TargetLink AUTOSAR Guidelines

Guidelines for the Implementation of AUTOSAR Software Components with TargetLink

The Guide applies to the following TargetLink versions:
- TargetLink 3.5
- TargetLink 3.4
- TargetLink 3.3
- TargetLink 3.2
- TargetLink 3.1

Version 2.0   October 2013
## Document Change Record

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Originator</th>
<th>Section</th>
<th>Reason for Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-03-05</td>
<td>0.9</td>
<td>Ulrich Eisemann</td>
<td>all</td>
<td>First draft created</td>
</tr>
<tr>
<td>2011-03-05</td>
<td>1.0</td>
<td>Ulrich Eisemann</td>
<td>all</td>
<td>First official version finalized</td>
</tr>
<tr>
<td>2011-03-23</td>
<td>1.1</td>
<td>Ulrich Eisemann</td>
<td>all</td>
<td>Minor corrections, extension regarding the initialization of variables, guidelines regarding SWC container exchange added. Spelling corrections. Extension of guideline</td>
</tr>
<tr>
<td>2013-10-05</td>
<td>2.0</td>
<td>Ulrich Eisemann</td>
<td>Individual Guidelines</td>
<td>Introduction in chapter 1 slightly extended and modified&lt;br&gt;Names for individual guidelines have been introduced for quick reference that remain stable across different versions of that document&lt;br&gt;Guidelines “Modeling Servers for Client-Server Operation” [TLAR_MODELING_SERVER] has been added&lt;br&gt;Extensions for TargetLink 3.5, TargetLink 3.4, TargetLink 3.3, incremental code generation&lt;br&gt;Guideline “No Multiple Instantiation of Software Components” has been adjusted&lt;br&gt;Guideline “Dealing with AUTOSAR Compositions” has been rephrased&lt;br&gt;Guideline “AUTOSAR SWCs and Incremental Code Generation” has been adapted&lt;br&gt;Guideline “Required Predefined Data Dictionary Specifications” has been adjusted&lt;br&gt;Guideline “Atomic Software Components and TargetLink Subsystems” has been adjusted&lt;br&gt;Guideline “Modeling Clients for Synchronous Client-Server Communication” has been adjusted&lt;br&gt;Guidelines “Structuring AUTOSAR-Related Data in the Data Dictionary” has been adjusted&lt;br&gt;General rework at lots of different guidelines with minor modifications</td>
</tr>
</tbody>
</table>
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

There are different ways to contact dSPACE Support:

- Send an e-mail or phone:
  - TargetLink Technical Support:
    support.tl@dspace.de
    +49 5251 1638-700
- Use the dSPACE Support Wizard:
  - On your dSPACE DVD at \Diag\Tools\dSPACESupportWizard.exe
  - Via Start – Programs – dSPACE Tools (after installation of the dSPACE software)
  - At http://www.dspace.com/goto?supportwizard

You can always find the latest version of the dSPACE Support Wizard here.

dSPACE recommends that you use the dSPACE Support Wizard to contact dSPACE Support.

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/goto?support for software updates and patches.

Important Notice

This document contains proprietary information that is protected by copyright. All rights are reserved. Neither the documentation nor software may 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.

© Copyright 1998 - 2011 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

AutomationDesk, CalDesk, ConfigurationDesk, ControlDesk, SystemDesk and TargetLink 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.

This publication and the contents hereof are subject to change without notice. dSPACE GmbH makes no warranty of any kind with regard to this publication, including but not limited to the implied warranties of merchantability or fitness for a particular purpose. dSPACE GmbH shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this publication.
Contents

1 Introduction ................................................................................................................................. 6
   1.1 Content of this Document ....................................................................................................... 7
   1.2 Version Dependency ............................................................................................................... 8
   1.3 Related Documents and useful Links .................................................................................... 8

2 Modeling on the Block Diagram Level ......................................................................................... 9
   2.1 Global AUTOSAR Code Generation Setting [TLAR_CG_SETTING] ......................................... 9
   2.2 Common Modeling Style for SWCs and Their Internal Behaviors [TLAR_MODELING_IB] ........ 9
   2.3 Atomic Software Components and TargetLink Subsystems [TLAR_SWCS_TLSUBSYSTEMS] ... 10
   2.4 Code for SWCs and Runnables vs. Stub Code [TLAR_SWCS_STUBCODE] .............................. 11
   2.5 Modeling Runnables [TLAR_MODELING_RUNNABLES] ............................................................ 12
   2.6 Using TargetLink Imports/Outports at the Borders of Runnables [TLAR_RUNNABLE_PORTS] ... 15
   2.7 Activation of Runnables and RTE Events [TLAR_RUNNABLE_EXECUTION_RTE_EVENT] ....... 18
   2.8 Using SWC Sender/Receiver Blocks [TLAR_SENDERRECEIVER_BLOCKS] ......................... 19
   2.9 Modeling Clients for Synchronous Client-Server Communication [TLAR_MODELING_CLIENT] ... 20
   2.10 Modeling Servers for Client-Server Operations [TLAR_MODELING_SERVER] ..................... 24
   2.11 Initialization of AUTOSAR-Related Signals [TLAR_INIT_AR SIGNALS] ............................ 28
   2.12 Initialization of non-AUTOSAR-Related Signals inside an SWC [TLAR_INIT_NONARSIGNALS] .. 32

3 Modeling in the Data Dictionary .................................................................................................. 36
   3.1 Required Predefined Data Dictionary Specifications [TLAR_PREDEFINED_DD] ..................... 36
   3.2 Names for AUTOSAR-Related Data Dictionary Objects [TLAR_DDARNAMES] ..................... 38
   3.3 Structuring AUTOSAR-Related Data in the Data Dictionary [TLAR_DDSTRUCTURE] ............. 40
   3.4 Assigning AUTOSAR Packages to Data Dictionary Objects [TLAR_PACKAGE_ASSIGNMENTDD] ... 41
   3.5 Performing ARXML Imports and Outports with Package Support [TLAR_PACKAGE_SUPPORTDD] ... 42
   3.6 Modeling Types in AUTOSAR Projects [TLAR_TYPESDD] ....................................................... 44
   3.7 Specifying AUTOSAR Scalings (CompuMethods) in the Data Dictionary [TLAR_SCALINGDD] ... 46
   3.8 Specifying AUTOSAR Units in the Data Dictionary [TLAR_UNITDD] ...................................... 47
   3.9 Specifying Sender/Receiver or Mode Switch Interfaces in the Data Dictionary [TLAR_SRINTERFACEDD] ... 48
   3.10 Specifying Client/Server Interfaces in the Data Dictionary [TLAR_CSINTERFACEDD] ............. 49
   3.11 Specifying Interrunnable Variables in the Data Dictionary [TLAR_IRVDD] ............................ 50

4 Limitations ................................................................................................................................... 53
   4.1 Restricted Subset of the Software Component Template [TLAR_ARSUBSET] ........................... 53
   4.2 Dealing with AUTOSAR Compositions [TLAR_COMPOSITIONS] ...................................... 53
4.3 Multiple Instantiation of Software Components [TLAR_MULTIPLEINSTANCES]..................................55
4.4 Restricted Simulation Support for Sender-Receiver Communication with Event Semantics [TLAR_EVENTSEMANTICSSIMULATION]..................................................................................56
4.5 AUTOSAR SWCs and Incremental Code Generation using Model Referencing/Incremental Subsystems [TLAR_INCREMENTALCG]................................................................................................57

5 Guidelines for SWC Container Exchange between SystemDesk and TargetLink..............................58

5.1 General Guidelines for SystemDesk and TargetLink ........................................................................58
5.1.1 Specification of Workflow Rules [TLAR_CONTAINER_WORKFLOW]..............................................58
5.1.2 Use of Container Set Files for SWC Container Exchange [TLAR_CONTAINER_CONTAINERSETS]..................................................................................................................58

5.2 SystemDesk Guidelines for SWC Container Exchange .....................................................................61
5.2.1 Partitioning Software Components into Packages [TLAR_CONTAINER_SD_SWCPACKAGES].................................................................61
5.2.2 Package Assignment for all Objects in the SystemDesk Project [TLAR_CONTAINER_SD_PACKAGES]..................................................................................................................63
5.2.3 Separation of Shared and Component-specific Data [TLAR_CONTAINER_SD_DATASEPARATION]..................................................................................................................63
5.2.4 File Names for AUTOSAR Packages [TLAR_CONTAINER_SD_FILENAMES].............................65
5.2.5 Specifying Workflow Rules for SystemDesk [TLAR_CONTAINER_SD_WORKFLOW].....................65

5.3 TargetLink Guidelines for the SWC Container Exchange ..................................................................67
5.3.1 Software Components and TargetLink Subsystems [TLAR_CONTAINER_TL_SUBSYSTEMS]..........................67
5.3.2 Specifying Workflow Rules for TargetLink [TLAR_CONTAINER_TL_WORKFLOW].....................69
5.3.3 Specifying the Name of the Container Catalog File [TLAR_CONTAINER_TL_CATALOGNAME]........70

6 List of References ..................................................................................................................................72
1 Introduction

This document contains some guidelines for TargetLink® users who want to model, implement and test AUTOSAR software components (SWCs) with TargetLink. TargetLink is a production code generator for implementing control functions modeled in Simulink®/Stateflow®/TargetLink in the form of very efficient C code. TargetLink particularly supports the AUTOSAR use case, in which models are implemented in the form of AUTOSAR software components, "filling in" individual components of the software architecture designed in tools like dSPACE SystemDesk. Figure 1 shows a sketch of the AUTOSAR software architecture and the parts that are addressed by TargetLink. Please note that TargetLink deals with the application layer of the AUTOSAR software architecture exclusively. It does not constitute an RTE generator. TargetLink generates a partial “stub” RTE for SIL/PIL simulations though and is able to produce so-called RTE application header files for the AUTOSAR compatibility mode since TargetLink 3.4.

Figure 1: The parts of the AUTOSAR software architecture addressed by TargetLink: TargetLink is used exclusively to model the behavior of AUTOSAR SWCs, provide an implementation and test it in MIL/SIL/PIL simulations.

According to the AUTOSAR methodology, architecture or AUTOSAR authoring tools like dSPACE SystemDesk and the behavior modeling tool TargetLink work together by exchanging standardized AUTOSAR ARXML files, see Figure 2. The key element in this is the Software Component Template, which describes the content of AUTOSAR software components. The SystemDesk user has the role of a software architect who decomposes the entire software into a number of software components and makes sure that the software components communicate with one another through compatible interfaces. The software architect typically exchanges ARXML files with multiple function/software developers who use TargetLink to provide the implementation of individual software components, see Figure 2. With TargetLink, the software components are not only implemented, but can also be tested in the Simulink/TargetLink environment in the usual manner. Typically, the development of software components is an iterative process in which ARXML files are exchanged multiple times between tools like SystemDesk and TargetLink to accommodate changes.
Figure 2: Working with TargetLink in an AUTOSAR Tool Chain. dSPACE SystemDesk is taken as an example of a typical AUTOSAR architecture tool used in an AUTOSAR tool chain in combination with TargetLink. TargetLink and the AUTOSAR architecture tool exchange SWC description files based on the standardized AUTOSAR XML format.

Since TargetLink 3.2 and SystemDesk 3.0, both tools do not just support the exchange of AUTOSAR ARXML files but provide a more convenient, efficient and transparent way to develop AUTOSAR software by exchanging so-called SWC containers, see Figure 3. The guidelines in this document are valid for both approaches. Guidelines for the exchange of SWC containers between TargetLink and SystemDesk are specifically described in chapter 0 of this document. Chapter 0 therefore mostly contains guidelines for SystemDesk to ensure proper SWC container exchange with TargetLink.

1.1 Content of this Document

The rules in this document cover the modeling of AUTOSAR software components and subsequently generating AUTOSAR compliant code as well as an AUTOSAR software component description. It is subdivided into the following chapters:

- Chapter 2 contains guidelines regarding the modeling of AUTOSAR SWCs on the block diagram level, i.e. how to use the blocks from the TargetLink AUTOSAR library.
- Chapter 3 contains guidelines regarding the specification of AUTOSAR-related data in the Data Dictionary. This chapter is particularly relevant if a lot of AUTOSAR-related data is created manually in the Data Dictionary rather than being imported from ARXML files.
- Chapter 4 contains some limitations when it comes to modeling AUTOSAR SWCs with TargetLink.
- Chapter 5 contains mostly guidelines for SystemDesk to ensure proper SWC container exchange with TargetLink. This chapter is only relevant if AUTOSAR-compliant development is carried out using SWC container exchange between SystemDesk and TargetLink.
Figure 3: SWC container exchange between SystemDesk and TargetLink is a more convenient way of developing AUTOSAR-compliant software. Instead of exchanging individual AUTOSAR XML files, users simply have to import/export SWC containers. Other files like .c and .h implementation files are automatically taken care of.

1.2 Version Dependency

Note that the rules in this document apply to TargetLink versions as recent as TargetLink 3.1 although they might be useful for older versions, too. To use the TargetLink AUTOSAR support, the TargetLink AUTOSAR Module (TAS) is required.

1.3 Related Documents and useful Links

Besides the TargetLink AUTOSAR Guidelines contained in this document, the following documents also provide valuable information for implementing SWCs with TargetLink:

- The regular TargetLink AUTOSAR Modeling Guide, which is part of every TargetLink installation.

- An Excel sheet with a detailed list of AUTOSAR features and the extent to which these features are supported by TargetLink. The Excel table is available upon request from TargetLink Support via support.tl@dspace.de.

- Additional application notes and other material like the TargetLink AUTOSAR Utilities or TargetLink AUTOSAR webinars are available for download from the TargetLink Product Support Center via the following URL: www.dspace.com/tlpsc.
2 Modeling on the Block Diagram Level

This chapter contains guidelines that are relevant to the block diagram level when developing AUTOSAR SWCs with TargetLink.

2.1 Global AUTOSAR Code Generation Setting

[TLAR_CG_SETTING]

The *code generation mode* option in the TargetLink Main Dialog has to be set to *AUTOSAR* for the design and the implementation of AUTOSAR SWCs with TargetLink.

**Purpose**

Modeling SWCs with TargetLink

**Example**

Figure 4 shows the required *code generation mode* setting which is available in the TargetLink Main Dialog.

![Figure 4: In order to develop AUTOSAR software components, the global code generation mode option must be set to AUTOSAR.](image)

2.2 Common Modeling Style for SWCs and Their Internal Behaviors

[TLAR_MODELING_IB]

A modeling style often used for the development of SWCs makes use of Simulink function calls to model RTE events, which might be emitted from a separate Stateflow chart serving as a scheduler. Runnables are then modeled as function-call-triggered subsystems activated by the Stateflow chart.
Note: This modeling style also supports the introduction of feedback loops between different runnables and allows a precise definition of the execution order of different runnables. The modeling style is very flexible because it helps to avoid algebraic loops in the Simulink model.

Purpose
Modeling SWCs and their internal behavior with TargetLink

Example
The TargetLink demo model ar_poscontrol, whose TargetLink subsystem is sketched in Figure 5, demonstrates a potential modeling style in accordance with the guidelines described in this section.

![Diagram](image)

Figure 5: A potential modeling style describing the interior of an SWC on the top hierarchy level of a TargetLink subsystem (see the TargetLink ar_poscontrol demo model). Runnables are activated by Simulink function calls (thereby imitating the RTE events). Please note that only the content of runnable subsystems ends up in the production code for an SWC. Other parts are mere simulation code (stub code). SWC sender/receiver blocks can be used to represent the actual AUTOSAR ports of the SWC in the model. However, their presence is entirely optional.

2.3 Atomic Software Components and TargetLink Subsystems
[TLAR_SWCS_TLSUBSYSTEMS]

The following guidelines apply to SWCs and the mapping of runnables and SWCs onto TargetLink subsystems:

- An atomic software component does not need to have a direct representation in a TargetLink model, since it is primarily the runnables of the SWC that need to be modeled. However, one might use an ordinary virtual subsystem as a representation of an SWC in a TargetLink model.
- With TargetLink versions prior to TargetLink 3.4, all runnables of the same SWC should be modeled in the same TargetLink subsystem, i.e. they should not be modeled in different TargetLink subsystems.
- For TargetLink versions prior to TargetLink 3.4, it is recommended to use just one TargetLink subsystem for the modeling of SWCs inside one TargetLink model. Hence, if mul-
Multiple SWCs are designed inside one TargetLink model, then they should be placed inside the same TargetLink subsystem.

**Note:** For TargetLink versions prior to TargetLink 3.4, it is therefore not recommended to split one software component into different TargetLink subsystems, see Figure 6.

**Note:** Since TargetLink 3.4, incremental code generation for AUTOSAR SWCs is supported by TargetLink. Consequently, different runnables of the same SWC can be mapped onto different TargetLink subsystems, incrementally generated subsystems as well as referenced models.

**Purpose**
Modeling SWCs with TargetLink while maintaining the ability to do SIL/PIL simulations.

**Example**
Figure 6 shows the splitting of different runnables of the same SWC onto different TargetLink subsystems which should not be done using TargetLink versions prior to TargetLink 3.4.

**Figure 6:** For TargetLink versions prior to TargetLink 3.4, it is impermissible to partition runnables of the same software component (here SWC1) into different TargetLink subsystems (here TargetLink_Subsystem_1 and TargetLink_Subsystem_2). The content of one SWC must be contained in one TargetLink subsystem exclusively for TargetLink versions prior to TargetLink 3.4. When using TargetLink 3.4 or newer versions however, this modeling style is permitted.

### 2.4 Code for SWCs and Runnables vs. Stub Code

[TLAR_SWCS_STUBCODE]

All parts of a SWC that are supposed to become code fragments of the SWC must be contained within runnable subsystems. All model parts outside of runnable subsystems will not become part of the code for a SWC but will result in simulation/stub code for SIL/PIL simulations in Simulink only.
Note
In addition to an SWC’s runnables, model fragments which do not belong to any runnable can also be part of a TargetLink subsystem. However, those model fragments will serve for simulation purposes in Simulink only, since only the content of runnables is part of the code of an SWC and can therefore become part of an AUTOSAR application.

Purpose
Modeling SWCs with TargetLink

Example
Figure 7 shows an example of a model of SWC that is made up of two runnables. Functional blocks outside the runnables are not part of the SWC but serve for simulations in Simulink in MIL/SIL/PIL mode only.

![Figure 7: The root level of a TargetLink subsystem: It contains not only two runnables (Controller_Runnable and Position_Linearization_Runnable) belonging to one or two SWCs but also an additional Gain Block (named Gain) which is outside of any runnable and hence not part of a SWC. Consequently, it serves for simulation purposes in Simulink only (in MIL, SIL and PIL mode). The code for the Gain Block is not part of any SWC.](image)

2.5 Modeling Runnables
[TLAR_MODELING_RUNNABLES]

The following guidelines must be applied to the modeling of runnables:
- A runnable has to be represented by a Simulink subsystem with a TargetLink Function/Runnable block inside it. The subsystem should be atomic.
- The option Implement subsystem as runnable in the Function/Runnable block must be selected.
- The subsystem which contains the runnable may contain an arbitrary number of other subsystems which can also have a TargetLink function block inside them.
- Runnables must not be nested since this is not supported by the AUTOSAR standard. However, they can contain an arbitrary number of conventional subfunctions.
- Whether the subsystem of the runnable is an atomic subsystem, a function-call-triggered subsystem or an enabled or edge-triggered subsystem, is up to the user. Modeling styles using atomic and/or function-call triggered subsystems appear to be the most common
approaches. However, the activation of the runnable subsystem has no bearing on the
code for an SWC but is for simulation purposes in MIL/SIL/PIL mode in Simulink only.

- It is generally not recommended (although possible) to specify the root level of a Tar-
getLink subsystem as a runnable. Instead, subsystems below that level should be used.

**Purpose**
Modeling AUTOSAR SWCs with TargetLink

**Remark**
- Upon code generation, a C function is generated for each runnable.
- All model fragments that reside **outside of runnable subsystems do not become part**
  of any SWC, see Figure 7. For them, code is generated nonetheless which constitutes
  simulation/stub code for the Simulink simulation only.

**Example**
Figure 8 shows the basic structure of a simple runnable. In order for the subsystem to be-
come an AUTOSAR runnable, the **Implement subsystem as runnable** option must be select-
ed in the Function/Runnable block as demonstrated in Figure 9. Figure 10 shows the imper-
missible nesting of runnable subsystems.

![Figure 8: The subsystem becomes a runnable due to the presence of the Runnable block (green).](image)
Figure 9: The *Implement subsystem as runnable* checkbox to implement the subsystem as an AUTOSAR runnable must be selected.

Figure 10: Impermissible nesting of runnables. A subsystem with a runnable block contains another subsystem with a second runnable. This modeling style is not AUTOSAR-compliant. However, a runnable can contain an arbitrary number of subsystems with normal function specifications.
2.6 Using TargetLink Inports/Outports at the Borders of Runnables

[TLAR_RUNNABLE_PORTS]

For the signals entering or leaving a runnable (except for its potential trigger/enable port), the following guidelines must be observed:

- All data signals entering a runnable subsystem must be connected to a TargetLink inport before they are connected to blocks other than a Simulink Inport block. TargetLink inports are used either directly at the root level of the runnable subsystem or in subsystems below it. In any case, signals must have passed a TargetLink inport before they enter another block which is not a Simulink inport block.

- All data signals leaving a runnable subsystem must be connected to a TargetLink outport after they are connected to blocks other than a Simulink outport block. TargetLink outports are used either directly at the root level of the runnable subsystem or in subsystems below it. In any case, signals must have passed a TargetLink outport block before they leave the runnable subsystem.

- All signals entering or leaving a runnable constitute signals that are read/written via AUTOSAR RTE macros, e.g. data elements, operation arguments, interrunnable variables, PIMs etc. AUTOSAR communication mechanisms must therefore be specified in TargetLink inport/outport blocks for those signals by selecting the *Use AUTOSAR communication* option in order to be fully AUTOSAR-compliant.

- In some situations, it might be preferable to deviate from the AUTOSAR standard and occasionally read/write e.g. from/to global variables inside a runnable. For those purposes, the *Use AUTOSAR communication* option must be cleared as well as the global *StrictRunnableInterfaceChecks* code generation option.

- Function Call events must be fed into a runnable subsystem only via its Trigger port, not via a normal Simulink inport, since this is not in accordance with the AUTOSAR standard. Similarly, function call events must not leave a runnable subsystem. The runnable itself, however, can naturally be a function-call-triggered subsystem. Blocks in the runnable can also emit a function call event as long as it does not cross the border of the runnable.

- If AUTOSAR-related signals (e.g. data elements in a Sender/Receiver interface) are structures/structs (in the sense of the C language), then TargetLink BusPorts have to be used. For signals that are not structures, ordinary TargetLink ports are used.

**Purpose**
Modeling SWCs with TargetLink

**Remark**
In some situations, it might be preferable to deviate from the AUTOSAR standard and occasionally read/write e.g. from/to global variables inside a runnable (e.g. for complex device drivers). TargetLink does permit this kind of modeling if the *StrictRunnableInterfaceChecks* option is cleared.

**Example**
Figure 11 shows the proper use of TargetLink Inports/Outports in a runnable either directly on the level of the runnable itself or in a subsystem inside it. Figure 12 sketches impermissible modeling configurations for a runnable because the interface signals of the runnable are not properly defined in TargetLink Ports. Figure 13 demonstrates the specification of AUTOSAR communication mechanisms in a TargetLink Inport block. Figure 14 shows the *StrictRunnableInterfaceChecks* code generation option that allows some of the signals entering/leaving a runnable subsystem to be non-AUTOSAR signals. Figure 15 sketches a situation in which a function call event crosses the border of a runnable, which is not permitted.
**Figure 11:** Proper use of TargetLink Inports/Outports in a runnable. All signals entering or leaving a runnable have to be connected via TargetLink Inports/Outports before they enter or leave the runnable. Simulink Inports/Outports are allowed between the borders of the runnable subsystem and the TargetLink ports.

**Figure 12:** Improper use of the signals entering/leaving a runnable subsystem. The In1 input signal constitutes the input signal of a Look-up Table block before it has been specified properly in a TargetLink Inport block. Similarly, the In1 and In2 input signals are muxed together before they are properly specified in a TargetLink Inport block. In addition, the Out1 output signal produced by the Look-up Table block is not properly specified with a TargetLink Outport block.
Figure 13: AUTOSAR communication mechanisms need to be specified for all signals that enter or leave the runnable’s subsystem. For sender/receiver, client/server or interrunnable communication, the “Use AUTOSAR communication” checkbox needs to be selected. For communication via per instance memories, the “Use AUTOSAR communication” checkbox must be cleared and a per instance memory variable must be selected on the Output tab of the dialog.

Figure 14: The *StrictRunnableInterfaceChecks* code generation option can be cleared to allow access to non-AUTOSAR variables inside the runnable or e.g. for Per Instance Memories. The option is set by default.
Figure 15: It is not permitted to feed a function call event (emitted from the Function-Call Generator block) into a runnable, since this is not in accordance with the AUTOSAR standard. In a similar manner, function call events must not leave a runnable. Inside the runnable however, function calls can be used at will.

2.7 Activation of Runnables and RTE Events

[TLAR_RUNNABLE_EXECUTION_RTE_EVENT]

For proper simulation of the runnables in Simulink (in MIL/SIL/PIL mode), the activation of the runnables (in the proper order) has to be modeled according to Simulink semantics:

- In those cases where runnables constitute function-call-triggered subsystems, a function call inside the model has to trigger the execution of the runnable for proper Simulink simulation. The initiator of the function call (e.g. a Stateflow chart or a Function-Call Generator block) is not part of the SWC.

- In those cases where the runnable constitutes an ordinary Simulink subsystem (e.g. a plausible modeling style if the runnable is triggered by a TIMING_EVENT) the execution of the runnable (and its order) is based on the data flow (as usual in Simulink).

The specified RTE events have no impact on the Simulink simulation but define the activation of the runnables in the actual AUTOSAR application running on an ECU.

Purpose

Simulating SWCs in Simulink

Remark

The activation in Simulink (in MIL/SIL/PIL mode) has to be distinguished from the specification of RTE events for runnables. The specified RTE events affect neither the Simulink simulation nor the generated code. They are written to the SWC description file and are used by an RTE generator to properly activate the runnables in the actual AUTOSAR application.
2.8 Using SWC Sender/Receiver Blocks
[TLAR_SENDERRECEIVER_BLOCKS]

The use of SWC sender/receiver blocks is entirely optional. The purpose of those blocks is to provide an appropriate representation of AUTOSAR ports on the model level. If SWC sender/receiver blocks are used, the following rules must be observed:

- SWC sender/receiver ports must be placed outside of runnables.
- Between SWC sender/receiver ports and the corresponding TargetLink inport/outport blocks inside the runnables of an SWC, only SL ports and Bus Creator/Selector/Mux/Demux blocks should be used. Otherwise, the SWC sender/receiver blocks cannot be interpreted as AUTOSAR ports.
- SWC sender/receiver ports must not be used for bidirectional client/server communication, i.e. client/server operations with input as well as output arguments.

Purpose
Modeling SWCs with TargetLink

Remark
SWC sender/receiver ports are a proper representation of AUTOSAR ports with Sender/Receiver interfaces on the model level.

Example
Figure 16 shows the SWC sender/receiver blocks from the tl_autosar_lib library. Figure 17 sketches a common modeling style in which SWC sender/receiver blocks are used in a model to represent AUTOSAR ports.

Figure 16: The SWC sender/receiver blocks from the TargetLink AUTOSAR library. The use of the blocks is optional.
TargetLink AUTOSAR Guidelines

2.9 Modeling Clients for Synchronous Client-Server Communication [TLAR_MODELING_CLIENT]

TargetLink supports the modeling of clients in synchronous client-server communication in two different ways:

- If the operation of the client/server interface has more than one operation argument or if it returns an application error that the client wants to use; then the operation must be modeled using an operation call subsystem.
- In all other cases, i.e. if the operation contains either one input or one output argument and no application error (often referred to as Getter/Setter functions), then the client side of the operation can be modeled using just one TargetLink Inport or TargetLink Outport depending on whether the operation contains an input or output arguments. This applies to scalar signals and arrays (where normal TargetLink ports are used) as well as to a structures (for which TargetLink Busports need to be used).
- For TargetLink versions prior to TargetLink 3.3, structured operation arguments always need to be modeled inside an operation call subsystem. With these old TargetLink versions, it is not possible to use just one TargetLink Busport block for structured operation arguments in Getter/Setter operations.

Purpose
Modeling SWCs with TargetLink

Figure 17: Example of the use of SWC sender/receiver blocks on the model level. AUTOSAR ports are specified in the Data Dictionary and the respective AUTOSAR ports are referenced from the block dialogs of the SWC sender/receiver blocks.
**Remark**

Asynchronous client/server communication, as opposed to synchronous client-server communication, is currently not supported by TargetLink. However, this AUTOSAR feature is hardly ever used in general.

**Example 1:**

To invoke client/server operations with input as well as output arguments, an operation call subsystem has to be used, i.e. a subsystem with a Function/Runnable block (see Figure 18 for an example) along with a proper client/server interface specification in the Data Dictionary, see Figure 19. In the block dialog of the Function/Runnable block, the *Implement subsystem as operation call* option must be set and the proper operation must be referenced, see Figure 20. Each operation argument must be modeled using a TargetLink Inport/Outport block where the block dialog serves to specify the individual operation argument, see Figure 21.

![Figure 18: Example of using an Operation Call subsystem to invoke a client-server operation involving more than just one operation argument. Each Operation Call subsystem has to reside within a runnable from which the operation is invoked. If the operation involves multiple input arguments, then multiple TargetLink Inports have to be used.](image1.png)

![Figure 19: Specification of the *Saturation* operation in the Data Dictionary. The operation is bidirectional, e.g. it contains an input as well as an output signal and must therefore be modeled using an operation call subsystem as shown in Figure 18.](image2.png)
Figure 20: In order to specify the call of an operation of a client/server interface, the *Implement subsystem as operation call* option must be set in the Function/Runnable block. The desired operation in the Data Dictionary must also be referenced.

Figure 21: Each input/output argument of the operation must be modeled using a TargetLink Inport/Outport block where the individual operation argument is referenced from the block dialog.
Example 2:

Examples of client-server communication in the form of simple Get/Set functions are demonstrated in the TargetLink MCDC demo model. It is not necessary to model an entire operation call subsystem here. Instead, simple TargetLink Inport/Outport blocks can be used to model Get/Set operations. Figure 22 shows the modeling style which can be used for simple Get/Set operations. The required specifications in the block dialog of TargetLink Inports/Outports are shown in Figure 23. The operations are specified in the Data Dictionary as indicated in Figure 24.

Figure 22: The *position* block is used to specify a simple Get function, i.e. a client/server operation where just one signal is retrieved. In those situations, it is not necessary to model an entire operation call subsystem as shown in Example 1. A simple TargetLink Inport/Outport block is sufficient.

Figure 23: Referencing a unidirectional client/server operation from a TargetLink Inport block.
2.10 Modeling Servers for Client-Server Operations

[TLAR_MODELING_SERVER]

TargetLink supports the modeling of servers for client/server operations and the modeling has to be carried out in the following manner:

- The server side of a client/server operation, i.e. a so-called server runnable is first and foremost itself a runnable and has therefore to be modeled as a runnable subsystem as described in section 2.5.
- The runnable has to be triggered by an RTE event which is of the OPERATION_INVOKED_EVENT kind. As a matter of fact, that is what makes the runnable a server runnable.
- All ARGIN operation arguments, i.e. operation arguments that are passed from the client to the server have to be modeled using TargetLink InPorts or BusInports for structured arguments. The individual operation argument is selected on the AUTOSAR pane of the Inport.
- All ARGOUT operation arguments, i.e. operation arguments that are passed from the server back to the client have to be modeled using TargetLink Outports or BusOutports for structured arguments. The individual operation argument is selected on the AUTOSAR pane of the Outport.
- If the server operation has a PossibleErrorRef property set, i.e. the server returns an application error, then the application error has to be modeled using a TargetLink Outport inside the runnable subsystem. On the AUTOSAR pane of the Outport, is has to be specified to be an OperationReturnValue.
- If a server runnable is supposed to receive one port-defined argument value (officially supported since TargetLink 3.4), then this has to be modeled using an additional TargetLink Inport in the following manner:
The “Use AUTOSAR communication” checkbox on the AUTOSAR pane of the TargetLink Inport has to be unchecked because the specification of the Port-defined argument value have to be carried out on the Output pane of the TargetLink Inport.

- The "StrictRunnableInterfaceChecks" code generator option has to be unchecked.

- The variable class property on the Output pane of the TargetLink Inport has to be set to FCN_ARG, because the port-defined argument value is passed to the server runnable in the function signature of the runnable.

- The type property has to be set on the Output pane of the TargetLink Inport.

- The server runnable itself can use an arbitrary number of additional, other AUTOSAR communication mechanisms, e.g. use itself client/server communication to retrieve data from another server.

**Purpose**

Modeling SWCs with TargetLink

**Example 1:**

Figure 25 shows a very simple example of a server runnable. As required, the server runnable is triggered by an OPERATION_INVOKED_EVENT, see Figure 26 and implements the operation specified in the Data Dictionary, see Figure 27. The operation has just one argument of the ARG_OUT kind, which is modeled using a TargetLink outport where the ARG_OUT argument is selected, see Figure 28. The server runnable itself calls another client/server operation to get its own input signal. For that purpose, the server runnable contains a TargetLink Inport where another client/server operation is initiated, see Figure 29.

![Figure 25: Modeling a server runnable, i.e. a normal runnable which is triggered through an OPERATION_INVOKED_EVENT, see Figure 26. The runnable sets one output argument as required by the specification of the operation in the DD, see Figure 27. The server runnable uses an additional TargetLink inport to initiate itself another client/server operation to retrieve its input argument, see Figure 29.]
Figure 26: Specification of a server runnable, i.e. a runnable triggered by an OPERATION_INVOKED_EVENT.
Figure 27: The operation implemented by the server runnable. All operation arguments (in this case, there is just one) must be modeled as well as an application error if the server is supposed to return one.

Figure 28: Operation argument of the Client/Server operation which is specified in the server runnable.
Figure 29: A server runnable can execute an arbitrary number of other RTE calls, e.g. like invoking another client/server operation where it acts as a client to get or set data.

2.11 Initialization of AUTOSAR-Related Signals
[TLAR_INIT_ARSIGNALS]

A TargetLink AUTOSAR model contains different kinds of AUTOSAR-related signals, i.e. interrrollable variables or ports which have to be initialized properly for simulation purposes in MIL as well as SIL/PIL simulation modes if the simulation behavior depends on the initialization values. This can be the case if runnables are modeled as conditionally executed subsystems, e.g. function-call-triggered subsystems where the TargetLink outports must be properly initialized. For different kinds of signals, slightly different mechanisms apply to initialize the outports in accordance with AUTOSAR-related specifications:

- Outports for sender/receiver communication can be initialized by referencing the value property of a DD variable object which serves as an AUTOSAR constant specification, see Example 1.
- Outports for interrrollable communication can be initialized by referencing the value property of the interrrollable variable in the Data Dictionary, see Example 2.

Remark
The initialization of AUTOSAR-related signals on the real ECU (except for certain per instance memories) is carried out by the RTE based on the specifications in the software com-
ponent description file. The initialization of TargetLink outports described in the examples below is for simulation purposes in MIL/SIL/PIL mode only.

**Example 1: Initialization of a TargetLink outport for sender/receiver communication in a function-call-triggered runnable subsystem**

The example in Figure 30 shows a runnable whose execution in Simulink is triggered by a function call. The TargetLink Outports of the runnable subsystem (the upi block in this case) can and must therefore be initialized with a value, see Figure 31.

![Figure 30: Outports of a runnable subsystem which is conditionally executed (e.g. triggered by a function call) must be properly initialized for simulation purposes if the initial values have an impact on the simulation behavior.](image)

![Figure 31: Left: Sender/receiver communication is specified for the upi Outport in Figure 30. Right: In order to properly initialize the upi outport, the value property of a DD variable object is referenced which represents the constant value used for initialization of the AUTOSAR port. The TargetLink ddv command is used for this purpose. The Data Dictionary variable object representing the constant is automatically created during ARXML import. Naturally, it can also be created manually.](image)
According to the AUTOSAR standard, initialization values for sender/receiver ports are specified via a reference to a Data Dictionary variable object (called Init_Value_upi here) which is imported and exported during ARXML import/export.

The value property of the Init_Value_upi DD variable object which is referenced from the TargetLink outport in Figure 31 to initialize the port for simulation purposes.

Example 2: Initialization of a TargetLink Outport for interrunnable communication in a function-call-triggered runnable subsystem

The example in Figure 34 shows a runnable whose execution in Simulink is triggered by a function call. The TargetLink outports of the runnable subsystem (the lin_pos block in this case) can therefore be initialized with a value, see Figure 31.
Figure 34: A TargetLink Outport (\textit{lin\_pos} in this example) inside a runnable subsystem which is triggered by a function call. The initialization value of the \textit{lin\_pos} outport must be specified.

Figure 35: \textbf{Left:} Interrunnable communication is specified for the \textit{lin\_pos} outport in Figure 34. \textbf{Right:} In order to properly initialize the \textit{lin\_pos} outport, the \textit{value} property of a DD variable object is referenced which represents the interrunnable variable. The TargetLink \texttt{ddv} command is used for that purpose. The Data Dictionary variable object representing the interrunnable variable is automatically created during ARXML import. Naturally, it can also be created manually.
Figure 36: Data Dictionary variable object representing the interrunnable variable. Initialization values are stored in the Value property of the variable object. Please note that according to the AUTOSAR standard prior to AUTOSAR 4.0.3, interrunnable variables have to be scalar variables.

2.12 Initialization of non-AUTOSAR-Related Signals inside an SWC [TLAR_Init_NonARSignals]

Some signals inside a software component that are not specific AUTOSAR signals need proper signal initialization. These are first and foremost states. Such signals need to be initialized by the code created for the runnables of an SWC, since the RTE does not initialize these signals: They can be initialized in two different ways:

- Initialization at definition
- Initialization in an AUTOSAR runnable of the RestartFunction kind

**Purpose**
Proper initialization of variables where it is needed

**Example 1**
The state variable of the Unit Delay block in Figure 37 can be initialized at definition e.g. by selecting default as the variable class, see Figure 38. The state variable is then directly initialized at its definition, see Figure 39.
Figure 37: Model fragment with a Unit Delay block which has to be part of an AUTOSAR runnable. The state variable of the Unit Delay block needs proper initialization. It can be initialized either at definition (most common case) or in a runnable of the RestartFunction kind.

Figure 38: State of the Unit Delay block. Since the variable class is set to default, the state variable is going to be initialized at its definition, see Figure 39, i.e. somewhere in the files created for the runnable to which the fragment in Figure 37 belongs.

Figure 39: Code fragment created for the definition and initialization of the state variable of the Unit Delay block.

```c
// s1StaticLocalInit: Default storage class for static local variables with initvalue | Width: 32
/*
static sint32 x_S14_Unit_Delay = 0 /*/+ 1.88: 2^-8 @0h: 0 MIN/MAX: -4194304 .. 4194303.99604488 */
```

/*
Example 2
The state variable of the Unit Delay block in Figure 37 is to be initialized in a special restart runnable. For that purpose, a variable class with the required properties shown in Figure 40 must exist. The variable class is then selected in the block dialog of the state variable, see Figure 41, and a runnable of the RestartFunction kind must exist for the SWC to which the variable belongs, see Figure 42. As a consequence, the state variable is now initialized in the restart runnable, see Figure 43.

Figure 40: Variable class with InitAtDefinition set to off and RestartFunctionName explicitly set to “default”, as is required to initialize internal variables of an SWC in a runnable of the RestartFunction kind.

Figure 41: Referencing the variable class from the block dialog to initialize the state variable in a special “Restart runnable”, which is specified in Figure 42.
In order to initialize internal variables of an SWC in a runnable of the `RestartFunction` kind, the SWC must have a runnable with the `Kind` property set to `RestartFunction`. TargetLink will then initialize all internal variables of the SWC with variable class properties `InitAtDefinition` = “off” and `RestartFunctionName` = “default” in the restart runnable.

```c
/**
 * FUNCTION: Init_Controller
 * DESCRIPTION: Restart function for software component: Controller
 * PARAMETERS:
 * RETURNS: void
 * SETTINGS:
*/

void Init_Controller(void)
{
    T_014_Unit_Delay = 0;
}
```

Initialization of the state variable in a separate runnable of the `RestartFunction` kind.
3 Modeling in the Data Dictionary

This chapter contains guidelines that are relevant when specifying AUTOSAR-related data in the Data Dictionary. If AUTOSAR-related data are imported via AUTOSAR XML import, the AUTOSAR-related data are automatically organized in a way that complies with the guidelines in this chapter. Hence, these guidelines are relevant primarily when AUTOSAR-related objects are created in the TargetLink Data Dictionary.

3.1 Required Predefined Data Dictionary Specifications

[TLAR_PREDEFINED_DD]

For AUTOSAR-compliant code generation, i.e. with the code generation mode property being set to “AUTOSAR”, TargetLink requires a number of predefined Data Dictionary settings. Consequently, you have to start an AUTOSAR project with a Data Dictionary template file that is based on the dsdd_master_autosar.dd template file. So, either you directly start from the dsdd_master_autosar.dd template file or you develop your own template file that is derived from it.

Purpose
Modeling SWCs with TargetLink

Remark
The Data Dictionary template file for AUTOSAR-related projects resides under <TL_Root>\Dsdd\Config\dsdd_master_autosar.dd

Please note that the dsdd_master_autosar.dd template file is TargetLink version dependent and it contains different objects for the individual TargetLink versions.

Example
Figure 44 shows a few of the predefined AUTOSAR-related objects which are required in the Data Dictionary project for AUTOSAR-compliant code generation. Figure 45 shows the use of the Data Dictionary Merge Explorer (for TargetLink versions prior to TargetLink 3.3) to merge existing Data Dictionary files with AUTOSAR related objects from dsdd_master_autosar.dd. Since TargetLink 3.4 the Data Dictionary Merge Explorer has been replaced with a general, powerful Data Dictionary Diff&Merge functionality.
Figure 44: Required AUTOSAR-related Data Dictionary objects from the dsdd_master_autosar.dd template.

Figure 45: Required AUTOSAR-related Data Dictionary objects from the dsdd_master_autosar.dd template can be merged with existing Data Dictionary files using the DD Merge Explorer (for TargetLink versions prior to TargetLink 3.3, where it is available under the View menu of the Data Dictionary Manager. Objects can be copied from right to left via context menu (right mouse click). In newer TargetLink versions, a powerful DD Diff&Merge functionality exists.
3.2 Names for AUTOSAR-Related Data Dictionary Objects
[TLAR_DDARNAMES]

The names for those Data Dictionary objects with relevance to AUTOSAR, i.e. interfaces, software components, modes, typedefs, scalings etc. have to comply with the following rules:

- The name lengths of all AUTOSAR-related objects in the Data Dictionary should not exceed 31 characters for AUTOSAR versions prior to AUTOSAR 4.0.
- The names for all AUTOSAR-related objects must be valid C identifiers.

**Purpose**
Compliance with the AUTOSAR standard, i.e. the autosar.xsd schema file

**Remark**
- AUTOSAR standards prior to AUTOSAR 4.0 require the name lengths of all AUTOSAR identifiers to be limited to 31 characters. Please note that this does not apply to RTE macros or to names that include path/package information. In addition, valid C identifiers are required for AUTOSAR elements.
- The validity of AUTOSAR identifiers including their maximum length is specified in the autosar.xsd schema file, see Figure 46.
- TargetLink 3.2 provides the IdentifierWarningLevel import/export option which can be modified to emit warnings when long identifiers are used. The default is set to 32, see the dsdd_autosar_config.xml file, which is part of the TargetLink installation.
- Compliance with the autosar.xsd schema file is checked upon ARXML import/export if the Validate import/export option is set to “On”.
- The Data Dictionary data model itself sometimes requires objects to be valid C identifiers and hence automatically performs those checks. However, this is only the case for some objects like software components but not for all Data Dictionary objects required in an AUTOSAR project.

```xml
<!-- Predefined element and attribute types -->
<xsd:simpleType name="IDENTIFIER">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="^[a-zA-Z][a-zA-Z0-9_]*(0,31)"/>
  </xsd:restriction>
</xsd:simpleType>
```

**Figure 46:** Excerpt from the autosar.xsd schema file for AUTOSAR 3.0 which specifies permitted characters as well as their numbers for AUTOSAR identifiers. The autosar.xsd schema files in TargetLink are stored in subdirectories of TL_Root\Dsdd\Autosar.

**Example**
Incorrect names for AUTOSAR-related objects are shown in Figure 47 and Figure 48.
Figure 47: The lengths of the interface and type examples exceed 31 characters, which must be avoided for AUTOSAR versions prior to AUTOSAR 4.0.

Figure 48: The names of AUTOSAR-related Data Dictionary objects like units must constitute valid C identifiers, unlike in this example.
3.3 Structuring AUTOSAR-Related Data in the Data Dictionary

[TLAR_DDSTRUCTURE]

For proper AUTOSAR workflows and round trips with other AUTOSAR-related tools, the Data Dictionary should have a predefined structure for the following AUTOSAR-related objects:

- Software components
- Interfaces
- Mode declaration groups
- Typedefs
- Scalings

The general rule is that the DD object organization must be carried out exactly in the same manner as the TargetLink AUTOSAR import arranges the DD object. Thereby, identical objects are properly updated in Round-Trip scenarios:

For TargetLink 3.1, 3.2 and TargetLink 3.3, the following rules apply

All AUTOSAR-related objects of the above object kinds must be placed in a subgroup of the top-level group object for the individual object kind. That means the above objects must be placed in the following subgroups:

- /Pool/Autosar/SoftwareComponents/<SWC_GroupName>
- /Pool/Autosar/Interfaces/<Interface_GroupName>
- /Pool/Autosar/ModeDeclarationGroups/<ModeDeclaration_GroupName>
- /Pool/Typedefs/<Typedef_GroupName>
- /Pool/Scalings/<Scaling_GroupName>

The subgroup objects are directly associated with AUTOSAR packages as shown in Figure 49.

Note: Since TargetLink 3.4, AUTOSAR packages can be fully represented using subgroups of the above elements.

Purpose

- Modeling SWCs with TargetLink
- Ensuring proper AUTOSAR round trips and updates in an iterative development process.

Remark

- During ARXML Import, TargetLink automatically structures the newly imported AUTOSAR objects in accordance with this rule if the EnablePackageSupport option is activated (this is the default and recommended). If AUTOSAR-related objects in the Data Dictionary are created from scratch, the rule ensures that those objects get updated properly in an iterative development process with multiple ARXML import/exports.
- Some other AUTOSAR-related objects are exempt from this, as will be described in successive rules.
- Data Dictionary objects which have nothing to do with AUTOSAR are naturally also exempt from this rule.

Example

Figure 49 shows the required structure in the Data Dictionary for AUTOSAR-related objects and TargetLink versions prior to TargetLink 3.4. The TargetLink AUTOSAR demo models shipped with each TargetLink release also demonstrate the required structure in the Data Dictionary.
Figure 49: The required structure for software components, interfaces, mode declaration groups in the Data Dictionary. This applies to TargetLink versions prior to TargetLink 3.4.

### 3.4 Assigning AUTOSAR Packages to Data Dictionary Objects

**[TLAR_PACKAGE_ASSIGNMENTDD]**

Each AUTOSAR-related object in the Data Dictionary which corresponds to a so-called AUTOSAR element must be explicitly assigned package information so that the package (and then also the arxml file) to which the object belongs is clearly defined. AUTOSAR packages for the objects listed in rule 3.3 are specified in the following manner:

- The names of the subgroup(s) from rule 3.3 must correspond to the name of the AUTOSAR subpackage(s).
- The subgroups from 3.3 must have a GroupInfo node where the full AUTOSAR package information is stored in the Package property.

**Purpose**

- Proper AUTOSAR round trips with other AUTOSAR-related tools.
- Proper partitioning of AUTOSAR elements into files.

**Remark**

- The package information also influences the file partitioning upon TargetLink’s ARXML export if the ExportStrategy property is set accordingly.
- If an AUTOSAR element in the Data Dictionary has no package information specified for it (which must not be the case according to this rule), it is assigned a file and a package based on the file name specified during ARXML export. That should be avoided.
- Package information for unit objects is specified differently, see 3.8.

**Example**

Figure 50 shows the specification of AUTOSAR package information for SWCs. The TargetLink demo models shipped with each TargetLink release also demonstrate the specification of package information in the Data Dictionary.
Figure 50: Specifying package information for software components in the Data Dictionary: In order to place AUTOSAR-related objects in packages, group objects and associated GroupInfo nodes have to be created first (here a software component group called Sensors). The package information is stored in the Package property of the GroupInfo node. The top level package is called ApplicationSWCs and the subpackage is called Sensors, which is also required to be name of the subgroup. The full AUTOSAR package information is therefore ApplicationSWCs/Sensors.

3.5 Performing ARXML Imports and Outports with Package Support

[TLAR_PACKAGE_SUPPORT]

The EnablePackageSupport option must be set to “on” as a default so that AUTOSAR package information is always taken into account upon AUTOSAR import/export.

Purpose
Proper AUTOSAR round trips with other AUTOSAR-related tools

Example
The specification of the EnablePackageSupport option depends on the TargetLink version used. Figure 51 shows the content of the dsdd_autosar_config.xml file stored under the <TL_Root>\Dsdd\config directory in TargetLink 3.1. The file serves to store the default ARXML import and export options unless other values are explicitly specified. Figure 52 shows the EnablePackageSupport option in the TargetLink AUTOSAR Import dialog of TargetLink 3.1. In newer TargetLink versions like TargetLink 3.3, AUTOSAR import/export default options are specified under the /Pool/Autosar/Config/ImportExport node, see Figure 53.
Figure 51: The EnablePackageSupport option specified in the dsdd_autosar_config.xml file must be set to “On”, which is also the default in TargetLink.

Figure 52: To support package information, the Enable package support option must not be cleared upon import/export of SWC description files.
3.6 Modeling Types in AUTOSAR Projects

[TLAR_TYPESDD]

For the specification of types used for AUTOSAR signals (data elements, operation arguments, interrunnable variables, per instance memories, etc.) the following rules must be observed:

- All AUTOSAR-related types must reside in typedef groups with proper package assignment as described in 3.3 and 3.4.
- The type name must be a valid C identifier with proper length as described in 3.2.
- For TargetLink versions prior to TargetLink 3.4, only user types must be used for AUTOSAR signals, i.e. the BaseType property must be “off”. The predefined-data types like In16, Int32, directly residing under /Pool/Typedef, must not be used. Similarly, the predefined AUTOSAR-related types under /Pool/Typedef/platformtypes must not be used for AUTOSAR signals either.
- All integer AUTOSAR types must have a constraints subnode with a specified scaling object. Minimum and maximum values for the type can be specified as constraints as well.
- Types which are used for AUTOSAR signals must have a ModuleRef property either set to n.a. or Rte_Type.
- For TargetLink versions prior to TargetLink 3.4, types which are used only for internal signals of an SWC/runnable but not for AUTOSAR signals must not have a ModuleRef property that is set to Rte_Type.
Purpose
Modeling SWCs with TargetLink

Example
Figure 54 shows some of the predefined data types as well as base types which must not be used for AUTOSAR signals in older TargetLink versions. Figure 55 shows how a scaling object must be referenced for AUTOSAR integer types. Figure 56 shows an impermissible ModuleRef value for a type that is used for an AUTOSAR signal.

Figure 54: The predefined data types under /Pool/Typedefs/PlatformTypes must not be used for TargetLink versions prior to TargetLink 3.4. Similarly, the TargetLink base types like Int16, Int32, UInt8 residing directly under /Pool/Typedefs, must not be used for older TargetLink versions.

Figure 55: All AUTOSAR Integer data types must have a Scaling object referenced at the Constraints subnode of the type. Moreover, Min and Max values can be specified for the type.
3.7 Specifying AUTOSAR Scalings (CompuMethods) in the Data Dictionary [TLAR_SCALINGDD]

For the specification of scaling objects used for AUTOSAR signals (data elements, operation arguments, interr Runnable variables, per instance memories etc.), the following rules must be observed:

- The scaling must reside in a subgroup with proper package assignment as described as described in 3.3 and 3.4.
- The scaling name must be a valid C identifier with proper length as described in 3.2. Please note that the data model of the Data Dictionary permits other identifiers, too.
- If an AUTOSAR unit has to be associated with the scaling, then the Unit object must be referenced from the UnitRef property of the scaling object. It must not be entered as a string in the Unit property.

Purpose
Proper AUTOSAR round trips with other AUTOSAR related tools.

Remark
Data Dictionary scaling objects correspond to so-called CompuMethods in the AUTOSAR standard. Please note, that the above rules only apply to those scalings that are used for AUTOSAR signals. They need not be observed for signals used internally inside an SWC.

Example
Figure 57 shows the proper specification of a scaling object that can be used for AUTOSAR signals.
Figure 57: The scaling object SC_ref stored in a Subgroup object called DataTypeSemantics of /Pool/Scalings. The scaling references an AUTOSAR unit called MyMeter, which is stored under /Config/Units.

3.8 Specifying AUTOSAR Units in the Data Dictionary

For the proper specification of units used for AUTOSAR signals (Data Elements, Operation Arguments, Interrunnable Variables, Per Instance Memories etc.) the following rules must be observed:

- The unit object must be created under /Config/Units. No subgroup object can/must be created.
- The AUTOSAR package to which the unit belongs must be specified in the package property of the unit object.
- The unit name must be a valid C identifier with proper length as described in 3.2. Please note that the data model of the Data Dictionary permits other identifiers too and hence does not provide automatic checks for that.
- The unit object must only be referenced from a scaling object as shown in Figure 57.

Purpose
Proper AUTOSAR round trips with other AUTOSAR related tools.

Remark
Please note that the above rules only apply to those units that are used for AUTOSAR signals. They need not be observed for signals used internally inside an SWC.

Example
Figure 58 shows the proper specification of a unit object that can be used for AUTOSAR signals.
Figure 58: Specifying AUTOSAR packages for units. Unlike for software components, interfaces etc., the AUTOSAR package information is directly stored at the Package property of individual unit objects. All units are directly stored below the /Config/Units node in the Data Dictionary and no subgroups can/must be created.

3.9 Specifying Sender/Receiver or Mode Switch Interfaces in the Data Dictionary [TLAR_SRINTERFACEDD]

For the specification of sender/receiver interface objects in the Data Dictionary, the following rules must be observed:

- The interface object must reside in a subgroup of /Pool/Autosar/Interfaces with proper package assignment as described in 3.3 and 3.4.
- The interface name must be a valid C identifier with proper length as described in 3.2.
- The names of the data elements or mode elements in the interface must be a valid C identifiers with proper lengths as described in 3.2.. Please note that the data model of the Data Dictionary makes sure that they are valid C identifiers.
- It is recommended to specify sender/receiver interfaces with either data elements only or mode elements only but not a mixture of both. Since AUTOSAR 4, it is also no longer possible to mix data elements and mode elements in one interface.
- The width property must be set for data elements that are arrays.
- The type property must be set for all data elements.
- It is recommended to not specify a scaling object in the Scaling property. The scaling object must be indirectly specified via the referenced type as described in rule 3.6.
- Since TargetLink 3.1p1, it is possible to specify values for the Min and Max properties of a data element if its range is a subset of the range specified for the referenced type.

Purpose
Proper AUTOSAR round trips with other AUTOSAR related tools.
Example
Figure 59 shows the proper specification of a sender/receiver interface with a data element.

Figure 59: Proper specification of a sender/receiver interface with just one data element. The interface object `IF_upi` is stored in a Subgroup object of `/Pool/Autosar/Interfaces` called `PortInterface`. The data element `upi` references the type of the signal but no scaling object.

3.10 Specifying Client/Server Interfaces in the Data Dictionary

For the specification of client/server interface objects in the Data Dictionary, the following rules must be observed:

- The interface object must reside in a subgroup of `/Pool/Autosar/Interfaces` with proper package assignment as described in 3.3 and 3.4.
- The interface name must be a valid C identifier with proper length as described in 3.2. Please note that the data model of the Data Dictionary automatically checks that it is a valid C identifier.
- The names of the operation arguments in the interface must be valid C identifiers with proper lengths as described in 3.2. Please note that the data model of the Data Dictionary makes sure that it is a valid C identifier.
- The `width` property must be set for operation arguments that are Arrays.
- The `type` property must be set for all operation arguments.
- It is recommended to not specify a scaling object in the `Scaling` property. The scaling object must be indirectly specified via the referenced type as described in rule 3.6.
- Since TargetLink 3.1p1, it is possible to specify values in the `Min` and `Max` properties of an operation argument if its range is a subset of the range specified for the referenced type.
- Operation Arguments that are used in a TargetLink model must be either of the ARGIN or ARGOUT kind. TargetLink currently does not support ARGINOUT arguments for Client/Server operations for usage in TargetLink models. However, a specification in the Data Dictionary is possible.

Purpose
Proper AUTOSAR round trips with other AUTOSAR-related tools
Example
Figure 60 shows the proper specification of a client/server interface with one operation.

3.11 Specifying Interrunnable Variables in the Data Dictionary
[TLAR_IRVDD]

For the specification of interrunnable variables in the Data Dictionary, the following rules must be observed:

- All interrunnable variables of an SWC and its internal behavior must be represented as Data Dictionary variable objects, which should be created in the /Pool/Variables/<SWC_Name>/IRV subgroup.
- The above subgroup must be explicitly specified in the InterrunnableVariablesRef property of the RelatedVariables object beneath the software component.
- Since the interrunnable variable automatically belongs to the internal behavior of a software component, no package information can/must be specified for it.
- The name of the interrunnable variable must be a valid C identifier with proper length as described in 3.2. Please note that the data model of the Data Dictionary does not automatically check that it is a valid C identifier.
- The interrunnable variable requires specific settings for the Class and NameTemplate property as described in the official TargetLink AUTOSAR Modeling Guide (The TargetLink AUTOSAR Utilities provide additional support for that also).
- The Value property must be set to the initialization value if proper initialization of the Interrunnable Variable by the RTE is required.
- The width property must not be set since interrunnable variables need to be scalar as required by the AUTOSAR standard prior to AUTOSAR 4.0.3.
- The type property must be set to assign a proper type.
- No local scaling object must be created for an interrunnable variable since such scalings are not supported by the AUTOSAR standard.
Purpose
Proper modeling of AUTOSAR SWCs with TargetLink

Example
Figure 61, Figure 62 and Figure 63 demonstrate the specification of interrunnable variables in the Data Dictionary.

Figure 61: Proper specification of an interrunnable variable. The InitConstantName, InitConstantUuid and InitConstantPackage properties can be used to assign a constant object to the initialization value of the interrunnable variable (0 in this case).

Figure 62: Local scaling objects are not permitted for Interrunnable Variable objects. Instead, scalings must be referenced by the respective types of the variables.
Figure 63: It is recommended to explicitly specify the Variable Group object to save interrunnable variables to, i.e. /Pool/Variables/<SWC_Name>/IRV.
4 Limitations

This chapter sums up some of the limitations most relevant when modeling AUTOSAR software components with TargetLink.

4.1 Restricted Subset of the Software Component Template [TLAR_ARSUBSET]

TargetLink supports a subset of all the AUTOAR features defined in the Software Component Template. A detailed list of all the features and the extent to which they are supported by TargetLink is available from the official TargetLink Support via support.tl@dspace.de in the form of an Excel sheet. These limitations must be observed when developing SWCs with TargetLink.

Purpose
Modeling SWCs with TargetLink

4.2 Dealing with AUTOSAR Compositions [TLAR_COMPOSITIONS]

TargetLink does not support AUTOSAR compositions (a collection of individual SWCs wrapped up in one SWC called a composition) but only atomic software components. In fact, compositions are outside the scope of TargetLink and should be handled in AUTOSAR architecture tools like dSPACE SystemDesk.

Purpose
Modeling SWC with TargetLink

Remark
The ARXML import mechanism in TargetLink ignores compositions as such but imports the atomic software component contained in the files and referenced from compositions. After code generation, the individual atomic software components can be exported in SWC description files though they are not wrapped up in a composition. It is also possible to simulate multiple software components (which might be part of a composition) but it is the responsibility of the user to connect the different components in the model (i.e. connectors of a composition are neither imported nor automatically created in a TargetLink model).

Example
Figure 64 shows an excerpt from an SWC description file containing a composition of two atomic software components. Such an AUTOSAR xml-file can be imported into the dSPACE Data Dictionary, but only atomic software components are imported, see Figure 65. After code generation, the atomic software components can be exported in the form of an SWC description file though without any reference to a composition.
Figure 64: An extract from an SWC description file containing a composition named Composition_SWC1_SWC2, which itself contains two atomic software components named ASWC1 and ASWC2.

Figure 65: After import of the xml-file from Figure 64, the atomic software components ASWC1 and ASWC2 are stored in the Data Dictionary without any reference to a composition.
4.3 Multiple Instantiation of Software Components

[TLAR_MULTIPLEINSTANCES]

TargetLink versions prior to TargetLink 3.5 do not support multiple instances of the same AUTOSAR software component type. Consequently, the supportsMultipleInstantiation property of software component objects in the Data Dictionary must not be set to on for those TargetLink versions.

Note:
Since TargetLink 3.5, software component types can be generated for multiple instantiation. Consequently, the supportsMultipleInstantiation property must be set to on or off depending on what is desired.

Workaround:
If the same functionality is to be deployed, use code duplication instead of multiple instantiation.

Purpose
Modeling SWCs with TargetLink.

Remark
- Multiple instances of AUTOSAR software components are used to share the same code across multiple instances of a SWC on one ECU and hence serve to reduce code size and testing efforts. However, this AUTOSAR feature is not supported by TargetLink versions prior to TargetLink 3.5.
- Since TargetLink 3.5, you can generate code for software component prototypes which can be instantiated multiple times on an ECU or e.g. in dSPACE VEOS. However, you cannot instantiate the same software component multiple times in TargetLink itself.

Example
Figure 66 shows the supportsMultipleInstantiation property of SWCs which specifies whether a SWC supports multiple instantiation. Since TargetLink versions prior to TargetLink 3.5 do not allow multiple instances of a SWC, the property must be set to off (or <n.a.> if the RTE generator does not require this property to be set).

Figure 66: The supportsMultipleInstantiation property of software components in the Data Dictionary must not be set to “on” for TargetLink versions prior to TargetLink 3.5.
4.4 Restricted Simulation Support for Sender-Receiver Communication with Event Semantics

[TLAR_EVENTSEMANTICSSIMULATION]

Sender-receiver communication for data elements with specified event semantics is simulated in MIL/SIL/PIL mode like ordinary Simulink-like data element semantics. Consequently, no queues for those data elements are simulated and e.g. no queue overruns can occur.

Note: In an AUTOSAR application, data elements with event semantics possess a queue that individual values are read from and written to. This behavior cannot be properly simulated in Simulink, so that TargetLink uses the ordinary data element semantics without a queue for simulation purposes in the way familiar to Simulink users.

Purpose
Simulating SWCs with TargetLink

Remark
Specifying whether a data element has event semantics (and an associated queue) or not has an impact on the code of the SWC itself, since the RTE macros for data and event semantics differ. The specification is also part of the SWC description file. When the overall AUTOSAR application is built, it is the responsibility of the RTE generator to provide the required queue.

Example
The specification for a data element to have a queue and therefore event semantics is shown in Figure 67. The property is supported for the import and export of SWC description files, but ignored during simulation.

Figure 67: The IsQueued property of a data element determines, whether the data element has event semantic (isQueued = on) or not (isQueued = off). In order to specify the queue length in the Data Dictionary, an EventReceiverComSpec-object has to be specified for the associated port.
4.5 AUTOSAR SWCs and Incremental Code Generation using Model Referencing/Incremental Subsystems

[TLAR_INCREMENTALCG]

When model referencing/incremental code generation is used for modeling AUTOAR software components, the following rules apply:

- For TargetLink 3.1, TargetLink 3.2, TargetLink 3.3, all AUTOSAR-related signals must remain outside of referenced models, incrementally generated subsystems. Model referencing/incremental code generation can be used for the interior of AUTOSAR runnables as long as no AUTOSAR signals cross the border of the referenced model/incrementally generated subsystem.

- Since TargetLink 3.4, incremental code generation using model referencing/incrementally generated subsystems is supported in the following manner:
  - You can model an arbitrary number of software components in a referenced model/incrementally generated subsystem.
  - Runnables of the same software components can be developed in separate referenced models/incrementally generated subsystems.
  - Runnables can be generated incrementally using model referencing or incrementally generated subsystems.
  - Parts of Runnables (as opposed to the whole runnable, see above) can be developed incrementally using model referencing/incrementally generated subsystems as long as the runnable part contains only Per Instance Memories and AUTOSAR Calibration parameters. Please note that the runnable fragment to be developed separately must always be embedded in a runnable subsystem so that TargetLink knows to which software component and runnable the part belongs.

- Since TargetLink 3.5, fragments of a runnable can contain not only Per Instance Memories and AUTOSAR Calibration parameters but also operation call subsystems.

Purpose
Modeling AUTOSAR SWCs with TargetLink
5 Guidelines for SWC Container Exchange between SystemDesk and TargetLink

This chapter deals specifically with guidelines that are relevant to using SWC container exchange between TargetLink and SystemDesk. The SWC container exchange was introduced with TargetLink 3.2/SystemDesk 3.0 in order to improve AUTOSAR workflows, increase transparency, ensure safe AUTOSAR round trips and provide easy access to SystemDesk’s/VEOS simulation capabilities for TargetLink AUTOSAR users. This chapter contains modeling guidelines for both SystemDesk and TargetLink (mostly SystemDesk) in order to ensure smooth SWC container exchange. The guidelines in this chapter are therefore not relevant to users who do not use SystemDesk.

Note: SWC container exchange between SystemDesk and TargetLink can only be used with SystemDesk versions as recent as SystemDesk 3.0 and TargetLink versions as recent as TargetLink 3.2.

5.1 General Guidelines for SystemDesk and TargetLink

5.1.1 Specification of Workflow Rules [TLAR_CONTAINER_WORKFLOW]

SWC container exchange between SystemDesk and TargetLink is highly configurable and can be adapted to company-specific requirements. In older SystemDesk and TargetLink versions, the ContainerWorkflowDefinition.ctw files in the SystemDesk and TargetLink installation have to be adapted for this. Since TargetLink 3.4 and SystemDesk 3.2, the file can be selected in the preference settings of the Container manager. Among the things to be configured are:
- Folder structure of the different files in the SWC container
- Naming schemes for the automatic assignment of file categories
- Workflow rules which determine container synchronization based on file categories

Purpose
Proper company-specific AUTOSAR workflows

Example
Figure 68 and Figure 69 show excerpts from a container workflow definition file which can be adapted by the user. Figure 70 shows the selection of a workflow definition file in TargetLink 3.4.

5.1.2 Use of Container Set Files for SWC Container Exchange [TLAR_CONTAINER_CONTAINERSETS]

For container import and container synchronization, container set files should be selected in the provided dialogs. After that, the individual container/SWC can be selected from the Catalog History control. There is no need for the user to work with container catalog files directly although this is possible also.

Purpose
Efficient container exchange between TargetLink and SystemDesk

Example
Figure 71 and Figure 72 show how to import SWC containers into a SystemDesk or TargetLink project. The container import is started and (see Figure 72) a container set file is selected first. The container is then selected from the Catalog History.
Figure 68: Excerpt from the workflow definition file contained in the TargetLink and SystemDesk installation, which can be adapted by the user to influence the container export/import/synchronization behavior of SystemDesk and TargetLink in order to meet company-specific workflow requirements.

---

```xml
<WorkflowRuleSet>
  <Description>From TargetLink to SystemDesk</Description>
  <Condition>((Source.Owner) == 'TargetLink') AND ((Target.Owner) == 'SystemDesk')</Condition>
  <FileRule>
    <Description>Delete obsolete code files generated by TargetLink.</Description>
    <Condition>
      (Resource.DestinationDirectory) == 'TargetLink' AND
      (Resource.SourceFile) == 'TargetLink'
    </Condition>
    <Operation>Delete</Operation>
  </FileRule>
</WorkflowRuleSet>
```

Figure 69: Another excerpt from the workflow definition file.
Figure 70: Setting the workflow definition file that controls the SWC container exchange using the Container manager GUI from TargetLink 3.4

Figure 71: Performing a container import in SystemDesk.
5.2 SystemDesk Guidelines for SWC Container Exchange

This section contains guidelines that have to be observed when working in SystemDesk.

5.2.1 Partitioning Software Components into Packages

The following rules apply to the partitioning of software components into different packages:

- Each and every software component that is exchanged with TargetLink must reside in its own package. Placing multiple software components in one package is not permitted.
- The name of the package should match the name of the software component. If that is not the case or if a component resides within a subpackage, then the name of the container must be explicitly specified in TargetLink as described in subsection 5.3.3.

Purpose:

- Ensuring proper partitioning of different software components into separate files
- Ensuring identical names for container catalog files in SystemDesk and TargetLink

Remark:

- The above restrictions will be eliminated in future TargetLink versions.
- The PackageSynch script from the SystemDesk solutions pool http://www.dspace.com/goto?SystemDeskSolutions can be used to automatically create the package structure based on the object and folder structure in SystemDesk’s
In order to comply with the above guideline, SWCs must reside in different folders.

**Example**
The example in Figure 73 shows two different SWCs (ControllerSWC and SignalProcessingSWC) residing in different packages as required by this guideline. The ControllerSWC resides in a package of identical name, i.e. ControllerSWC and there is no subpackage. Since the name of the container exported from SystemDesk is based on the package/subpackage name, the container name is ControllerSWC, see Figure 74. That is also the name that TargetLink will use for the container upon export from TargetLink, so it is not mandatory to separately specify the container name in TargetLink as described in subsection 5.3.3. The SignalProcessingSWC component however resides in a subpackage. The full package path is Processing/SignalProcessingSWC. Consequently, the name of the container exported for the SWC from SystemDesk will be Processing.SignalProcessingSWC (Package.Subpackage in general), see Figure 74. In this case, the name of the container file must be explicitly specified in TargetLink as shown in subsection 5.3.3 to achieve identical names for SystemDesk and TargetLink.
**Figure 73:** Different SWCs (ControllerSWC and SignalProcessingSWC in this example) must reside in different packages (see the Package Manager on the right of the screenshot). They must not be combined in one package. The PackageSynch script from the SystemDesk solutions pool http://www.dspace.com/goto?SystemDeskSolutions can be used to automatically create the package structure based on the object and folder structure in the Project Manager (see the left part of the screenshot). For that purpose, SWCs must reside in different folders.

![Figure 73](image)

**Figure 74:** Exported containers for the SWCs ControllerSWC and SignalProcessingSWC from Figure 73. Since the names of the containers exported from SystemDesk is based on the package information of the SWCs, the container catalog files exported from SystemDesk are called ControllerSWC and Processing.SignalProcessingSWC.

5.2.2 Package Assignment for all Objects in the SystemDesk Project 
[TLAR_CONTAINER_SD_PACKAGES]

Each and every object in the SystemDesk project that is exchanged with TargetLink must be assigned to an AUTOSAR package.

**Purpose:**
Ensuring proper partitioning of all AUTOSAR elements into files and packages

**Remark:**
The PackageSynch script from the SystemDesk solutions pool http://www.dspace.com/goto?SystemDeskSolutions can be used to automatically assign all objects in SystemDesk's Project Manager to a package.

5.2.3 Separation of Shared and Component-specific Data 
[TLAR_CONTAINER_SD_DATASEPARATION]

Component-specific elements like SWCs must be placed in different packages and files than those elements which are shared among multiple developers, like interfaces and data types. More specifically, at least the following different packages must exist:

- A package for each software component along with its internal behavior and implementation. The SWC, internal behavior and implementation must be placed in the same package.
- Interfaces must reside in separate packages that contain nothing but interfaces.
- Data types and scalings must reside in a separate package where they are not mixed up with components and interfaces.

**Purpose**

Proper partitioning of component-specific and shared data into separate packages and files and hence clear responsibilities of TargetLink/SystemDesk users for different files.

**Remark**

The responsibility for shared data like interfaces or data types must lie with the SystemDesk user/software architect.

**Example**

Figure 75 shows an example of a proper structure of the SystemDesk Project Manager/Packages Manager to comply with the above guidelines.

![Figure 75: Example of a proper structure in the Project Manager and Package Manager to separate component-specific data on one hand and shared data on the other.](image)
5.2.4 File Names for AUTOSAR Packages
[TLAR_CONTAINER_SD_FILENAMES]

In order to ensure smooth SWC container exchange with TargetLink, the ARXML file names exported from SystemDesk must comply with the following rules:

- For packages without any subpackages, the ARXML file names (without extensions) must match the names of the packages.
- For subpackages with the structure MyPackage/MySubpackage/MySubsubpackage, for example, the names of the arxml files must be MyPackage.MySubpackage.MySubsubpackage.

**Purpose**
Ensuring identical ARXML file names for SystemDesk and TargetLink

**Remark**
- TargetLink 3.2 uses the above naming scheme for ARXML files, in which the file names are automatically created based on the package names.
- The above restrictions will be eliminated in future TargetLink versions.
- The PackageSynch script from the SystemDesk solutions pool [http://www.dspace.com/goto?SystemDeskSolutions](http://www.dspace.com/goto?SystemDeskSolutions) can be used to automatically create the packages and file names based on the object and folder structure in SystemDesk’s Project Manager.

**Example**
Figure 76 shows an example of proper file names in accordance with the above requirements.

5.2.5 Specifying Workflow Rules for SystemDesk
[TLAR_CONTAINER_SD_WORKFLOW]

The SWC container exchange between SystemDesk and TargetLink is highly configurable and can be adapted to company-specific requirements. In order to influence SystemDesk’s import and export of SWC containers prior to SystemDesk 3.2, the following file must be adapted:

<SystemDesk_Root>\Config\ContainerWorkflowDefinition.ctw
where <SystemDesk_Root> designates the root directory of the SystemDesk installation.

**Purpose**
Proper company-specific AUTOSAR workflows

**Remarks**
The specifications in the above container workflow definition file are also relevant when the Container Manager is started from SystemDesk.

**Example**
Figure 77 shows an excerpt from the container workflow definition file which can be adapted by the user.
Figure 76: Proper ARXML file names in accordance with the package structure to ensure proper tool-interoperability with TargetLink. Where subpackages are used, the ARXML file name must be a concatenation of all packages and subpackages, each separated by a dot.
Figure 77: Excerpt from the container workflow definition file, which is part of the SystemDesk installation and which can be adapted by the user to influence the container export/import/synchronization behavior of SystemDesk in order to meet company-specific requirements.

5.3 TargetLink Guidelines for the SWC Container Exchange

This section contains guidelines that have to be observed when working in TargetLink.

5.3.1 Software Components and TargetLink Subsystems

For TargetLink versions prior to TargetLink 3.4, different SWCs should be developed in different TargetLink models when using the SWC container exchange. Since TargetLink 3.4, it is sufficient if different SWCs are developed in different code generation units, i.e. TargetLink subsystems or incrementally generated subsystems or referenced models.

Purpose

Proper SWC container exchange with SystemDesk
Example

Figure 78 shows a TargetLink model that contains multiple TargetLink subsystems in order to develop multiple SWCs within one model as it is supported since TargetLink 3.4.

**Figure 78:** If multiple SWCs are modeled within one TargetLink model, they should be placed in different code generation units like different TargetLink subsystems as supported since TargetLink 3.4. Here, 4 different SWCs are modeled, each residing in a separate TargetLink subsystem.
5.3.2 Specifying Workflow Rules for TargetLink

[TLAR_CONTAINER_TL_WORKFLOW]

SWC container exchange between SystemDesk and TargetLink is highly configurable and can be adapted to company-specific requirements. In order to influence TargetLink’s import and export of SWC containers prior to TargetLink 3.4, the following file must be adapted:

`<TargetLink_Root>/Dsdd/Config/ContainerWorkflowDefinition.ctw`

where `<TargetLink_Root>` designates the root directory of the TargetLink installation.

**Purpose**

Proper company-specific AUTOSAR workflows

**Remarks**

The specifications in the above container workflow definition file are also relevant when the Container Manager is started from TargetLink.

**Example**

Figure 79 shows an excerpt from the container workflow definition file which can be adapted by the user.

---

Figure 79: Excerpt from the container workflow definition file, which is part of the TargetLink installation and which can be adapted by the user to influence the container export/import/synchronization behavior of TargetLink in order to meet company-specific requirements.
5.3.3 Specifying the Name of the Container Catalog File

[TLAR_CONTAINER_TL_CATALOGNAME]

The name of the Container Catalog for a SWC to be exported from TargetLink should be explicitly specified using the Catalog property below the SWC node in the Data Dictionary. In the following situations, the specification of the name is not just recommended but mandatory (see also subsection 5.2.1):

- If a SWC resides not just in a package but in a subpackage or
- If the name of the package differs from the name of the SWC itself.

The Catalog property must be set based on the package structure of the SWC by concatenating the different packages and subpackages each separated by a dot. If for instance, a SWC resides in the package MyPackage/MySubPackage/MySubSubPackage, then the Catalog property must be set to MyPackage.MySubPackage.MySubSubPackage.

Purpose
Ensuring identical names for SWC Containers exported from TargetLink and SystemDesk

Remarks
For proper synchronization of SWC containers between TargetLink and SystemDesk, containers exported for the same SWC need to have identical names. This guideline ensures that this is always the case.

Example
Figure 80 shows the explicit specification of the container catalog file name for a SWC using the Catalog property. As a consequence, container catalog names exported from both SystemDesk and TargetLink are identical and can be properly compared using the Container Manager, see Figure 81.

Figure 80: Specifying a name for the container catalog for a SWC in the Data Dictionary. The specification is taken into account upon container export from the Data Dictionary. In this example the SWC ASP_Legacy resides in the package ASP/ASP_Legacy (specification is not entirely visible in this example). Consequently, the Catalog property must be set to ASP.ASP_Legacy in accordance with the package structure.
Figure 81: Comparison of different versions of a container catalog (ASP.ASP_Legacy in this example) for the same SWC with the Container Manager (left side exported from TargetLink, right side exported from SystemDesk). For proper comparison and synchronization, container catalogs for the same SWC must have identical names in SystemDesk and TargetLink. That is why the specification of a catalog name in the Data Dictionary according to Figure 80 is sometimes necessary and hence generally recommended.
6 List of References


Excel sheet with a detailed list of all sorts of AUTOSAR features, indicating whether or not they are supported by TargetLink and SystemDesk. The Excel sheet is available from the TargetLink support via support.tl@dspace.de