSoftwareComponent</code> .

Result

You have created a software component for use in an Adaptive AUTOSAR model.

How to Specify an Adaptive AUTOSAR Function

Objective

To specify an [Adaptive AUTOSAR Function](#), you must create a DD `AdaptiveAutosarFunction` object.

Preconditions

The following preconditions must be fulfilled:

- You created a software component. Refer to [How to Create Software Components for Adaptive AUTOSAR](#) on page 46.
- You created and specified a DD Module object.

Method**To specify an Adaptive AUTOSAR function**

- 1 From the DD `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>` object's context menu, select **Create AdaptiveAutosarFunctions**.
- 2 From the `AdaptiveAutosarFunctions` object's context menu, select **Create AdaptiveAutosarFunction**.
- 3 Rename the created DD `AdaptiveAutosarFunction` object as required.

- 4 In the DD AdaptiveAutosarFunction object's Property Value List specify the properties as shown in the following table:


Property	Value
FunctionClassRef	Select a DD FunctionClass object that fits your use case.
ModuleRef	Select a module for the Adaptive AUTOSAR Function's code.
NameTemplate	Select a name template for the Adaptive AUTOSAR function.

Result

You specified an Adaptive AUTOSAR function's module and function class.

Related topics

Basics

[Basics on Modules and Module Ownership](#) ( TargetLink Customization and Optimization Guide)

HowTos

[How to Create Module Specifications in the Data Dictionary](#) ( TargetLink Customization and Optimization Guide)
[How to Create Software Components for Adaptive AUTOSAR](#)..... 46

How To Create Interfaces (Adaptive AUTOSAR)

Objective

To create an interface according to [Adaptive AUTOSAR](#) in TargetLink, you must create a DD <Interface> object.

Interfaces

Interfaces must be created for the following kinds of AUTOSAR communication:

AUTOSAR Communication Kind	DD <Interface> Object	Communication Subject
Event	/Pool/Autosar/Interfaces/<ServiceInterface>	ArEvent
Field	/Pool/Autosar/Interfaces/<ServiceInterface>	ArField
Operation	/Pool/Autosar/Interfaces/<ServiceInterface>	Operation
PersistencyDatabase	/Pool/Autosar/Interfaces/<PersistencyKeyValueDatabaseInterface>	PersistencyDataElement

Method**To create an interface**

- 1 In the Data Dictionary Navigator, right-click the `/Pool/Autosar/Interfaces/` subtree.
- 2 Choose **Create InterfaceGroup** from the context menu.
- 3 Enter a name for the interface group, e.g., **AdaptiveInterface**.
- 4 Right-click the interface group and choose **Create GroupInfo** from the context menu.
- 5 In the **Property Value List**, enter a package name, for example, **AdaptiveInterface**. This lets you export interfaces in the group to the specified package. Interfaces in AUTOSAR files located in the package can be imported to that location.
- 6 Right-click the interface group and choose **Create <Interface>** from the context menu.
- 7 Rename the <Interface> object created in step 6 as required.

Result

You created a DD <Interface> object. This object lets you specify communication subjects and can be connected to your software components via ports.

Related topics**Basics**

Accessing Fields.....	58
Accessing Persistent Memory.....	71
Calling and Implementing Methods.....	66
Receiving and Sending Events.....	62

How to Create Ports

Objective

To create ports according to Adaptive AUTOSAR in TargetLink, you must create a DD <Port> object.

Ports

Software components communicate via connection points, which are called ports. Ports can be divided into provide ports, require ports, and provide-require ports:

Mapping Between AUTOSAR Communication Kind, AUTOSAR Ports, and DD Port Objects			
AUTOSAR Communication Kind	Provide Port	Require Port	Provide-Require Port
Field	DD ServiceProvidePort object	DD ServiceRequirePort object	Not defined by AUTOSAR
Operation	DD ServiceProvidePort object	DD ServiceRequirePort object	Not defined by AUTOSAR
Event	DD ServiceProvidePort object	DD ServiceRequirePort object	Not defined by AUTOSAR
PersistencyDatabase	DD PersistencyProvidePort object	DD PersistencyRequirePort object	DD PersistencyProvideRequirePort object

Preconditions

The following preconditions must be fulfilled:

- You created an SWC, such as **MySWC**. Refer to [How to Create Software Components for Adaptive AUTOSAR](#) on page 46.
- You created an interface. Refer to [How To Create Interfaces \(Adaptive AUTOSAR\)](#) on page 48.

Method**To create a port**

- 1 From the `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports/` object's context menu, select **Create <Port>**.
- 2 Rename the DD <Port> object created in step 1 as required.
- 3 In the DD <Port> object's Property Value List, locate the <InterfaceRef> property to reference a suitable <Interface> object:

AUTOSAR Communication	<InterfaceRef> Property	<Interface> Object
Service based communication	DD ServiceInterfaceRef property	DD ServiceInterface object
Access to persistent data	DD PersistencyKeyValueDatabaseInterfaceRef property	DD PersistencyKeyValueDatabaseInterface object

Result

You created a port and referenced a suitable <Interface> for communication.

Next steps

If you are modeling the consumer side of service-based communication, you have to specify the service discovery settings. Refer to [How to Specify Service Discovery and Instance Selection](#) on page 51.

You have to create the subject of communication. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

Related topics**HowTos**

How To Create Interfaces (Adaptive AUTOSAR).....	48
How to Create Software Components for Adaptive AUTOSAR.....	46
How to Specify Service Discovery and Instance Selection.....	51

How to Specify Service Discovery and Instance Selection

Objective

To specify service discovery and instance selection for service-based communication via ara::com in TargetLink, you must use a DD ServiceDiscovery object.

Precondition

You created a DD ServiceRequirePort object. Refer to [How to Create Ports](#) on page 49.

Method**To specify service discovery and instance selection**

- 1 In the Data Dictionary Navigator, right-click the /Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports/<ServiceRequirePort> object.
- 2 Choose CreateServiceDiscoveries from the context menu.
- 3 Right-click the ServiceDiscoveries group and choose Create ServiceDiscovery.
- 4 Rename the DD ServiceDiscovery object as required.
- 5 In the Property Value list, set the SelectionCriteria property according to your use case.

Result

You specified the settings for service discovery and instance selection.

Related topics**Basics**

Basics on Service Discovery in TargetLink.....	16
--	----

HowTos

How to Create Ports.....	49
--------------------------	----

How to Create an Adaptive AUTOSAR Function Subsystem

Objective

To model a partial functionality of an adaptive application with TargetLink, you have the following options:

- To model the implementation of a method, you must use a [Method Behavior subsystem](#).
- To model a generic functionality, you must use an [Adaptive AUTOSAR Function subsystem](#).

This instruction shows you how to create an Adaptive AUTOSAR Function subsystem. To create a Method Behavior subsystem, refer to [How to Model a Method Implementation](#) on page 68.

Restrictions

The following restrictions hold for Adaptive AUTOSAR Function subsystems:

- They must not be placed in the subsystem hierarchy of Adaptive AUTOSAR Function subsystems.
- They must not be placed in the subsystem hierarchy of Method Behavior subsystems.
- They must not be placed in the subsystem hierarchy of Method Call subsystems.

Preconditions

- You created and specified a DD AdaptiveAutosarFunction object, such as MyAdaptiveAutosarFunction. Refer to [How to Specify an Adaptive AUTOSAR Function](#) on page 47.
- You opened a TargetLink model with a Data Dictionary for the Adaptive AUTOSAR use case. Refer to [How to Start Modeling from Scratch](#) on page 45.

Method

To create an Adaptive AUTOSAR Function subsystem

- 1 In the model, create an atomic subsystem and rename it as MyAdaptiveAUTOSARFunctionSubsystem.
- 2 Open MyAdaptiveAUTOSARFunctionSubsystem and add a Function block.
- 3 Open the block dialog of the Function block.
- 4 On the AUTOSAR page, make the following settings and close the dialog:

Property	Value
AUTOSAR mode	Adaptive
Role	Adaptive AUTOSAR Function
Function	MyAdaptiveAutosarFunction

Result

You created an Adaptive AUTOSAR Function subsystem.

Related topics**HowTos**

How to Get or Set the Value of a Field As a Service Consumer.....	58
How to Model a Method Call.....	66
How To Model Access to a Key-Value Pair of a Key-Value Storage.....	71
How to Receive Events (Adaptive AUTOSAR).....	62
How to Send Events (Adaptive AUTOSAR).....	64
How to Update the Value of a Field as a Service Provider.....	60

References

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

How to Create Communication Subjects for Adaptive AUTOSAR

Objective

To model communication for Adaptive Applications in TargetLink, you must create communication subjects.

Communication subjects

The following table shows the interface and its communication subjects and where they are specified in the Data Dictionary:

Interface	Communication Subject ¹⁾
DD ServiceInterface object	<ul style="list-style-type: none"> ▪ <ServiceInterface>/Methods/<Operation>²⁾ ▪ <ServiceInterface>/Fields/<ArField> ▪ <ServiceInterface>/Events/<ArEvent>
DD PersistencyKeyValueDatabaseInterface object	<ul style="list-style-type: none"> ▪ <PersistencyKeyValueDatabaseInterface>/PersistencyDataElements/<PersistencyDataElement>

¹⁾ The object path relative to /Pool/Autosar/Interfaces/.

²⁾ In TargetLink, a method is specified via a DD Operation object.

Preconditions

The following preconditions must be fulfilled:

- You created one of the following:
 - A DD ServiceInterface object.
 - A DD PersistencyKeyValueDatabaseInterface object.
 Refer to [How To Create Interfaces \(Adaptive AUTOSAR\)](#) on page 48.
- You specified one or more types. Refer to [Creating CPP Implementation Data Types From Scratch](#) on page 36.

Method**To create communication subjects for Adaptive AUTOSAR**

- 1 Select Create <CommunicationSubject> from one of the following objects' context menus:

- .../<ServiceInterface>/Methods/

Note

For methods, the communication subject consists of a method (implemented as a DD Operation object) and its operation arguments. You have to create the following child objects for the .../<ServiceInterface>/Methods/<Operation>/ object:

1. .../<ServiceInterface>/Methods/<Operation>/OperationArguments/
2. .../<ServiceInterface>/Methods/<Operation>/OperationArguments/<OperationArgument>

- .../<ServiceInterface>/Fields/
- .../<ServiceInterface>/Events/
- .../<PersistencyKeyValueDatabaseInterface>/PersistencyDataElements/

TargetLink creates a DD object for the communication subject.

- 2 Rename the DD object created in step 1 as required.
- 3 In the Property Value List, specify the following settings, depending on the kind of communication subject:

Communication subject	Property	Value
Operation	Kind ¹⁾	Select ARGIN for operation arguments that are input parameters of the method or ARGOUT for operation arguments that are output parameters of the method.
	Type ¹⁾	Select a user-defined data type. ²⁾
	Width ¹⁾	If you select an array user type, you have to specify a consistent width. ²⁾

Communication subject	Property	Value
ArField	Type	Select a user-defined data type. ²⁾
	Width	If you select an array user type, you have to specify a consistent width. ²⁾
	HasGetter	If you want to allow service consumers to access the field via Get() methods, select on.
	HasSetter	If you want to allow service consumers to access the field via Set() methods, select on.
ArEvent	Type	Select a user-defined data type. ²⁾
	Width	If you select an array user type, you have to specify a consistent width. ²⁾
PersistencyDataElement	Type	Select an AdaptivePlatformType .

¹⁾ At a DD

.../**<ServiceInterface>**/Methods/**<Operation>**/OperationArguments/**<OperationArgument>** object.

²⁾ If you select a struct user type, you have to create components of the communication subject that are consistent with the selected struct user type. You can use the Adjust to Typedef context menu command to synchronize this communication subject with the referenced DD Typedef object.

Result

You created a communication subject for Adaptive AUTOSAR.

Related topics

HowTos

How to Create Communication Subjects for Adaptive AUTOSAR.....	53
How To Create Interfaces (Adaptive AUTOSAR).....	48
How to Get or Set the Value of a Field As a Service Consumer.....	58
How to Model a Method Call.....	66
How to Model a Method Implementation.....	68
How to Update the Value of a Field as a Service Provider.....	60

References

[Adjust to Typedef](#) ( TargetLink Data Dictionary Manager Reference)

Modeling Communication as Defined by Adaptive AUTOSAR

Where to go from here

Information in this section

Communication as defined by ara::com.....	58
Communication as defined by ara::per.....	71

Communication as defined by ara::com

Where to go from here

Information in this section

Accessing Fields.....	58
Receiving and Sending Events.....	62
Calling and Implementing Methods.....	66

Accessing Fields

Where to go from here

Information in this section

How to Get or Set the Value of a Field As a Service Consumer.....	58
How to Update the Value of a Field as a Service Provider.....	60

How to Get or Set the Value of a Field As a Service Consumer

Objective

To get or set the value of a field as a service consumer, you must use port blocks that are specified as require ports and that are contained in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#).

If you want to model the access to fields as a service provider, refer to [How to Update the Value of a Field as a Service Provider](#) on page 60.

Preconditions

One of the following preconditions must be fulfilled:

- You modeled an [Adaptive AUTOSAR Function subsystem](#), refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the fields.
- You modeled a [Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the fields.

Additionally, the following preconditions must be fulfilled:

- You created a DD ArField object. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.
This field must belong to the interface that is referenced via the port that connects the software component and the interface.
- You specified the service discovery and instance selection. Refer to [How to Specify Service Discovery and Instance Selection](#) on page 51.
This service discovery and instance selection must belong to the ServiceRequirePort that connects the software component and the interface that contains the fields.

Method

To get or set the value of a field as a service consumer

- 1 In the model, open the [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#) in which you want to access the field.
- 2 Add the following port blocks, depending on your use case:

Use Case	Port Block
Getting the value of a field	InPort ¹⁾
Setting the value of a field	OutPort ²⁾

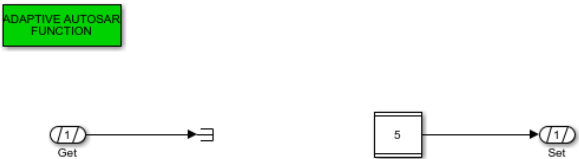
- ¹⁾ If the field is structured, use a Bus Inport instead.
²⁾ If the field is structured, use a Bus Outport instead.

- 3 Open the port block's block dialog and make the following settings on its AUTOSAR page:

Property	Value
AUTOSAR mode	Adaptive
Kind	Field
Port	A DD ServiceRequirePort object that references the DD ServiceInterface object that contains the DD ArField object you want to access.
Service discovery	The DD ServiceDiscovery object used to specify service discovery and instance selection. This object must be a child object of the already referenced DD ServiceRequirePort object.
Field	A DD ArField object contained in the DD ServiceInterface object that is referenced by the DD ServiceRequirePort object.

Result

You modeled the access of a field as a service consumer. Example:



Related topics**HowTos**

How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Model a Method Implementation.....	68
How to Update the Value of a Field as a Service Provider.....	60

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 Update the Value of a Field as a Service Provider

Objective

To update the value of a field as a service provider, you must use port blocks that are specified as provide ports and that are contained in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#).

If you want to model access to fields as a service consumer, refer to [How to Get or Set the Value of a Field As a Service Consumer](#) on page 58.

Preconditions

One of the following preconditions must be fulfilled:

- You modeled an [Adaptive AUTOSAR Function subsystem](#). Refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52:
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the fields.
- You modeled a [Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the fields.

Additionally, the following preconditions must be fulfilled:

- You created a DD ArField object. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.
This field must belong to the interface that is connected to the software component used for the field access.
- You created a DD ServiceProvidePort object. Refer to [How to Create Ports](#) on page 49.
This port must connect its software component and the interface that contains the fields.

Method

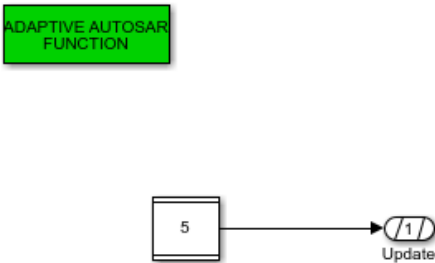
To update the value of a field as a service provider

- 1 In the model, open the [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#) in which you want to access the field.
- 2 Add an OutPort or Bus Outport block.
- 3 Open the port block's block dialog and make the following settings on its AUTOSAR page:

Property	Value
AUTOSAR mode	Adaptive
Kind	Field
Port	A DD ServiceProvidePort object that references the DD ServiceInterface object that contains the DD ArField object you want to access.
Field	A DD ArField object contained in the DD ServiceInterface object that is referenced by the DD ServiceProvidePort object.

Result

You modeled the access of a field as a service provider. Example:



Related topics

HowTos

How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Get or Set the Value of a Field As a Service Consumer.....	58
How to Model a Method Implementation.....	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)

Receiving and Sending Events

Where to go from here

Information in this section

How to Receive Events (Adaptive AUTOSAR)	62
How to Send Events (Adaptive AUTOSAR)	64

How to Receive Events (Adaptive AUTOSAR)

Objective

To receive events, you must use InPort or Bus InPort blocks that are specified as require ports and contained in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#).

If you want to model sending events, refer to [How to Send Events \(Adaptive AUTOSAR\)](#) on page 64.

ARA adapter code

Note

The generated [ARA adapter code](#) for events uses the API as defined in Adaptive AUTOSAR release 18-10.

If you generate code for a subsystem that contains InPort or Bus InPort blocks specified as described here, the resulting ARA adapter code includes calls to the following methods that are defined in the Adaptive AUTOSAR release 18-10:

- `GetSubscriptionState()`
- `Update()`
- `GetCachedSamples()`

Additionally, a single adapter code function for subscribing to events is generated for each CGU. This function contains calls to the `Subscribe()` method defined in Adaptive AUTOSAR release 18-10 with a `cacheSize` of 1 and `kLastN` as `EventCacheUpdatePolicy`.

Preconditions

One of the following preconditions must be fulfilled:

- You modeled an [Adaptive AUTOSAR Function subsystem](#), refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the events.
- You modeled a [Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.

The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the events.

Additionally, the following preconditions must be fulfilled:

- You created a DD ArEvent object. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

This event must belong to the interface that is referenced via the port that connects the software component and the interface.

- You specified the service discovery and instance selection. Refer to [How to Specify Service Discovery and Instance Selection](#) on page 51.

This service discovery and instance selection must belong to the ServiceRequirePort that connects the software component and the interface that contains the events.

Method

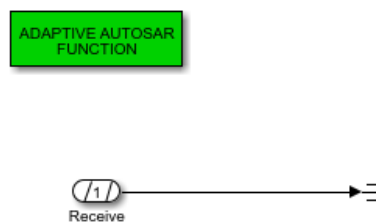
To receive events

- 1 In the model, open the [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#) in which you want to receive events.
- 2 Add an InPort or Bus InPort block and open its block dialog.
- 3 On the dialog's AUTOSAR page, make the following settings:

Property	Value
AUTOSAR mode	Adaptive
Kind	Event
Port	A DD ServiceRequirePort object that references the DD ServiceInterface object that contains the DD ArEvent object you want to access.
Service discovery	The DD ServiceDiscovery object used to specify service discovery and instance selection. This object must be a child object of the already referenced DD ServiceRequirePort object.
Event	A DD ArEvent object contained in the DD ServiceInterface object that is referenced by the DD ServiceRequirePort object.

Result

You modeled the reception of events. Example:



Related topics**HowTos**

How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Model a Method Implementation.....	68

References

Bus InPort Block (📖 TargetLink Model Element Reference)
InPort Block (📖 TargetLink Model Element Reference)

How to Send Events (Adaptive AUTOSAR)

Objective

To send events, you must use **OutPort** or **Bus Output** blocks that are specified as provide ports and contained in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#).

If you want to model the reception of events, refer to [How to Receive Events \(Adaptive AUTOSAR\)](#) on page 62.

ARA adapter code**Note**

The generated [ARA adapter code](#) for events uses the API as defined in Adaptive AUTOSAR release 18-10.

If you generate code for a subsystem that contains **OutPort** or **Bus Output** blocks specified as described here, the resulting ARA adapter code includes calls to the **Send()** and **Allocate()** methods as defined in the Adaptive AUTOSAR release 18-10.

Preconditions

One of the following preconditions must be fulfilled:

- You modeled an [Adaptive AUTOSAR Function subsystem](#), refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the events.
- You modeled a [Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component connected to the interface that contains the events.

Additionally, the following preconditions must be fulfilled:

- You created a DD ArEvent object. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

This event must belong to the interface that is connected to the software component used for the event access.

- You created a DD ServiceProvidePort object. Refer to [How to Create Ports](#) on page 49.

This port must connect its software component and the interface that contains the events.

Method

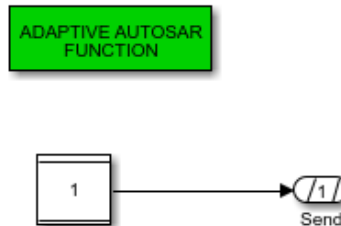
To send events

- 1 In the model, open the [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#) in which you want to send events.
- 2 Add an OutPort or Bus Outport block and open its block dialog.
- 3 On the dialog's AUTOSAR page, make the following settings:

Property	Value
AUTOSAR mode	Adaptive
Kind	Event
Port	A DD ServiceProvidePort object that references the DD ServiceInterface object that contains the DD ArEvent object you want to access.
Event	A DD ArEvent object contained in the DD ServiceInterface object that is referenced by the DD ServiceProvidePort object.

Result

You modeled sending events. Example:



Related topics**HowTos**

How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Model a Method Implementation.....	68

References

Bus Output Block (📖 TargetLink Model Element Reference)
OutPort Block (📖 TargetLink Model Element Reference)

Calling and Implementing Methods

Where to go from here**Information in this section**

How to Model a Method Call.....	66
How to Model a Method Implementation.....	68

How to Model a Method Call

Objective

To model a method call in TargetLink, you must create a [🔗 Method Call subsystem](#).

Restrictions

The following restrictions apply to Method Call subsystems

- They must not contain Adaptive AUTOSAR Function subsystems.
- They must not contain Method Behavior subsystems.
- They must not contain Method Call subsystems.

Preconditions

One of the following preconditions must be fulfilled:

- You created a [🔗 Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.
The subsystem's [🔗 Adaptive AUTOSAR Function](#) must belong to the SWC connected to the interface that contains the method.
- You created an [🔗 Adaptive AUTOSAR Function subsystem](#). Refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52.

The subsystem's [Adaptive AUTOSAR Function](#) must belong to the SWC connected to the interface that contains the method.

Additionally, the following preconditions must be fulfilled:

- You specified a method via a DD Operation object with operation arguments. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

This method must belong to the interface that the software component uses to call the method.

- You created and specified a DD ServiceDiscovery object. Refer to [How to Specify Service Discovery and Instance Selection](#) on page 51.

This service discovery and instance selection must belong to the port that connects the software component and the interface that contains the method.

Method

To model a method call

- 1 In the model, open the [Method Behavior subsystem](#) or [Adaptive AUTOSAR Function subsystem](#) in which you want to model a method call.
- 2 Create an atomic subsystem and rename it as **MyMethodCall**.
- 3 Open **MyMethodCall** and add a Function block.
- 4 Open the block dialog of the Function block.
- 5 On the AUTOSAR page, make the following settings and close the dialog:

Property	Value
AUTOSAR mode	Adaptive
Role	Method Call
Software component	The DD SoftwareComponent object connected to the interface used for the method call.
Require Port	The DD ServiceRequirePort object that specifies the port used for the method call.
Operation	The DD Operation object that specifies the method and its arguments.
Service discovery	The DD ServiceDiscovery object that specifies the service discovery and instance selection settings for the method call. This object must be a child object of the already referenced DD ServiceRequirePort object.

- 6 To the root level of the **MyMethodCall** subsystem, add the following model elements:

Model Element	Description
InPort ¹⁾	Add one block for each ARGIN operation argument of the method .
OutPort ²⁾	Add one block for each ARGOUT operation argument of the method.

¹⁾ If the operation argument is structured, use a Bus Inport block instead.

²⁾ If the operation argument is structured, use a Bus Outport block instead.

These model elements define the interface of the subsystem that TargetLink uses to generate the call of the method.

- 7 In the block dialog of each port block, make the following settings:

Property	InPort ¹⁾ Block	OutPort ²⁾ Block
AUTOSAR mode	Adaptive	
Kind	Operation	
Argument	One DD OperationArgument object that specifies an ARGIN argument of the method.	One DD OperationArgument object that specifies an ARGOUT argument of the method.

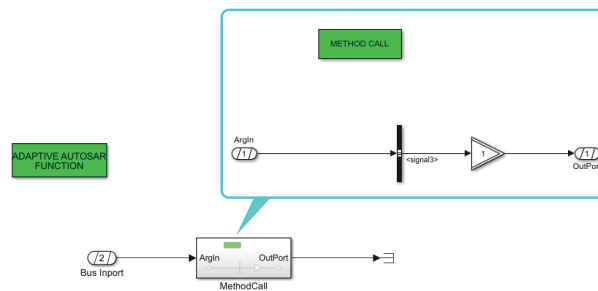
¹⁾ If the operation argument is structured, use a Bus Inport block instead.

²⁾ If the operation argument is structured, use a Bus Outport block instead.

- 8 Optionally, for simulation purposes, you can model the method within the subsystem.

Result

You modeled a method call.



Related topics

HowTos

How to Create an Adaptive AUTOSAR Function Subsystem.....	52
How to Create Communication Subjects for Adaptive AUTOSAR.....	53
How to Model a Method Implementation.....	68

References

Function Block (📖 TargetLink Model Element Reference)

How to Model a Method Implementation

Objective

To model a method implementation as defined by Adaptive AUTOSAR with TargetLink, you must create a [Method Behavior subsystem](#).

Restrictions

The following restrictions hold for Method Behavior subsystems

- They must not be placed in the subsystem hierarchy of Adaptive AUTOSAR Function subsystems.
- They must not be placed in the subsystem hierarchy of Method Behavior subsystems.
- They must not be placed in the subsystem hierarchy of Method Call subsystems.

Preconditions

The following preconditions must be fulfilled:

- You created a DD ServiceProvidePort object. Refer to [How to Create Ports](#) on page 49.

This port must connect its software component and the interface that contains the method.

- You created a DD AdaptiveAutosarFunction object. Refer to [How to Specify an Adaptive AUTOSAR Function](#) on page 47.

This [Adaptive AUTOSAR Function](#) must belong to the software component connected with the interface that contains the method.

- You specified a method via a DD Operation object with DD OperationArgument objects. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

This method must belong to the interface that the software component uses to provide the method.

Method**To model a method implementation**

- 1 In the model, create an atomic subsystem and rename it as **MyMethodImplementation**.
- 2 Open **MyMethodImplementation** and add a Function block.
- 3 Open the block dialog of the Function block.
- 4 On the AUTOSAR page, make the following settings and close the dialog:

Property	Value
AUTOSAR mode	Adaptive
Role	Method Behavior
Function	The DD AdaptiveAutosarFunction object that specifies the characteristics of the generated Adaptive AUTOSAR Function .
Provide Port	The DD ServiceProvidePort object that references the DD ServiceInterface object that contains the DD Operation object that specifies the method to implement.
Operation	The DD Operation object that specifies the method to implement.

- 5 To the root level of the **MyMethodImplementation** subsystem, add the following model elements:

Model Element	Description
InPort ¹⁾	Add one block for each operation argument of the ARGIN kind to be delivered from the service consumer to the service provider.
OutPort ²⁾	Add one block for each operation argument of the ARGOUT kind to be delivered from the service provider to the service consumer.

¹⁾ If the operation argument is structured, use a **Bus Inport** block instead.

²⁾ If the operation argument is structured, use a **Bus Outport** block instead.

These model elements define the interface of the subsystem that **TargetLink** uses to generate the signature of the method.

- 6 In the block dialog of each port block, make the following settings:

Property	InPort ¹⁾ Block	OutPort ²⁾ Block
AUTOSAR mode	Adaptive	
Kind	Operation	
Argument	The DD OperationArgument object that specifies the operation argument to be passed from the service consumer to the service provider.	The DD OperationArgument object that specifies the operation argument to be passed from the service provider to the service consumer.

¹⁾ If the operation argument is structured, use a **Bus Inport** block instead.

²⁾ If the operation argument is structured, use a **Bus Outport** block instead.

- 7 Model the behavior of the method between the port blocks.

Result

You modeled the behavior of a method.



Related topics

References

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

Communication as defined by ara::per

Accessing Persistent Memory

Where to go from here

Information in this section

How To Model Access to a Key-Value Pair of a Key-Value Storage.....	71
How To Specify Initial Values for Persistency Data Elements.....	73

How To Model Access to a Key-Value Pair of a Key-Value Storage

Objective

To model access to a key-value pair of a key-value storage, you must use a Data Store Memory block in combination with one or both of the following:

- A Data Store Read block in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#)
- A Data Store Write block in an [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#)

Restriction

This instruction shows you how to model for the generation of production code.

Preconditions

One of the following preconditions must be fulfilled:

- You modeled an [Adaptive AUTOSAR Function subsystem](#), refer to [How to Create an Adaptive AUTOSAR Function Subsystem](#) on page 52.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the same software component that is connected to the interface.
- You modeled a [Method Behavior subsystem](#). Refer to [How to Model a Method Implementation](#) on page 68.
The subsystem's [Adaptive AUTOSAR Function](#) must belong to the software component that is connected to the interface.

Additionally, the following preconditions must be fulfilled:

- You created a DD PersistencyKeyValueDatabaseInterface object. Refer to [How To Create Interfaces \(Adaptive AUTOSAR\)](#) on page 48.
- You created one or more DD PersistencyDataElement objects. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

- You created one of the following depending on your use case:

Use Case	DD Object
Reading data from a key-value storage	PersistenceRequirePort
Writing data to a key-value storage	PersistenceProvidePort
Reading and/or writing data	PersistenceProvideRequirePort

Refer to [How to Create Ports](#) on page 49.

The DD <Port> object must connect the DD SoftwareComponent object and the DD PersistenceKeyValueDatabaseInterface object.

Method

To model access to a key-value pair of a key-value storage

- 1 To your model, add a Data Store Memory block. Place the block either in the subsystem used for the access or above in the subsystem hierarchy.
- 2 Open the Data Store Memory block's dialog and make the following settings on its AUTOSAR page:

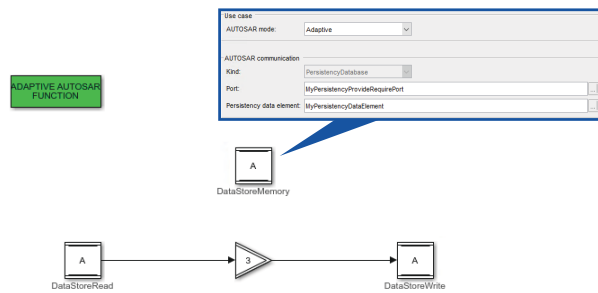
Property	Value
AUTOSAR mode	Adaptive
Port	<ul style="list-style-type: none"> ▪ Reading data: PersistenceRequirePort ▪ Writing data: PersistenceProvidePort ▪ Reading and/or writing data: PersistenceProvideRequirePort
Persistence data element	A DD PersistenceDataElement object contained in the DD PersistenceKeyValueDatabaseInterface object that is referenced by the DD <Port> object.

- 3 Open the [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#) in which you want to access the key-value pair of a key-value storage.
- 4 Add the following blocks depending on your use case:

Use Case	Blocks
Reading data from the key-value storage	<ul style="list-style-type: none"> ▪ A Data Store Read block
Writing data to a key-value storage	<ul style="list-style-type: none"> ▪ A Data Store Write block
Reading and writing data	<ul style="list-style-type: none"> ▪ A Data Store Read block ▪ A Data Store Write block

Result

You modeled the access to a key-value pair of a key-value storage. Example:

**Related topics****Basics**

Basics on Modeling AUTOSAR Communication via Data Stores ([TargetLink Classic AUTOSAR Modeling Guide](#))

HowTos

How to Create an Adaptive AUTOSAR Function.....	52
How to Create Communication Subjects for Adaptive AUTOSAR.....	53
How To Create Interfaces (Adaptive AUTOSAR).....	48
How to Model a Method Implementation.....	68
How To Specify Initial Values for Persistency Data Elements.....	73

References

Data Store Memory Block ([TargetLink Model Element Reference](#))
 Data Store Read Block ([TargetLink Model Element Reference](#))
 Data Store Write Block ([TargetLink Model Element Reference](#))

How To Specify Initial Values for Persistency Data Elements

Objective

To specify initial values for persistency data elements, you must create DD `PersistenceDataRequiredComSpec` objects or DD `PersistenceDataProvidedComSpec` objects.

Preconditions

- You created a DD `PersistenceKeyValueDatabaseInterface` object. Refer to [How To Create Interfaces \(Adaptive AUTOSAR\)](#) on page 48.
- You created one or more DD `PersistenceDataElement` objects. Refer to [How to Create Communication Subjects for Adaptive AUTOSAR](#) on page 53.

- You created one of the following:
 - A DD PersistencyRequirePort object
 - A DD PersistencyProvidePort object
 - A DD PersistencyProvideRequirePort object

Refer to [How to Create Ports](#) on page 49.

Workflow

The workflow to specify initial values for persistency data elements consists of the following parts:

- Specifying initial values via DD Variable objects (refer to [Part 1](#) on page 74).
- Referencing initial values at persistency data elements (refer to [Part 2](#) on page 74).

Part 1

To specify initial values via DD Variable objects

- 1 In the Data Dictionary Navigator, create a variable group for the initialization constants and name it **Constants**.
- 2 From the context menu, choose **Create GroupInfo**.
- 3 In the Property Value List, enter a package name, for example, **Constants**. This lets you export variables in the group to the specified package. Constants in AUTOSAR files located in the package can be imported to that location.
- 4 From the context menu, choose **Create Variable**.
- 5 Enter a name for the variable, for example, **my_InitializationConstant**.
- 6 In the Property Value List, specify the following settings:

Property	Value
Type	Select the AdaptivePlatformType that is consistent with the persistency data element.
Value	Specify the initialization constant value, for example, 0.
VariableClass	AUTOSAR/CONST_VALUE

Interim result

You created a DD Variable object that specifies the initial values for a persistency data element.

Part 2

To reference initial values for persistency data elements

- 1 Right-click the DD <Port> object and select one of the following depending on your requirements and the DD <Port> object:
 - **Create PersistencyDataRequiredComSpec**
 - **Create PersistencyDataProvidedComSpec**
- 2 In the Property Value List, use the **PersistencyDataElementRef** property to reference a DD **PersistencyDataElement** object.

This persistency data element must belong to the interface that is connected to the port for which you created the DD <ComSpec> object.

- 3 In the **Property Value List** of the persistency data element, select the variable created before via the **InitValueRef Browse** button.

Result

You specified initial values for a persistency data element via a **DD Variable** object.

Related topics**HowTos**

How to Create Communication Subjects for Adaptive AUTOSAR.....	53
How To Create Interfaces (Adaptive AUTOSAR).....	48
How to Create Ports.....	49
How To Model Access to a Key-Value Pair of a Key-Value Storage.....	71

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.....	78
A.....	79
B.....	82
C.....	83
D.....	86
E.....	88
F.....	89
G.....	90
I.....	90
L.....	92
M.....	93
N.....	95
O.....	96
P.....	97
R.....	98
S.....	100
T.....	102
U.....	104
V.....	104
W.....	105

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](#).

0

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](#).

C

Common Program Data folder 6
CommonProgramDataFolder 6
creating interfaces 48

D

Documents folder 6
DocumentsFolder 6

I

interfaces
creating 48

L

Local Program Data folder 6
LocalProgramDataFolder 6

