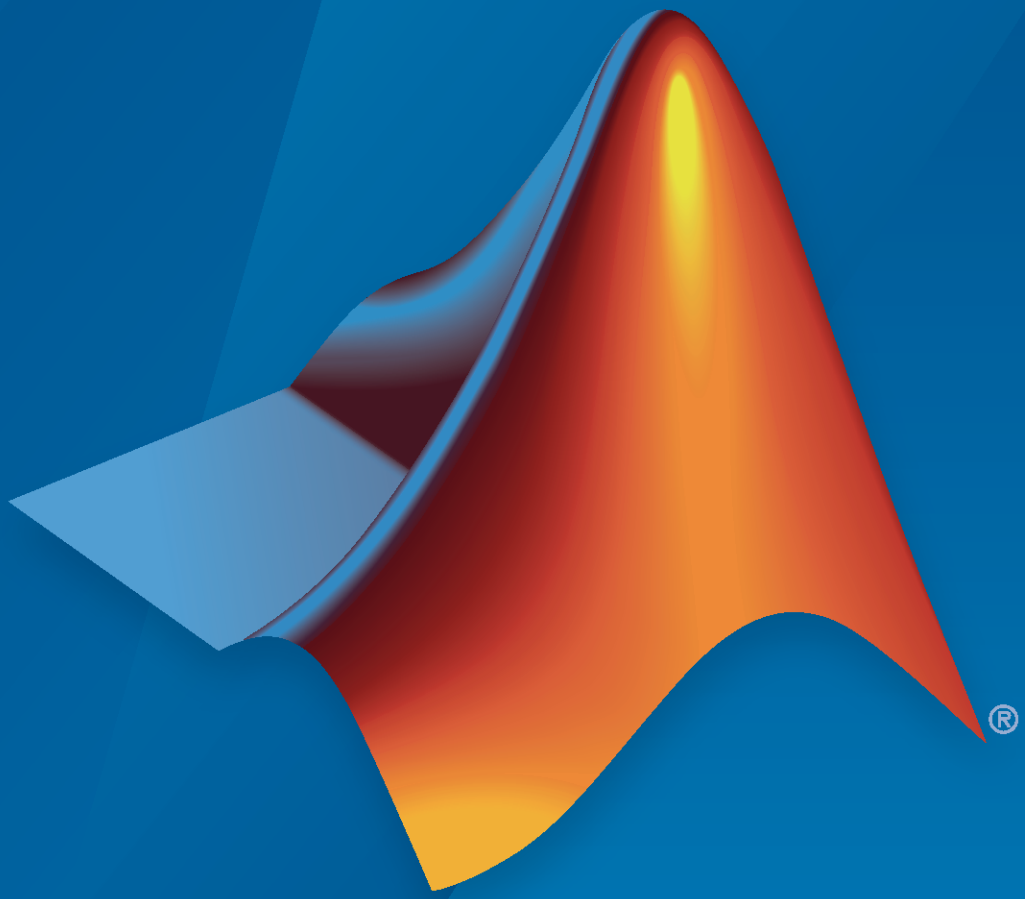
AUTOSAR Blockset Release Notes



MATLAB® & SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

AUTOSAR Blockset Release Notes

© COPYRIGHT 2019–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Contents

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2022b

Version: 3.0

New Features

Compatibility Considerations

AUTOSAR Software Component Modeling

AUTOSAR Classic Platform Release 20-11

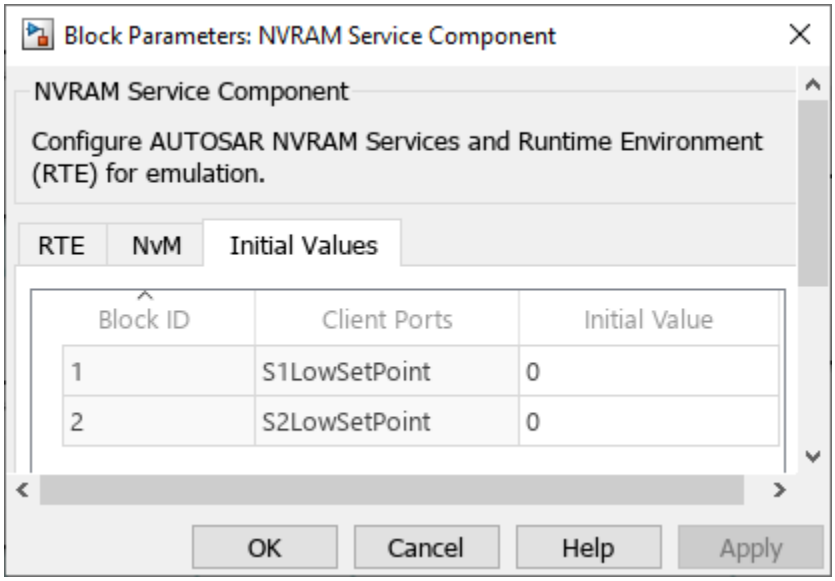
R2022b extends support of the AUTOSAR Classic Platform to include Release 20-11. AUTOSAR Blockset supports schema version R20-11 for import and export of ARXML files and generation of AUTOSAR-compliant C code.

If you import schema R20-11 ARXML files into Simulink®, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model.

If you create an AUTOSAR classic model in Simulink, the initial default schema version is R20-11. To change the schema version, use the model configuration parameter **Generate XML file for schema version** or an equivalent function call. For more information, see Select AUTOSAR Classic Schema.

Initial values for NVRAM service component blocks

R2022b enables you to define initial values for NVRAM services that are accessed during simulation by basic software. To configure the initial values, you can open the NVRAM service component block parameters, click the **Initial Values** tab, and set an initial value for each NVRAM block.



You can specify initial values with parameter objects to support common test workflows. Initial values can be specifiable for all Simulink data types, including standard types, fixed point, arrays, structured types, and enumerations. For more information, see NVRAM Service Component. For an example of how to use NVRAM service component to simulate ECU for basic software, see “Simulate AUTOSAR Basic Software Services and Run-Time Environment”.

Import for Basic Software blocks

Starting in R2022b, when importing a component with accesses to client ports for Basic Software operations, a basic software caller block is created. This creation of basic software blocks during ARXML import supports `createComponentAsModel`, `updateModel`, and composition import

workflows to support Dem, NvM and FiM function calls. You should now be able to see AUTOSAR Blockset caller blocks from the shipping libraries created in models. For more information, see “Component Creation”.

Implementation data types reference AUTOSAR implementation platform types

R2022b allows you to satisfy AUTOSAR data type specification requirements related to AUTOSAR platform types. To specify AUTOSAR platform types, you can use the AUTOSAR Dictionary **XML Options** tab to configure the new AUTOSAR platform type parameters:

- **Implementation Platform Types Package** – Specify the top-level package name for AUTOSAR platform types and base types. AUTOSAR implementation platform types are grouped in the `ImplementationDataTypes` subpackage, while the base types are grouped in the `BaseTypes` subpackage. For `Modular` ARXML export, the top-level package and its content is now exported in the `stub/XXXXX_platformtypes.arxml` file.
- **User-defined Implementation Types References** – Specify the implementation data type references behavior. If you select `PlatformTypeReference` your user-defined implementation data types reference an AUTOSAR implementation data type (`CATEGORY` is set to `Type_Reference` in the ARXML). If you select `BaseTypeReference` your user-defined data type references a SWBaseType (`CATEGORY` is set to`Value` in the ARXML).
- **Native Declaration** – Specify the native declaration. If you select `PlatformTypeName`, the native declaration inherits the Platform Type name. If you select `CIntegralTypeName`, the native declaration uses a C integral type name according to the hardware configuration specified in the model settings.

For more information, see “Configure AUTOSAR XML Options”.

LONG-NAME configuration for calibration and measurement elements

Configuration of the `LONG-NAME` for data elements is often required in calibration workflows to generate correct ASAP2 files. R2022b introduces the ability to directly configure the `LONG-NAME` for the following measurement and calibration data:

- Parameters
- Datastores
- Signals & States
- Sender-Receiver Data Elements

To configure the `LONG-NAME` of parameters, datastores, signals, or states, you can use the Property or Mapping Inspectors. To programmatically configure the `LONG-NAME`, you can use the mapping API functions `mapParameter`, `mapDataStore`, `mapSignal`, and `mapState`.

For sender-receiver data elements, you can configure the `LONG-NAME` by using the Property and Mapping Inspectors or the AUTOSAR Dictionary. To configure programmatically, you can use the getAUTOSARProperties API functions.

For more information, see “AUTOSAR Calibration and Measurement Data”.

Fix-Axis Lookup Tables ARXML Support

Starting in R2022b, AUTOSAR Blockset supports importing and exporting of `ARXML` description of Fix Axis Lookup table application data types. The `SwRecordLayout` that specifies how to serialize data in the memory of an AUTOSAR ECU can be exported.

For more information, see “Configure FIX_AXIS Lookup Tables by Using Simulink Parameter Objects”.

RecordValueSpecification ARXML Support for Lookup Table Constants

Starting in R2022b, the AUTOSAR Blockset supports `RecordValueSpecification` (RVS) for specifying lookup table constants along with `ApplicationValueSpecification`. With this support, you can export or import the Lookup table constants as RVS.

For more information, see “Exporting Lookup Table Constants as Record Value Specification”.

Calibration File Customization

R2022b enhances `Generate Calibration Files` tool to include or exclude the AUTOSAR elements in the ASAP2 file for an AUTOSAR classic model.

For more information, see “Generate ASAP2 and CDF Calibration Files” (Simulink Coder).

Simulink.ValueType object to store design attributes with data types

AUTOSAR Blockset extends support for the Simulink `Simulink.ValueType` object that allows you to store design attributes such as minimum units, maximum units, and dimensions within a Simulink data type in a workspace object. AUTOSAR Blockset supports the use of `ValueType` objects for both import and export workflows.

On import, AUTOSAR blocks can read application data types and their corresponding implementation data types from ARXML and model application data types as `ValueType` objects. You can also use the `createComponentAsModel` functionality with models that use the `ValueType` objects. On export, ARXML files are generated to contain application data types and implementation data types to represent `ValueType` in the exported ARXML.

For more information, see `Simulink.ValueType` and “Component Creation”.

Internal data packaging support for multi-instance AUTOSAR models

R2022b supports configuring the default data packaging used for internal data stores, signals, and states associated with each instance of an AUTOSAR model to use C-typed per instance memory in the generated code.

You can configure C-typed per instance memory as the default data packaging when:

- The model is rate-based.
- The model is configured for multi-instance code generation; that is, the Configuration Parameter **Code interface packaging** is set to `Reusable function`.

For more information, see “Specify Default Data Packaging for AUTOSAR Internal Variables”.

End-to-End Transformer protection for sender and receiver ports

Starting in R2022b, Simulink supports AUTOSAR read and write calls that include a Transformer error argument in the generated code to enable end-to-end (E2E) data consistency checks.

You can configure RTE calls to use the optional E2E Transformer argument as the E2E protection method when using AUTOSAR schema version 4.2 or later. For earlier versions of AUTOSAR schema, you can configure the E2E protection method to be E2E protection wrapper, which is the Simulink default setting.

For more information, see “Configure AUTOSAR S-R Interface Port for End-To-End Protection”.

Change the parameter value of AUTOSAR elements at precompile or postbuild using variant parameters

R2022b enables you to generate AUTOSAR code for variant parameters in precompile and postbuild binding times. You can model precompile binding times for AUTOSAR elements by configuring variant parameters with a `code compile` activation time, a MATLAB® variable condition, and variant logic defined by expressions. Precompile variant points are resolved with a call to the runtime environment (RTE) prior to the code compile. Similarly, you can model postbuild binding times for AUTOSAR elements by configuring variant parameters with a `startup` activation time, a MATLAB variable condition, and variant logic defined by expressions. Postbuild variant points are resolved with a call to the RTE after the code is compiled and deployed on an electronic control unit (ECU). For more information, see “Configure Variant Parameter Values for AUTOSAR Elements”.

AUTOSAR Adaptive Software Component Modeling

AUTOSAR Adaptive Platform Release 20-11

R2022b extends support of the AUTOSAR Adaptive Platform to include Release 20-11. AUTOSAR Blockset supports schema version R20-11 for import and export of ARXML files and generation of AUTOSAR-compliant C++ code.

If you import schema R20-11 ARXML files into Simulink, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model.

If you create an AUTOSAR adaptive model in Simulink, the initial default schema version is R20-11. To change the schema version, use the model configuration parameter **Generate XML file for schema version** or an equivalent function call. For more information, see “Select AUTOSAR Adaptive Schema”.

Modeling and code generation for ara::com::method asynchronous call behaviors

R2022b extends support of Adaptive AUTOSAR modeling and code generation of client-server communication by supporting asynchronous, non-blocking call behaviors.

In R2022a, AUTOSAR Blockset introduced support for Adaptive AUTOSAR modeling and code generation of client-server communication with synchronous call behaviors, which produce blocked client execution, where clients send requests to a server and wait for the response. In R2022b, AUTOSAR Blockset extends that support to asynchronous call behaviors, where clients send requests, continue execution after the request is sent, and process the response upon method completion.

For more information, see “Model Client-Server Communication”.

Enhanced model function scheduling with multithreading for Adaptive AUTOSAR example main

Starting in R2022b, code generation for Adaptive AUTOSAR models introduces an `Executor` class to enable easier management for scheduling and execution of model tasks.

The new scheduling and execution interface is enabled for models that use the following Configuration Parameter settings:

- **Code Generation > System target file** set to `autosar_adaptive.tlc`
- **Code Generation > Language standard** set to `C++11 (ISO)`
- **Code Generation > AUTOSAR Code Generation Options > XCP Configuration Transport layer** set to `None`

For more information about Adaptive AUTOSAR configuration, see “Configure AUTOSAR Adaptive Software Components”.

Top-down namespace support of user-defined data types for ARXML-imported Adaptive AUTOSAR models

Starting in R2022b, Simulink preserves namespaces for user-defined data types that are described in the imported AUTOSAR XML (ARXML) and provides round trip support for ARXML export and code generation.

After ARXML import and algorithm development, you can generate AUTOSAR-compliant C++ code from the updated model, which produces namespaces for the data types that are preserved in the `model.cpp` and `model.h` generated files. Exporting ARXML from the updated model preserves the `<NAMESPACES>` tagging for the scoped data types.

For more information about importing ARXML files, see “Import AUTOSAR XML Descriptions Into Simulink”.

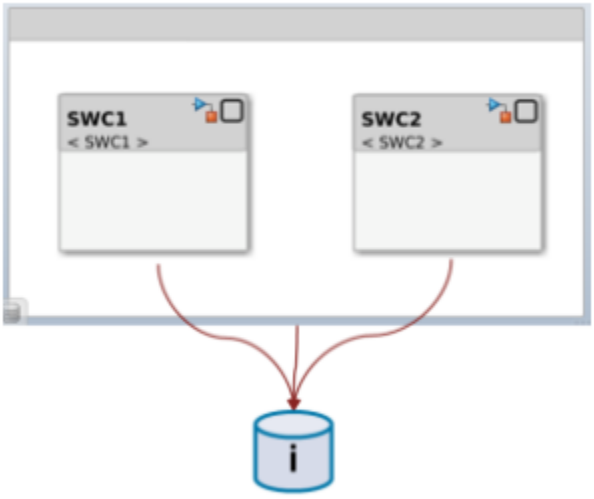
AUTOSAR Architecture Modeling

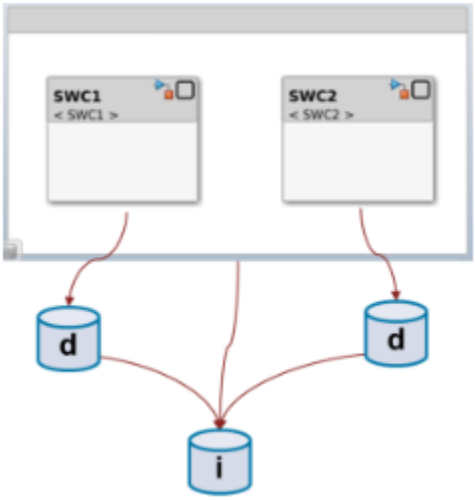
New interface dictionary to support data type and interface management for architecture model components and compositions

R2022b introduces a new dictionary, the interface dictionary, that is dedicated to managing the shared interfaces and data types across AUTOSAR components and compositions modeled in Simulink. The interface dictionary enables better data ownership and encapsulation within a component while still enabling sharing data types and interfaces across components. The interface dictionary can formally manage shared element definitions like interfaces and data types across components and compositions so that modeled design data and AUTOSAR properties are easier to scale in Simulink.

In R2022b, a top-down workflow is supported where you can create and link an interface dictionary to an AUTOSAR architecture model. Component models created from the AUTOSAR architecture models automatically have the interface dictionary linked to them. This design also allows regular data dictionaries to co-exist with interface dictionaries.

An interface dictionary can be linked directly to a model hierarchy or it can be indirectly referenced from a data dictionary:





You can graphically or programmatically configure the attributes of an interface dictionary and apply them to an architecture model using this basic workflow:

1. Create an interface dictionary.
2. Design data types and interfaces by using the interface dictionary.
3. Apply the interfaces to the architecture model in the Simulink environment.
4. Deploy the interface dictionary shared interface and data type content in the final application.

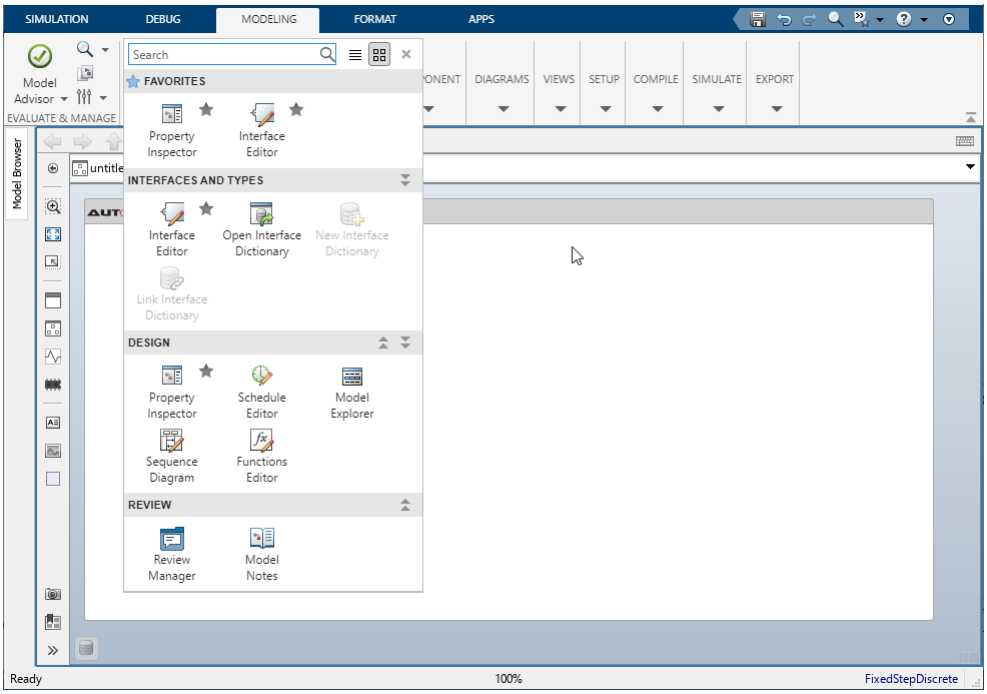
Additionally, ARXML export is supported for architecture models with interface dictionaries by exporting the dictionary ARXML into a folder with the interface dictionary name and adding the dictionary ARXML files to the created ZIP file for the application.

For more information, see “Manage Shared Interfaces and Data Types for AUTOSAR Architecture Models”.

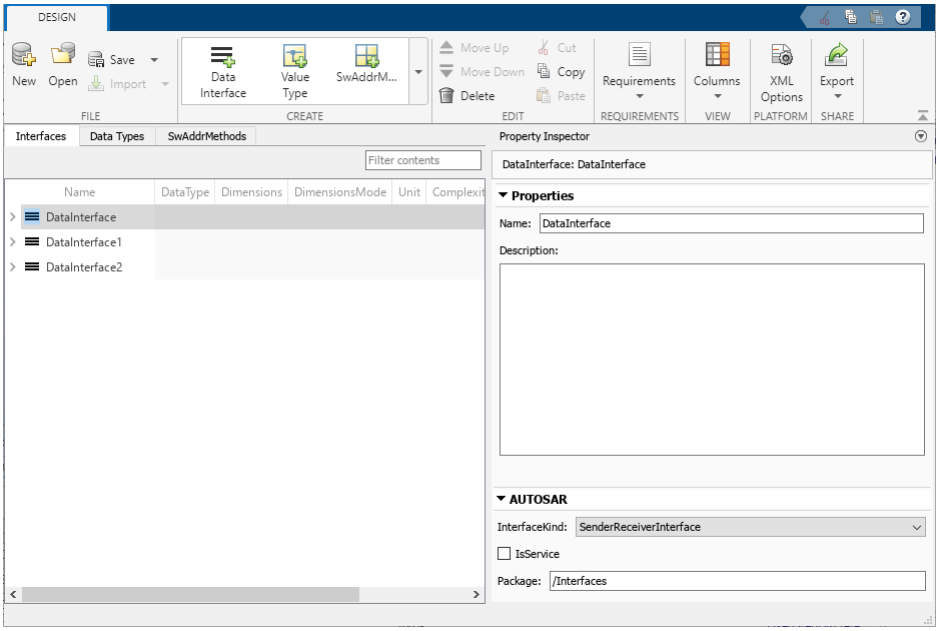
Graphical editor for interactive interface dictionary workflows

In 2022b, you can interactively configure shared interfaces, data types, and platform-specific attributes in an interface dictionary that you can then associate with an AUTOSAR model. You can graphically configure these attributes in the interface dictionary and then apply them to an architecture model using this basic workflow:

1. Create an interface dictionary. In an AUTOSAR architecture model, on the **Modeling** tab, open the **Design** menu and use the **Interfaces and Types** section to create or link to an existing interface dictionary to the model.

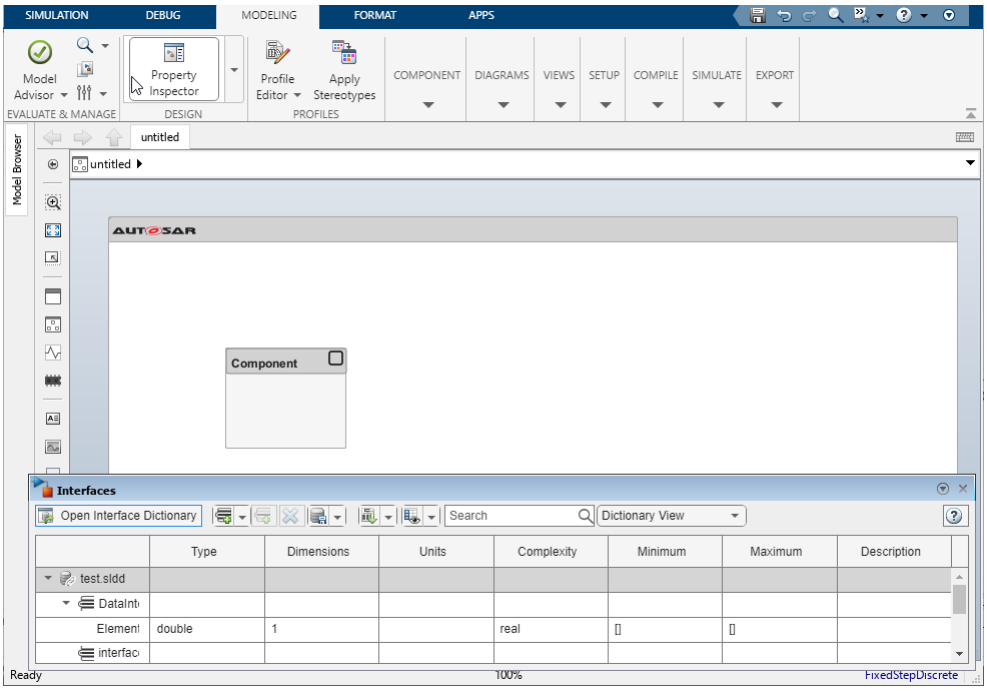


2. Design data types and interfaces by using the interface dictionary. In the dictionary, you can create and configure data types and interfaces.



In the Property Inspector pane, you can additionally view and configure the AUTOSAR interface properties **InterfaceKind**, **IsService**, and **Package**. Setting these properties in the Property Inspector will set them in the generated interface dictionary `.sldd` file.

3 Apply the interfaces to the architecture model. You can save the linked interface dictionary and use the Interface Editor to apply these properties to your modeled AUTOSAR application.



4 Deploy the interface dictionary content in the final application by building the model.

For more information, see “Manage Shared Interfaces and Data Types for AUTOSAR Architecture Models”.

Programmatic API for interface dictionary

R2022b introduces an API to support programmatic access to the interface dictionary. You can programmatically author, configure, and manage interfaces and data types in an interface dictionary by using the functions for the `Simulink.interface.Dictionary` object.

For more information, see “Manage Shared Interfaces and Data Types for AUTOSAR Architecture Models”.

Interface dictionary migrator

R2022b also introduces an interface dictionary migrator object that allows you to migrate data types and interfaces stored in the base workspace or in a data dictionary hierarchy to the interface dictionary associated with an architecture model. For more information, see `Migrator`.

Profiles and stereotypes for AUTOSAR architecture models

In R2022b, AUTOSAR expands on the current System Composer™ support of stereotypes and profiles to capture non-functional properties on architecture modeling elements. You can achieve this by creating a profile containing stereotype definitions and applying these stereotypes on the modeling elements. Stereotypes and profiles enable you to:

- Create custom views that focus on certain aspects of the architecture.
- Run analysis models by quantitatively evaluating the architecture for certain characteristics.

This custom meta-data is added on top of AUTOSAR elements and will not be imported or exported to ARXML but can be used in conjunction with System Composer for custom views and analysis. For more information, see “Create Profiles Stereotypes and Views for AUTOSAR Architecture Analysis” and “Use Stereotypes and Profiles” (System Composer).

Enable functions authoring in AUTOSAR Blockset architectures

In R2022b, functions authoring is enabled in AUTOSAR Blockset software architectures. The System Composer **Functions Editor** is now available from the toolstrip in the **Modeling** menu. The **Functions Editor** updates the diagram and populates it with the functions (runnables) from the reference components of the architecture model.

Compatibility Considerations

If an AUTOSAR architecture model created in a previous release is loaded in R2022b, and if it contains reference components, use the **Schedule Editor** to inspect and verify the partitions, which correspond to runnables. Otherwise, the partitions might not match ones from a previous release. Use the **Functions Editor** to specify the execution order of the functions.

Functionality being removed or changed

AUTOSAR Adaptive Linux Executable Toolchain

Behavior change

Beginning in R2022b, you can use the `AUTOSAR Adaptive Linux Executable` toolchain only if the Embedded Coder® Support Package for Linux® Applications is installed.

For more information, see “Build Out of the Box Linux Executable from AUTOSAR Adaptive Model”.

R2022a

AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2022a supports:

- Classic Platform Release 19-11
- PostBuild variants in ARXML
- Basic Software component event failure and recovery simulation and testing
- NvBlockNeeds parameters to read and write data at startup and shutdown
- Removal of unused data types and related elements in exported ARXML
- Improved ARXML for lookup table breakpoints
- Code replacement enhancements for blocks that do not overflow
- Generate Calibration Files tool for Classic AUTOSAR

AUTOSAR Classic Platform Release 19-11

R2022a extends support of the AUTOSAR Classic Platform to include Release 19-11. AUTOSAR Blockset supports schema version R19-11 for import and export of ARXML files and generation of AUTOSAR-compliant C code.

If you import schema R19-11 ARXML files into Simulink, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model.

If you create an AUTOSAR classic model in Simulink, the initial default schema version is 4.3. To set the schema version to R19-11, use the model configuration parameter **Generate XML file for schema version** or an equivalent function call. For more information, see Select AUTOSAR Classic Schema.

Postbuild Conditions for Startup Variants

R2022a allows you to import, export, and model postbuild conditions for AUTOSAR startup variants modeled in Simulink. You can model postbuild binding times for AUTOSAR components by configuring variant blocks with a `Startup` activation time, a MATLAB variable condition, and variant logic defined by expressions. Postbuild variant points are resolved with a call to the run time environment (RTE) after the code has been compiled and deployed on an electronic control unit (ECU). Optionally, you can elect generate the ARXML for these postbuild conditions as a package. For more information, see Configure Postbuild Variant Conditions for AUTOSAR Software Components.

Basic Software component event failure testing and simulation with Dem Status Override and Dem Status Inject blocks

R2022a introduces two new blocks, Dem Status Override and Dem Status Inject, to enable the simulation and testing of a Basic Software Component event failure and recovery. The Dem Status Override block enables you to override and set the status of an event in a modeled basic software component so that you can simulate and test specific behavior to quickly gain coverage. The Dem Status Inject block block enables you to simulate an instantaneous external event so that you can test the robustness of a basic software component and whether it can recover, or heal, itself from a failure.

For more information, about the blocks, see Dem Status Override and Dem Status Inject. For more information about how to use the blocks for simulation and testing, see Simulate and Verify AUTOSAR Component Behavior by Using Diagnostic Fault Injection.

Configure NvBlockNeeds parameters StoreAtShutdown and RestoreAtStart

R2022a introduces the ability to configure NVRAM block data to be read out at startup and written away at shutdown by setting the NVBlockNeeds parameters RestoreAtStart and StoreAtShutdown. Additionally, these parameters are included in the exported ARXML. For more information, see Configure AUTOSAR Per-Instance Memory.

Export ARXML without unused data types and related elements

Starting in R2022a, data types and related elements that are not used in the model are removed from the exported ARXML files. Removal of these unused data types and elements help prevent data type conflicts with other ARXML files when imported to third-party authoring tools and makes it easier to compare ARXML files in incremental workflows. For more information, see Generate AUTOSAR C Code and XML Descriptions.

Improved breakpoint ARXML and mapping support for model hierarchies

R2022a enhances `Simulink.Breakpoint` objects as model arguments to bring these objects on par with `Simulink.Parameter` and `Simulink.LookupTable` objects. You can now:

- Use `Simulink.Breakpoint` objects in model hierarchies to generate AUTOSAR COM_AXIS Lookup tables in ARXML
- Import, export, and map `Simulink.Breakpoint` objects in the model workspace as port or per-instance parameters
- Import, export, and map `Simulink.Breakpoint` objects in sub-component models as per-instance parameters

Test harness support is also provided for this new breakpoint functionality. For more information, see Configure Lookup Tables for AUTOSAR Calibration and Measurement.

Code replacement enhancements for blocks that do not overflow

In R2022a, the AUTOSAR 4.0 code replacement library replaces code from more blocks for these categories of entries:

- Multiplication
- Division
- Combination of multiplication and division
- Combination of multiplication and shift right

Compatibility Considerations

Previously, these entries in the AUTOSAR 4.0 library replaced code from the Product block only when **Saturate on integer overflow** was selected for the block. In R2022a, these entries also replace code from Product blocks that do not overflow, regardless of whether **Saturate on integer overflow** is selected.

Generate Calibration Files tool for Classic AUTOSAR

R2022a extends support of `Generate Calibration Files` tool for Classic AUTOSAR platform. The `Generate Calibration Files` supports generation and customization of ASAP2 file and generation of CDFX file.

For more information, see Generate ASAP2 and CDF Calibration Files (Embedded Coder).

AUTOSAR adaptive software component modeling

To improve AUTOSAR adaptive component modeling, R2022a supports:

- `ara::com::method` support
- Event-Based execution modeling and code generation
- Scoped enum classes
- Nested namespace customization in generated C++ code
- DDS bindings for Adaptive AUTOSAR applications

ara::com::method support

R2022a adds `ara::com` method support. For more information, see Model AUTOSAR Adaptive Service Communication.

Event-based execution modeling and code generation for ara::com events

R2022a enables you to model and generate code for event-triggered execution in AUTOSAR adaptive applications. You can now model software components that execute only when an event arrives. Previously, you could only model message or event communication by periodically polling for an event, which introduced the overhead of executing a runnable even when an event was not present. Event-triggered execution eliminates this overhead by executing a runnable only when an event arrives. You can also generate C++ code from these applications that is compliant with the AUTOSAR adaptive schema 00046-00048. For more information, see Model AUTOSAR Adaptive Service Communication.

Scoped enum classes for C++11 code generation

By default, R2022a uses C++11 style scoped enum classes in generated AUTOSAR adaptive code. Enum classes help facilitate easier integration by minimizing the need for adjustments to the generated code. For more information, see C++11 Style Scoped Enum Classes Generated for AUTOSAR Adaptive Applications.

Nested namespace customization for C++ code generation

You can now customize the generated C++ code from your modeled AUTOSAR adaptive application to generate the model class in a nested namespace. For more information, see Configure AUTOSAR Adaptive Code Generation.

Enable ara::com DDS binding for Adaptive AUTOSAR applications

Starting from R2022a, the generated code for adaptive AUTOSAR model supports DDS binding for `ara::com` enabling communication between adaptive AUTOSAR applications. This feature is supported out of the box with `AUTOSAR Adaptive Linux Executable` toolchain. The generated `ServiceInstanceManifest.arxml` contains DDS deployment artifacts. The applications communicating with each other are expected to have event deployment artifacts with same `TOPIC-NAME` and `DOMAIN-ID`.

For more information, see Build Out of the Box Linux Executable from AUTOSAR Adaptive Model.

R2021b

AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2021b adds:

- AUTOSAR interpolation and math routine library implementations for host code verification
- Improved lookup table ARXML support for model hierarchies
- Lookup table ARXML support for AdminData
- Improved performance for imported compositions by sharing AUTOSAR dictionary

AUTOSAR IFX, IFL, MFX, and MFL library implementations for host code verification

R2021b enhances MATLAB host code verification for AUTOSAR models by providing host implementations of IFX, IFL, MFX, and MFL routines in the AUTOSAR 4.0 library. The host library implementations enable software-in-the-loop (SIL) validation for models that trigger code replacements from the AUTOSAR 4.0 library. For more information, see AUTOSAR 4.0 Library Host Code Verification.

Improved lookup table ARXML support for model hierarchies

R2021b enhances ARXML export for lookup tables in model hierarchies. When a referenced model contains a lookup table, and the containing top model passes lookup table parameter values to the model arguments of the referenced model, top-model export now generates application data types for the lookup table parameters.

Parameterizing instances of reusable referenced model lookup tables allows you to place multiple instances of lookup table sub-units in an AUTOSAR model hierarchy and supports sub-unit level testing. For more information, see Parameterizing Instances of Reusable Referenced Model Lookup Tables.

Lookup table ARXML support for AdminData record layout annotations

R2021b enhances import and export of ARXML lookup table descriptions to support `AdminData` record layout annotations. `AdminData` record layout annotations can be used with third-party AUTOSAR tools. For more information, see Exporting `AdminData` Record Layout Annotations.

Improved performance for imported compositions by sharing AUTOSAR dictionary

To improve the performance of common tasks in AUTOSAR composition modeling, composition import can now store sharable component properties, such as interfaces and data types, into a Simulink data dictionary. Components within the composition can then share the stored properties.

For compositions that contain more than 20 software components, sharing AUTOSAR properties among components can significantly improve performance for composition workflows, including import, dictionary navigation, AUTOSAR validation, and code generation. Limiting property replication among components can reduce component model file sizes.

To configure a composition import to store AUTOSAR properties for component sharing, call the ARXML import function `createCompositionAsModel` and specify the `'DataDictionary'` and `'ShareAUTOSARProperties'` arguments. For more information, see Import AUTOSAR Composition and Share AUTOSAR Dictionary and the `createCompositionAsModel` reference page.

AUTOSAR adaptive software component modeling

To improve AUTOSAR adaptive component modeling, R2021b supports:

- Persistent memory (`ara::per`) for adaptive applications
- C++ reference types in the generated code
- Name and namespace customization for C++ model classes

Persistent memory for adaptive applications

Starting in R2021b, adaptive applications support code generation for persistent memory (`ara::per`). Persistent memory provides a mechanism to store information in the nonvolatile memory of an ECU. The information is available over boot and ignition cycles.

For more information, see Model AUTOSAR Adaptive Persistent Memory.

C++ reference types in the generated code

To better adhere to the AUTOSAR adaptive standard, C++ code generation now supports reference data types. Reference data types help the generated code integrate with AUTOSAR adaptive function calls, increase the readability of the generated C++ code, improve the compiler optimization, and enable the generated variables to be more tightly scoped. For more information, see Code Generation.

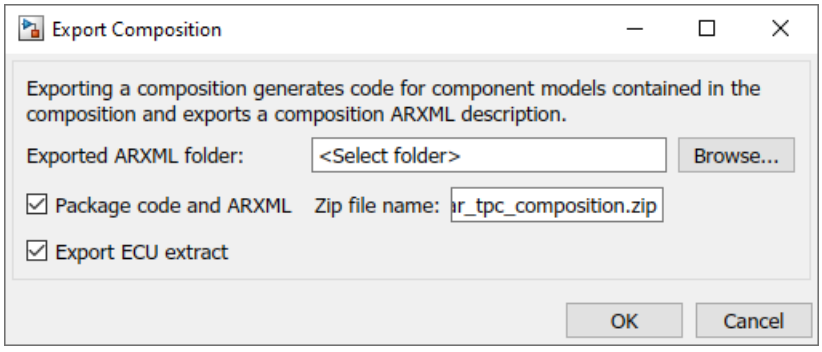
Name and namespace customization for C++ model classes

You can now control the generated C++ class names and namespaces for your AUTOSAR adaptive applications. For more information about how to configure these aspects of the generated code, see Configure AUTOSAR Adaptive Code Generation.

Export software component mapping for AUTOSAR ECU

R2021b adds the ability to export ECU extracts from compositions in an AUTOSAR architecture model. ECU extracts are an important input to AUTOSAR ECU configuration. In an AUTOSAR architecture, a top-level composition can model the software components mapped to one AUTOSAR ECU. To create a software description of the ECU-scoped system, you export an ECU extract from the composition.

In an open architecture model, you can export ARXML by using the Simulink Toolstrip, the software architecture canvas, or the `export` function. For example, from the **Modeling** tab, select **Export** > **Generate Code and ARXML**. In the Export Composition dialog box, select the option **Export ECU extract**.



To generate the ECU extract, the software automatically maps the software components in the composition to an ECU. If the composition contains nested compositions, the software uses a flattened version of the composition hierarchy, containing only components. Composition export generates the ECU extract into the file `System.arxml`, which is located in the composition folder.

For more information, see Export Composition ECU Extract and the `export` reference page.

AUTOSAR mapping workflow enhancements

Starting in R2021b, you can now configure additional code mapping properties from within the Code Mappings editor. These properties were previously accessible only in the Property Inspector.

To configure the properties, click the  icon in the row containing the element you want to configure.

The Code Mappings editor now also displays data related to In Bus Element and Out Bus Element blocks in a hierarchical view. In previous releases, this data displayed as a flat list in the Code Mappings editor.

Specify message queue properties on AUTOSAR component-level bus element ports

In R2021b, Simulink root-level In Bus Element blocks can provide receive message queue properties. If your AUTOSAR component model uses message communication, you can use an In Bus Element block to specify message queue attributes such as capacity, message sorting policy, and message overwriting policy.

For AUTOSAR architecture models, the software automatically configures In Bus Element blocks with message queue properties when you:

- Import an ARXML software component that uses message communication.
- Link to a classic implementation model that uses message communication.

Functionality being removed or changed

AUTOSAR adaptive default name for a model class is the model name

Beginning in R2021b, the default name for AUTOSAR adaptive model classes are the names of the models. Previously, the class names in the generated code used a default class name of the form *`model`*`ModelClass`. The new default is of the form *`model`*.

This change to the generated class names may cause integration scripts that use the class names to break. Update integration scripts to the new generated class names. For more information, see Code Generation.

R2021a

Version: 2.4

New Features

Bug Fixes

AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2021a adds:

- Simulink messaging over bus ports for queued S-R communication
- Export-function runnable modeling with Simulink bus ports
- Variant conditions and nonvirtual buses with Simulink bus ports
- API control of default internal data packaging
- Enhanced ARXML support for lookup tables

Simulink messaging over bus ports for queued S-R communication

AUTOSAR component models implement queued sender-receiver (S-R) communication by sending and receiving Simulink messages over root-level ports. In R2021a, you can use Simulink bus ports to model AUTOSAR message-based communication.

Bus ports enable intuitive modeling of AUTOSAR communication ports, interfaces, and groups of data elements. If you model AUTOSAR ports by using In Bus Element and Out Bus Element blocks, and type the bus ports by using bus objects, basic properties of AUTOSAR ports, interfaces, and data elements are configured without using the AUTOSAR Dictionary. To manage component interfaces, you configure Simulink bus objects.

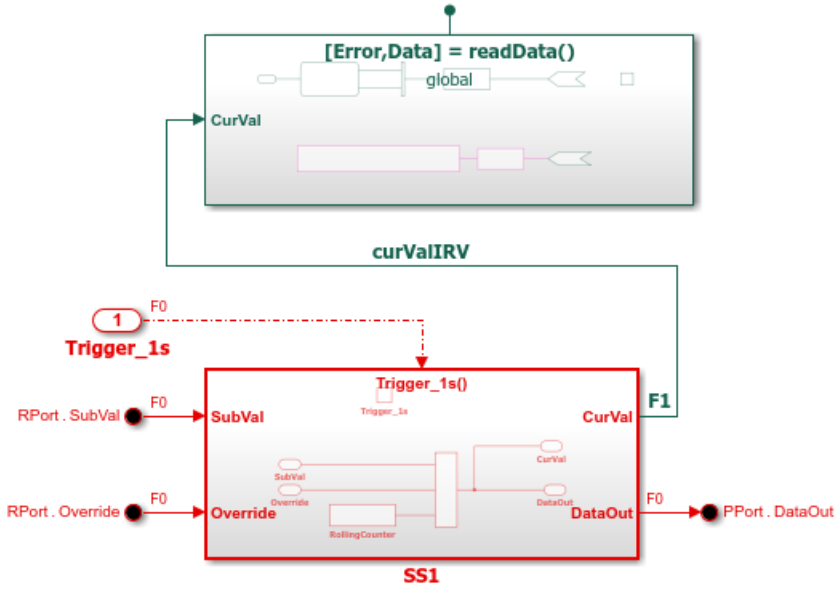
AUTOSAR architecture models can link queued S-R component models that have bus ports.

For more information, see Configure AUTOSAR Ports By Using Simulink Bus Ports, Messages, and Configure AUTOSAR Queued Sender-Receiver Communication.

Components with export-function runnables support Simulink bus ports

Simulink modeling styles for AUTOSAR runnables include export-function modeling. In R2021a, AUTOSAR components that model runnables by using exported functions can use Simulink bus ports.

For example, this adaptation of example model `autosar_swc_slfcns` uses bus ports. Function-call subsystem SS1 models a periodic runnable and generates an exported function.



In AUTOSAR architecture models, you can link component models that use the export-function modeling style and then use the Schedule Editor to schedule the simulation.

For more information, see Configure AUTOSAR Ports By Using Simulink Bus Ports and Model AUTOSAR Software Components.

Components with Simulink bus ports support variant conditions and nonvirtual buses

R2021a improves AUTOSAR port modeling by supporting variant conditions and nonvirtual buses with Simulink bus ports. In AUTOSAR software component models that model AUTOSAR ports by using bus ports, you can:

- Use variant conditions on component ports.
- Use nonvirtual buses on component interface data elements.

AUTOSAR architecture models can link the resulting component implementations.

For more information, see Configure AUTOSAR Ports By Using Simulink Bus Ports.

Default data packaging for AUTOSAR internal variables

R2021a provides functions to control the default data packaging used for internal variables in the generated code for an AUTOSAR component model. You can specify that internal data store, signal, and state data is packaged:

- With or without packing it in a structure
- With private or public visibility

For more information, see Specify Default Data Packaging for AUTOSAR Internal Variables and the function ref pages `getInternalDataPackaging` and `setInternalDataPackaging`.

Lookup table ARXML support for row-major layout and improved tool interoperability

R2021a enhances ARXML export and import of lookup tables to support row-major array layout and improved interoperability with third-party AUTOSAR authoring tools.

To specify row-major order for lookup table values in the linear memory of an ECU, ARXML descriptions of multidimensional lookup tables must specify a `SwRecordLayout` with the CATEGORY set to `ROW_DIR`. In R2021a:

- Exporting Simulink row-major multidimensional lookup tables generates ARXML lookup table descriptions with the `SwRecordLayout` category set to `ROW_DIR`.
- Importing ARXML files with only row-major multidimensional lookup table descriptions creates Simulink lookup tables with **Array layout** set to `Row-major` and **Use algorithms optimized for row-major array layout** enabled.

Third-party AUTOSAR authoring tools expect exported lookup table descriptions to contain `ApplicationDataTypes` with VALUE-AXIS-DATA-TYPE-REF and INPUT-VARIABLE-TYPE-REF elements. VALUE-AXIS-DATA-TYPE-REF and INPUT-VARIABLE-TYPE-REF are expected to reference table value and breakpoint data types specified in Simulink lookup table blocks. In R2021a:

- Exporting Simulink lookup tables generates ARXML lookup table descriptions containing `ApplicationDataTypes` that have VALUE-AXIS-DATA-TYPE-REF and INPUT-VARIABLE-TYPE-REF elements.

- Importing ARXML lookup table descriptions creates Simulink lookup tables that have table value and breakpoint data types configured based on imported `ApplicationDataType` settings.

For more information on modeling AUTOSAR lookup tables, see Model AUTOSAR Calibration Parameters and Lookup Tables and Configure Lookup Tables for AUTOSAR Measurement and Calibration.

AUTOSAR adaptive software component modeling

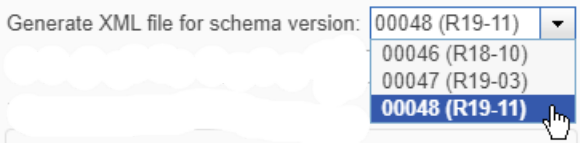
To improve AUTOSAR adaptive component modeling, R2021a adds:

- Adaptive Platform Release 19-11 support
- Simulink messaging over bus ports for event-based communication
- Enhanced CMake tooling for adaptive models
- Run-time logging (`ara::log`) for adaptive executables

AUTOSAR Adaptive Platform Release 19-11

R2021a extends support of the AUTOSAR Adaptive Platform to include Release 19-11. AUTOSAR Blockset supports the 000048 (R19-11) schema for import and export of ARXML files and generation of AUTOSAR-compliant C++ code.

- In R2021a, 000048 (R19-11) is the default schema version for AUTOSAR adaptive models created in Simulink.



- If you import schema 000048 (R19-11) ARXML files into Simulink, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model.
- Building an AUTOSAR adaptive model using schema 000048 (R19-11) generates ARXML descriptions and C++ code that comply with Adaptive Platform Release 19-11.

For more information on AUTOSAR schema versions, see Select AUTOSAR Schema.

Simulink messaging over bus ports for event-based communication

AUTOSAR adaptive models implement event-based communication by sending and receiving Simulink messages over root-level ports. In R2021a, you can use Simulink bus ports to model AUTOSAR message-based communication.

Bus ports enable intuitive modeling of AUTOSAR communication ports, interfaces, and groups of data elements. If you model AUTOSAR ports by using In Bus Element and Out Bus Element blocks, and type the bus ports by using bus objects, basic properties of AUTOSAR ports, interfaces, and events are configured without using the AUTOSAR Dictionary. To manage component interfaces, you configure Simulink bus objects.

For more information, see Configure AUTOSAR Ports By Using Simulink Bus Ports, Messages, and Model AUTOSAR Adaptive Service Communication.

Enhanced CMake tooling for adaptive models

R2021a improves the `CMakeLists.txt` generation framework for AUTOSAR adaptive models. You can use modern CMake tooling to build Linux standalone executables, static libraries, and shared libraries.

For more information, see Build Library or Executable from AUTOSAR Adaptive Model and Build Out of the Box Linux Executable from AUTOSAR Adaptive Model.

Run-time logging (ara::log) for adaptive executables

Starting in R2021a, adaptive applications can forward event logging information to a console, file, or network, as defined in the AUTOSAR `Specification of Diagnostic Log and Trace`. You can collate and analyze log data from multiple applications.

For more information, see Configure Run-Time Logging for AUTOSAR Adaptive Executables.

AUTOSAR architecture modeling

To improve AUTOSAR architecture modeling, R2021a adds:

- ARXML support for execution order constraints at architecture (VFB) level
- Model Linker app for meeting component model linking requirements

ARXML support for execution order constraints at architecture (VFB) level

In an AUTOSAR architecture model, you can use the Schedule Editor to schedule and specify the execution order of AUTOSAR component runnables. R2021a adds support for ARXML descriptions of execution order constraints at Virtual Function Bus (VFB) level. You can:

- Import VFB-level execution order constraints from ARXML files.
- Open an AUTOSAR architecture model and use the Schedule Editor to modify the execution order of component runnables. The editor displays every component runnable in the composition hierarchy.
- As part of composition export, export VFB-level execution order constraints to an ARXML timing module, *`modelname`*`_timing.arxml`.

For more information, see Schedule Component Runnables.

Model Linker app for meeting component model linking requirements

R2021a provides an AUTOSAR Model Linker app, which opens as needed to help you link AUTOSAR component blocks to existing Simulink implementation models.

In an architecture model, when you initiate linking of an AUTOSAR Software Component block to an implementation model, the software verifies whether the specified model meets linking requirements. For example, the implementation model must use the same target as the architecture model, use a fixed-step solver, and use root-level bus ports.

If the implementation model does not meet one or more of the linking requirements, the software opens the AUTOSAR Model Linker app, which offers fixes for the unmet requirements. When you click **Fix All**, the software fixes the unmet requirements and finishes linking the component block to the model.

For more information, see Link to Implementation Model.

R2020b

AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2020b adds:

- Classic Platform Release 4.4 support
- Enhanced port parameter modeling
- IncludedDataTypeSets for internal data types
- ARXML descriptions for execution order constraints
- Enhanced signal data mapping

Support for AUTOSAR Classic Platform Release 4.4

R2020b extends support of the AUTOSAR Classic Platform to include Release 4.4. AUTOSAR Blockset supports schema version 4.4 for import and export of ARXML files and generation of AUTOSAR-compliant C code.

If you import schema version 4.4 ARXML files into Simulink, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model. For more information on schema import and export, see Select AUTOSAR Schema.

Port parameter modeling enhancements

R2020b enhances the workflow for modeling AUTOSAR port parameters for port-based parameter communication. In Simulink, you can model the receiver side of AUTOSAR parameter communication.

To enhance port parameter modeling, R2020b extends model workspace parameter mapping to support AUTOSAR port-based parameters. By using the **Parameters** tab of the Code Mappings editor or the `mapParameter` function, you can map Simulink model workspace parameters to AUTOSAR receiver port parameters. Mappable parameters include Simulink parameter, lookup table, and breakpoint objects.

Because port parameter data is scoped to the model workspace and the AUTOSAR component:

- Different components can use the same parameter names without naming conflicts.
- An AUTOSAR composition can contain multiple instances of a parameter receiver component, each with instance-specific port parameter data values.

For more information, see Port Parameters and Configure AUTOSAR Port Parameters for Communication with Parameter Component.

Import and export AUTOSAR IncludedDataTypeSets

In R2020b, you can import and export ARXML descriptions of AUTOSAR included data type sets (`IncludedDataTypeSets`). An `IncludedDataTypeSet` defines AUTOSAR data types that are internal to a component and not present in the component interface descriptions. Multiple components can import an `IncludedDataTypeSet` to share a common set of internal data types.

In an AUTOSAR software component model, you can configure internal data types to be exported in ARXML `IncludedDataTypeSets` and generated in C header files. For more information, see Included Data Type Sets and Configure Internal Data Types for AUTOSAR `IncludedDataTypeSets`.

ARXML support for execution order constraints

In an AUTOSAR model, you can use the Schedule Editor to schedule and specify the execution order of AUTOSAR component runnables. R2020b adds support for ARXML descriptions of execution order constraints. You can:

- Import execution order constraints from ARXML files.
- Open an AUTOSAR component model and use the Schedule Editor to modify the execution order of runnables.
- Export execution order constraints to ARXML files.
- Update execution order constraints in an AUTOSAR component model by importing ARXML changes.

For more information, see Configure AUTOSAR Runnable Execution Order.

Signal data mapping enhancements

R2020b enhances the workflow for mapping Simulink signals to AUTOSAR variables for run-time calibration. You can now selectively add or remove model signals from AUTOSAR component signal mapping.

In the AUTOSAR Component Designer app, use a Code Mappings editor button or a model cue:

- In the model canvas, select one or more signals. Open the Code Mappings editor, **Signals/States** tab, and click an add or remove button.
- In the model canvas, select a signal. Place your cursor over the displayed ellipsis and select an add or remove model cue.

Alternatively, from a MATLAB script or command line, call function `addSignal` or `removeSignal`.

Previously, for AUTOSAR components that were created in Simulink (bottom-up workflow), the initial default signal mapping included all named signals in the model. In R2020b, named signals are no longer mapped by default. After creating a mapped AUTOSAR component in Simulink, you selectively add and map signals.

For AUTOSAR components imported or updated from ARXML (round-trip workflow), the signal mapping workflow is unchanged. After import, you add content to the component model to implement functionality. Updating the model mapping to reflect Simulink changes automatically creates and maps signals to represent imported AUTOSAR variables.

For more information, see Map Block Signals and States to AUTOSAR Variables.

AUTOSAR adaptive software component modeling

To improve AUTOSAR adaptive component modeling, R2020b adds:

- Linux executables for adaptive models
- ASAP2 file generation enhancements

Linux executables for adaptive models

In R2020b, you can create a Linux executable for AUTOSAR adaptive models. By selecting the `AUTOSAR Adaptive Linux Executable` toolchain for an AUTOSAR adaptive component model, you can create an AUTOSAR executable and run it as a standalone application.

For more information, see Create Linux Executable for Run-Time Calibration.

ASAP2 file generation enhancements

In R2020b, you can configure ASAP2 (A2L) file generation to specify an ASAP2 file version and whether to exclude or include comments.

- Previously, you could generate only version 1.31 ASAP2 files. Now, in the ASAP2 Generator app, you can select a supported ASAP2 file version. The default version is 1.71.
- Previously, the generated ASAP2 file included comments. Now you can choose to exclude comments.

For more information, see Configure AUTOSAR Adaptive Data for Run-Time Measurement and Calibration.

Import AUTOSAR software composition into architecture model

To improve AUTOSAR software architecture modeling, R2020b allows you to import ARXML descriptions of AUTOSAR software compositions into architecture models. Previously, you could only import ARXML composition descriptions outside architecture models, by using the ARXML importer function `createCompositionAsModel`.

In an open AUTOSAR architecture model with no functional content, you can import an AUTOSAR software composition by using either an app or a function:

- On the **Modeling** tab, select **Import from ARXML**. The AUTOSAR Importer app opens. Work through the procedure to import the ARXML composition description and create a composition editor representation of the composition in the software architecture canvas.
- Call the architecture function `importFromARXML`.

You can specify composition import options, including:

- Whether to include or exclude AUTOSAR software components, which define composition behavior.
- Simulink data dictionary in which to place data objects for imported AUTOSAR data types.
- Names of existing Simulink behavior models to link to imported AUTOSAR software components.
- Component options to apply when creating Simulink behavior models for imported AUTOSAR software components.

For more information, see Import AUTOSAR Composition from ARXML, `importFromARXML`, and the example Import AUTOSAR Composition into Architecture Model.

Functionality being removed or changed

Support for AUTOSAR Classic Platform schema versions 3.x and 2.1 has been removed

Errors

R2020b removes support for AUTOSAR Classic Platform schema versions 3.x and 2.1. Validating an AUTOSAR model that uses schema version 3.x or 2.1 generates an error, which prevents code generation.

Use schema version 4.0 or later instead. The default schema version for new AUTOSAR models is 4.3. For more information, see Select AUTOSAR Schema.

R2020a

AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2020a adds:

- Basic Software blocks for modeling function inhibition
- Export of multidimensional matrices for AUTOSAR variables

Model function inhibition by using Basic Software blocks

For the AUTOSAR Classic Platform, AUTOSAR Blockset provides Basic Software (BSW) blocks, which allow you to model software component calls to BSW services that run in the AUTOSAR run-time environment. BSW services include NVRAM Manager and Diagnostic Event Manager.

R2020a extends BSW service support to include component calls to Function Inhibition Manager (FiM) services. As defined in the AUTOSAR specification, the Function Inhibition Manager provides a control mechanism for selectively inhibiting (that is, deactivating) function execution in software component runnables, based on function identifiers (FIDs) with inhibition conditions. For example, an FID can represent functionality that must be stopped if a specific failure occurs.

The Function Inhibition Manager is closely related to the Diagnostic Event Manager (Dem), because inhibition conditions can be based on the status of diagnostic events. For example, if a sensor failure event is reported to the Diagnostic Event Manager, the Function Inhibition Manager can inhibit the associated function identifier and stop execution of the corresponding functionality.

R2020a adds FiM and Dem blocks that allow you to:

- Query the status of function inhibition conditions.
- Configure function inhibition criteria based on diagnostic event status.
- Define operation cycles to scope failures to a time period.

For more information, see Configure Calls to AUTOSAR Function Inhibition Manager Service.

Export multidimensional matrices for AUTOSAR variables

For an AUTOSAR component model with multidimensional arrays, if you set the model configuration parameter **Array layout** to `Row-major`, you can preserve dimensions of multidimensional arrays in the generated C code. Preserving array dimensions in the generated code can enhance code integration.

R2020a improves ARXML export to honor the row-major setting for multidimensional signals and states mapped to AUTOSAR variables. If **Array layout** is set to `Row-major`, ARXML export no longer flattens multidimensional matrices in the generated descriptions for signals and states mapped to AUTOSAR variables.

In the generated C code, access to the variable data depends on the type of storage. For internal storage, such as static memory, the code uses multidimensional indexing. For external storage, the code calls AUTOSAR `Rte` functions, which use one-dimensional indexing.

When you use `Row-major` array layout, you can disregard the setting of the AUTOSAR model configuration parameter **Support root-level matrix I/O using one-dimensional arrays**. **Support root-level matrix I/O using one-dimensional arrays** provides a workaround for exporting multidimensional matrices in `Column-major` array layout.

AUTOSAR adaptive software component modeling

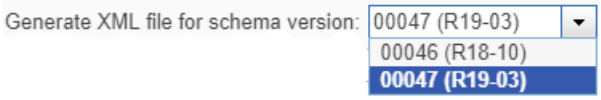
To improve AUTOSAR adaptive component modeling, R2020a adds:

- Adaptive Platform Release 19-03 support
- Data calibration based on XCP communication and ASAP2 file generation
- Dynamic service discovery

Support for AUTOSAR Adaptive Platform Release 19-03

R2020a extends support of the AUTOSAR Adaptive Platform to include Release 19-03. AUTOSAR Blockset supports the 000047 (R19-03) schema for import and export of ARXML files and generation of AUTOSAR-compliant C++ code.

- In R2020a, 000047 (R19-03) is the default schema version for AUTOSAR adaptive models created in Simulink.



- If you import schema 000047 (R19-03) ARXML files into Simulink, the ARXML importer detects and uses the schema version and sets the schema version parameter in the model.
- Building an AUTOSAR adaptive model using schema 000047 (R19-03) generates ARXML descriptions and C++ code that comply with Adaptive Platform Release 19-03.

For more information on AUTOSAR schema versions, see Select AUTOSAR Schema.

Calibrate data for adaptive applications by using XCP and ASAP2

R2020a allows you to configure run-time calibration of adaptive application data based on XCP Slave communication and ASAP2 (A2L) file generation. The XCP and ASAP2 capabilities are defined outside the Adaptive Platform (AP) specifications, which as of Release 19-03 do not address data calibration.

As part of generating and deploying adaptive code, you can configure interfaces for XCP Slave communication in the generated C++ code and export A2L files containing model data for measurement and calibration.

Before deploying adaptive code, you can:

- Use the Configuration Parameters dialog box to configure the model, to generate XCP Slave function calls in adaptive C++ code.
- Use the ASAP2 Generator app to configure and generate an ASAP2 (A2L) file that describes model data for measurement and calibration.

For more information, see Configure AUTOSAR Adaptive Data for Run-Time Measurement and Calibration.

Find adaptive services by using dynamic discovery

R2020a allows you to configure applications to use dynamic discovery to subscribe to adaptive services as they become available. Previously, applications found and subscribed to adaptive services one time during initialization. One-time discovery may require adaptive services to start before applications and prevent applications from using new services as they become available. Now you can

configure, in your model or programmatically, the service discovery mode of each required service port as `OneTime` or `DynamicDiscovery`.

For more information, see Configure AUTOSAR Adaptive Service Discovery Modes.

AUTOSAR software architecture modeling

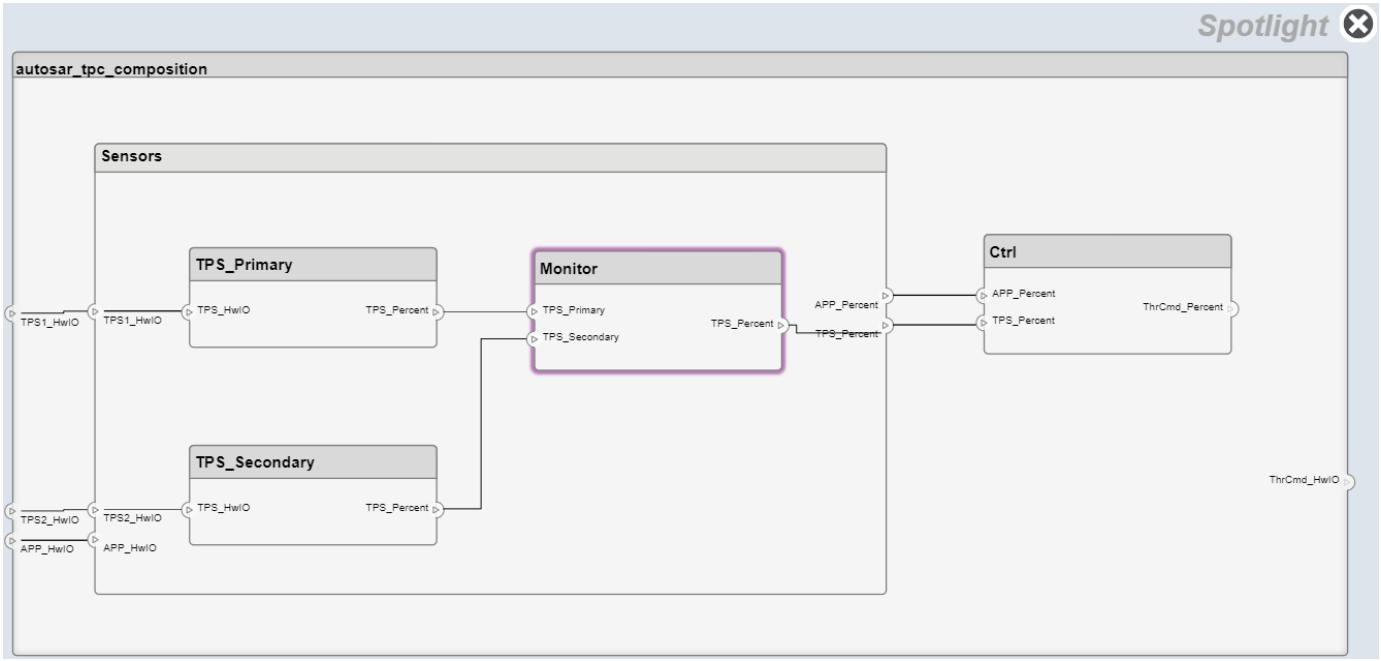
To improve AUTOSAR software architecture modeling, R2020a adds:

- Spotlight views for analyzing dependencies
- Programmatic interface for authoring architecture models

Use spotlight view to analyze component or composition dependencies

In an AUTOSAR architecture model, to help analyze component or composition dependencies, you can create a spotlight view. A spotlight view is a simplified view of an architecture component or composition that captures its upstream and downstream dependencies.

Here is a spotlight view of component `Monitor` in AUTOSAR example model `autosar_tpc_composition`.



For more information, see View AUTOSAR Component or Composition Dependencies.

Programmatically create and configure architecture models

R2020a adds a programmatic interface for developing AUTOSAR architecture models. Use the AUTOSAR architecture functions to:

- Create, load, open, save, or close an AUTOSAR architecture model.
- Add, connect, or remove AUTOSAR components, composition, and ports.

- Find AUTOSAR elements and modify properties.
- Define component behavior by creating or linking Simulink models.
- Add Basic Software (BSW) service component blocks for simulating BSW service calls.
- Export composition and component ARXML descriptions and generate component code (requires Embedded Coder).

For more information, see Software Architecture Modeling and Configure AUTOSAR Architecture Model Programmatically.

Functionality being removed or changed

Support for AUTOSAR Classic Platform schemas 3.x and 2.1 will be removed
Still runs

Support for AUTOSAR Classic Platform schemas 3.x and 2.1 will be removed in a future release. Use schema 4.0 or later instead. The default schema for new AUTOSAR models is 4.3. For more information, see Select AUTOSAR Schema.

R2019b

Version: 2.1

New Features

Bug Fixes

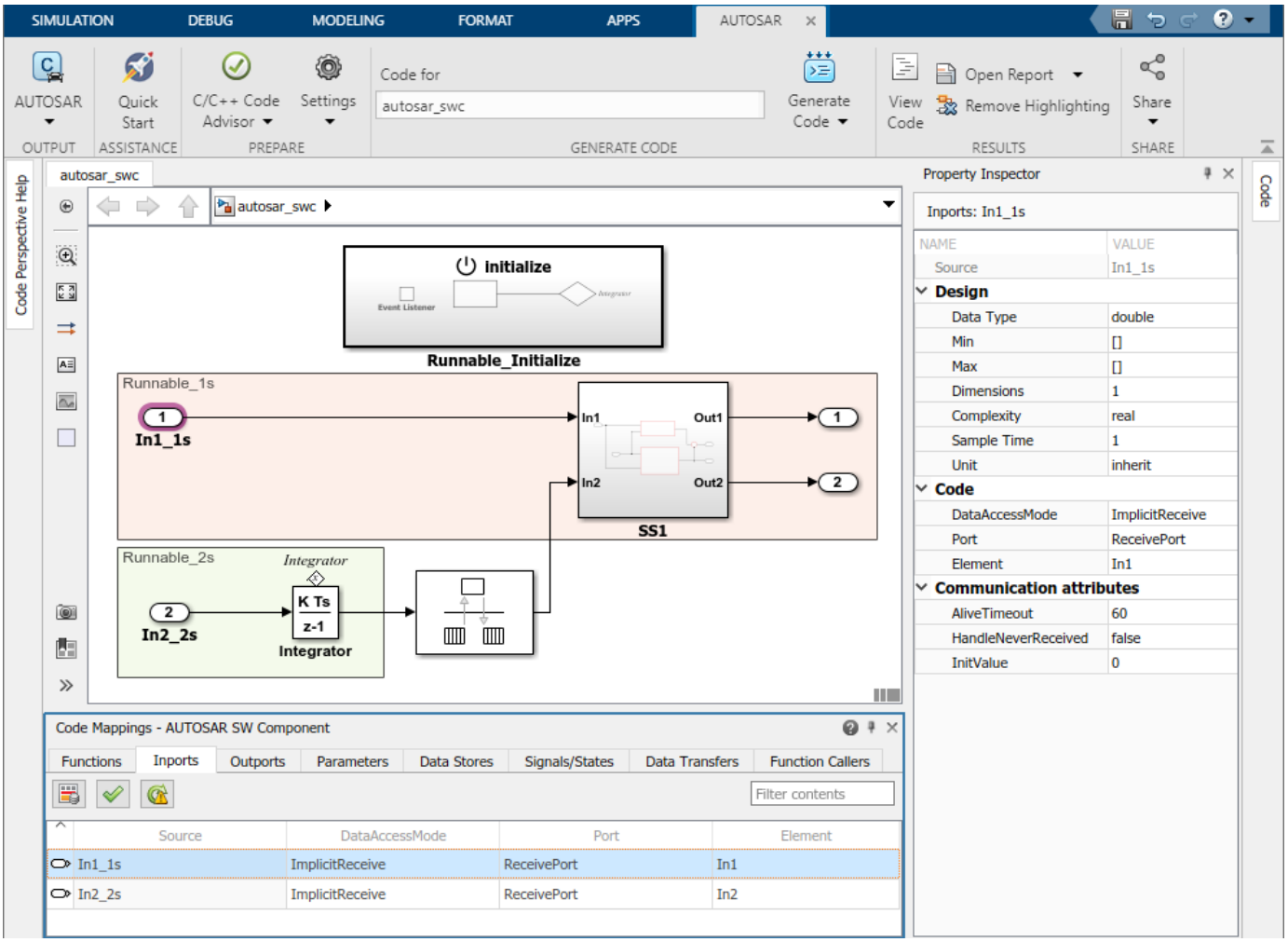
AUTOSAR Component Designer app and AUTOSAR tab

To support AUTOSAR software component modeling, R2019b introduces an AUTOSAR Component Designer app and an **AUTOSAR** tab. The app and the tab support common tasks for component-level AUTOSAR software development.

The AUTOSAR Component Designer app opens an AUTOSAR code perspective, which displays the **AUTOSAR** tab, a help panel, a Property Inspector dialog box, and, directly under the model, the Code Mappings editor.

To use the AUTOSAR Component Designer app, open a component model. On the **Apps** tab, click **AUTOSAR Component Designer**.

- If the model has a mapped AUTOSAR software component, the app opens the AUTOSAR code perspective, with the **AUTOSAR** tab displayed.
- If the model does not have a mapped AUTOSAR software component, the app opens the AUTOSAR Component Quick Start. Use the AUTOSAR Component Quick Start to configure the model for the AUTOSAR Classic or Adaptive Platform. When you complete the quick-start procedure and click **Finish**, your model opens in the AUTOSAR code perspective, with the **AUTOSAR** tab displayed.



AUTOSAR software component modeling

To improve AUTOSAR classic component modeling, R2019b adds submodel data mapping, variation points for calibration data, intuitive port modeling with Simulink bus ports, and runnable scheduling with the Schedule Editor.

Map calibration data for submodels referenced from component models

In R2019b, the Code Mappings editor adds support for submodels referenced from AUTOSAR software component models. In an open submodel, you can:

- Configure the submodel as a model referenced from an AUTOSAR software component model. Use the AUTOSAR Component Quick Start or the AUTOSAR function `autosar.api.create`.
- In the AUTOSAR code perspective, use the Code Mappings editor to configure the submodel internal data.
- To generate C code and AUTOSAR XML (ARXML) files that support run-time calibration of the submodel internal data, open and build the component model that references the submodel.

For more information, see Map Calibration Data for Submodels Referenced from AUTOSAR Component Models.

Export variation points for calibration data

In R2019b, you can export variation points for AUTOSAR calibration data, including:

- Parameters — Calibration, shared internal, instance-specific, or constant memory
- Per-instance memory — C- typed or AR-typed
- Inter-runnable variables (IRVs) — Implicit or explicit

You can model calibration data in combination with different types of variant conditions. Model the variant conditions by using Variant Source and Variant Sink blocks, Variant Subsystem blocks, or model reference variants. When you build your model, the exported AUTOSAR XML (ARXML) files contain the conditionally used data elements and their variation points.

For more information, see Export Variation Points for AUTOSAR Calibration Data.

Model AUTOSAR ports by using Simulink bus ports

In R2019b, you can model AUTOSAR ports by using Simulink bus ports instead of Simulink signal ports. Bus port blocks In Bus Element and Out Bus Element can simplify model interfaces. For more information, see Simplify Bus Interfaces (Simulink).

Bus ports provide a more intuitive way to model AUTOSAR communication ports, interfaces, and groups of data elements. If you model AUTOSAR ports with In Bus Element and Out Bus Element blocks, and type the bus ports with bus objects, basic properties of AUTOSAR ports, interfaces, and data elements are configured without using the AUTOSAR Dictionary. For more information, see Configure AUTOSAR Ports By Using Simulink Bus Ports.

In an AUTOSAR architecture model, if you link to an existing software component model that uses root Inport and Outport blocks, the software automatically converts the signal ports to bus ports.

Configure runnable execution order by using Schedule Editor

In R2019b, you can use the Schedule Editor to schedule and specify the execution order of AUTOSAR component runnables for simulation. In the Schedule Editor, you can:

- View a graphical representation of runnables as partitions in an AUTOSAR component.
- Directly specify the execution order of runnables.

For more information, see Using the Schedule Editor (Simulink) and Configure AUTOSAR Runnable Execution Order By Using Schedule Editor.

You can also use the Schedule Editor in AUTOSAR architecture modeling. For more information, see Configure AUTOSAR Scheduling and Simulation.

Code Mappings editor changes

These changes were made to the Code Mappings editor tabs:

- The **Entry-Point Functions** tab was renamed to **Functions**.
- The tab order, from left to right, was changed to **Functions**, **Inports**, **Outports**, **Parameters**, **Data Stores**, **Signals/States**, **Data Transfers**, and **Function Callers**.
- On the **Parameters** tab, parameter categories were renamed.
  - **Local parameters** was renamed to **Model parameters**.
  - **Parameter arguments** was renamed to **Model parameter arguments**.

Also, the Code Mappings editor now retains mapping information when you perform cut/copy/paste or undo/redo operations on data stores, states, or signals in the model diagram. You can:

- Cut/copy and paste a data store, state, or signal. Code Mapping Editor copies the **Mapped To** information for the element. A signal cut or copy must include the originating block.
- Undo/redo an operation on a data store, state, or signal. **Mapped To** information is retained.

AUTOSAR adaptive software component modeling

To improve AUTOSAR adaptive component modeling, R2019b adds ARXML import, service instance identification, and memory allocation for event sends.

Import ARXML software descriptions

In R2019b, you can import AUTOSAR XML (ARXML) descriptions of adaptive software components, service interfaces, and data types into Simulink. Use the ARXML importer to:

- Create an initial Simulink representation of an AUTOSAR adaptive software component.
- Update a mapped AUTOSAR adaptive component model with shared ARXML definitions of service interfaces and data types.

You can participate in round-trip exchanges of adaptive component ARXML descriptions between Simulink and other development environments.

For more information, see Import AUTOSAR Adaptive Software Descriptions.

Configure service instance identification for ARXML manifest and generated code

In R2019b, you can configure service instance identification for AUTOSAR required and provided ports. When you build an adaptive software component model:

- Exported ARXML files include a service instance manifest file, which describes port-to-service instance mapping.
- Generated C++ code uses the configured service instance information in `ara::com` function calls.

For more information, see Configure AUTOSAR Adaptive Service Instance Identification.

Configure event sends with memory allocation

To send service event data, the AUTOSAR Adaptive Platform supports these methods:

- By reference — The send function uses memory in the application address space. After the send returns, the application can modify the event data.
- By `ara::com` allocated memory — The application requests `ara::com` middleware to allocate memory for the data. This method avoids data copies by `ara::com` middleware and can be more efficient for frequent sends or large amounts of data. But the application loses access to the memory after the send returns.

In R2019b, you can configure adaptive event sends to request `ara::com` memory allocation. Previously, all event sends were by reference.

For more information, see Configure Memory Allocation for AUTOSAR Adaptive Service Data.

AUTOSAR software architecture modeling

R2019b introduces AUTOSAR software architecture modeling for the Classic Platform (requires System Composer). Using a Simulink Start Page template, you create an architecture model. Within the architecture model, you add compositions and components and link new or existing models. You simulate the behavior of the aggregated components. If you have Embedded Coder software, you can export composition and component AUTOSAR XML (ARXML) descriptions and generate component code.

Create architecture models

In R2019b, you can create an AUTOSAR architecture model, which provides resources and a canvas for developing AUTOSAR composition and component models for the Classic Platform. Without leaving the architecture model, you can:

- Add and connect AUTOSAR compositions and components.
- Link components to requirements (requires Simulink Requirements™).
- Define component behavior by creating or linking Simulink models.
- Configure scheduling and simulation.
- Generate and package composition ARXML descriptions and component code (requires Embedded Coder)

Architecture models provide an end-to-end, top-to-bottom AUTOSAR software design workflow. In Simulink, you can author a high-level application design, implement behavior for application components, add Basic Software (BSW) service calls and service implementations, and simulate the application.

For more information, see Create AUTOSAR Architecture Models.

Add and connect compositions and components

After you create an AUTOSAR architecture model, use the composition editor and the Simulink Toolstrip **Modeling** tab to add and connect compositions and components. Common tasks include:

- Add Component and Composition blocks from the palette or toolstrip.
- Create ports by clicking Component or Composition block edges.
- Connect Component and Composition blocks by dragging signal lines.
- Configure additional AUTOSAR properties by using the Property Inspector.

For more information, see Add and Connect AUTOSAR Compositions and Components.

Define component behavior by creating or linking models

After you add and connect Component and Composition blocks in an AUTOSAR architecture model, you can add Simulink behavior to the components. For each AUTOSAR Component block, you can:

- Create a model based on the block interface.
- Link to an implementation model.
- Create a model from an AUTOSAR XML (ARXML) component description.

For more information, see Define AUTOSAR Component Behavior by Creating or Linking Models.

Configure scheduling and simulation

To simulate the behavior of the aggregated components in an AUTOSAR architecture model, click **Run**. To configure scheduling and simulation, you can:

- Add Basic Software (BSW) blocks, including Diagnostic Service Component and NVRAM Service Component blocks, to simulate calls to BSW services.
- Create a test harness model to connect inputs and plant elements to the architecture model.
- Use the Schedule Editor to schedule and specify the execution order of component runnables for simulation.

For more information, see Configure AUTOSAR Scheduling and Simulation.

Generate and package composition ARXML descriptions and component code

In AUTOSAR architecture models, with one click, you can export composition and component ARXML descriptions, generate component code, and package build artifacts. If you initiate an export that encompasses a composition, it generates:

- XML descriptions for compositions, component prototypes, ports, and connectors.
- AUTOSAR compliant code for components.
- A ZIP file that packages ARXML files, code files, and required artifacts for integration with an AUTOSAR run-time environment.

For more information, see Generate and Package AUTOSAR Composition XML Descriptions and Component Code.

R2019a

Introducing AUTOSAR Blockset

In R2019a, the AUTOSAR Blockset product replaces the Embedded Coder Support Package for AUTOSAR Standard. You use AUTOSAR Blockset to design and simulate AUTOSAR software.

AUTOSAR Blockset provides an AUTOSAR dictionary and blocks for developing Classic and Adaptive AUTOSAR software using Simulink models. You can define AUTOSAR software component properties, interfaces, and data types, and map them to existing Simulink models using the AUTOSAR editor. Alternatively, the blockset provides an application interface that lets you automatically generate new Simulink models for AUTOSAR by importing software component and composition descriptions from AUTOSAR XML files.

AUTOSAR Blockset provides blocks and constructs for AUTOSAR library routines and Basic Software (BSW) services, including Memory Access and Diagnostics. By simulating the BSW services together with your application software model, you can verify your AUTOSAR ECU software without leaving Simulink.

AUTOSAR Blockset supports C and C++ production code generation and AUTOSAR XML file export (with Embedded Coder). The software is qualified for use with the ISO 26262 standard (with IEC Certification Kit).

Product restructuring overview

In R2019a, AUTOSAR Blockset provides AUTOSAR software design and simulation support that previously was provided by Embedded Coder and the Embedded Coder Support Package for AUTOSAR Standard. In the new product, in general, software interfaces and development workflows are unchanged from previous releases. Product restructuring introduced these differences:

- Product requirements and dependencies:
  - AUTOSAR Blockset requires only MATLAB and Simulink.
  - Embedded Coder is required to generate AUTOSAR C/C++ code and XML files.
  - No support package is required.
- Second development platform:
  - R2019a adds AUTOSAR Adaptive Platform support to existing Classic Platform support.
  - Classic and adaptive development workflows are logically distinct, and are handled separately in AUTOSAR Blockset software interfaces and documentation.
  - Existing Classic Platform workflows are unchanged, except where enhanced by R2019a features.
- AUTOSAR block libraries relocated:
  - AUTOSAR Blockset libraries replace block libraries from Embedded Coder and Embedded Coder Support Package for AUTOSAR Standard.
  - Existing block names are unchanged; new blocks are added.
- AUTOSAR example models renamed and relocated:
  - For example, models `autosar_swc*.slx` in folder `examples/autosarblockset` replace models `rtwdemo_autosar_swc*.slx`, formerly in folder `toolbox/rtw/rtwdemos`.
  - Models remain accessible from the MATLAB command line.

- AUTOSAR HTML help and PDF books restructured and relocated:
  - AUTOSAR Blockset HTML help replaces AUTOSAR help in Embedded Coder and Embedded Coder Support Package for AUTOSAR Standard.
  - AUTOSAR Blockset Reference, User's Guide, and Release Notes PDF books replace the Embedded Coder AUTOSAR PDF book.

Resources for upgrading from AUTOSAR Standard support package

If you are upgrading to AUTOSAR Blockset from Embedded Coder Support Package for AUTOSAR Standard, review information about compatibility and upgrade issues at the following locations:

- AUTOSAR Blockset Release Notes (latest release). In HTML notes, to view only compatibility considerations, select **Incompatibilities Only**.
- On the MathWorks® web site, view the R2018b Embedded Coder Support Package for AUTOSAR Standard Release Notes. This document provides compatibility information for AUTOSAR Standard support package releases up through R2018b. To locate compatibility considerations in the page, click **expand all in page** and search for `compatibility`.

You can also refer to the rest of the archived documentation, including release notes, for Embedded Coder Support Package for AUTOSAR Standard and Embedded Coder.

Compatibility Considerations

If you are upgrading to R2019a AUTOSAR Blockset from Embedded Coder Support Package for AUTOSAR Standard, these minor compatibility considerations might apply.

- R2019a changes AUTOSAR example model locations. If you reference model names in scripts, update the model names. The new example models use the prefix `autosar_` and are located in the folder *`matlabroot`*`/examples/autosarblockset`.
- R2019a restructures and relocates AUTOSAR HTML help and PDF books. Consider updating any bookmarks or references.

For function interface compatibility considerations related to R2019a new features, see “Incrementally auto-configure and map new Simulink elements in AUTOSAR model” on page 8-5.

For function interface changes that support R2019a workflow improvements, consider migrating to the improved workflows. For more information, see “Reuse existing AUTOSAR elements for software components created in Simulink” on page 8-5 and the lookup tables subsection of “Code Perspective enhancements for mapping data stores, model workspace parameters, and internal signals and states” on page 8-6.

AUTOSAR Classic Platform support extended to Release 4.3.1

R2019a extends support of AUTOSAR Classic Platform schema version 4.3 to include schema revision 4.3.1. AUTOSAR Blockset supports the new schema revision for import and export of ARXML files and generation of AUTOSAR-compatible C code.

If you import schema 4.3.1 ARXML code into Simulink, the ARXML importer detects and uses the schema version and revision, and sets the schema version parameter in the model. For more information on schema import and export, see Select an AUTOSAR Schema.

Support for AUTOSAR Adaptive Platform Release 18.10

In R2019a, you can flexibly model the structure and behavior of software components for the AUTOSAR Adaptive Platform. The Adaptive Platform defines a service-oriented architecture for automotive components that must flexibly adapt to external events and conditions. AUTOSAR Blockset supports Adaptive Platform Release 18.10.

You can:

- Model AUTOSAR adaptive software components using event-based communication.
- With Embedded Coder, generate C++ code compliant with the AUTOSAR Adaptive Platform.
- Deploy generated code and build a target executable.

For more information, see Model AUTOSAR Adaptive Software Components and Configure AUTOSAR Adaptive Software Components.

Generate AUTOSAR IFL and IFX library routines for interpolation using AUTOSAR lookup table blocks

R2019a provides AUTOSAR lookup table blocks that you can configure to generate specific IFL and IFX interpolation routines in your AUTOSAR-compliant C code.

- Curve – Approximate one-dimensional function
- Curve Using Prelookup – Use precalculated index and fraction values to accelerate approximation of one-dimensional function
- Map – Approximate two-dimensional function
- Map Using Prelookup – Use precalculated index and fraction values to accelerate approximation of two-dimensional function
- Prelookup – Compute index and fraction for Curve Using Prelookup or Map Using Prelookup block

To configure AUTOSAR library routine generation for a lookup table block, open the block parameters dialog box. Modify the block parameters to configure a specific AUTOSAR routine. If you select the AUTOSAR 4.0 code replacement library (CRL) for your model, code generated from the block is replaced with the AUTOSAR library routine that you configured. For more information, see Configure Lookup Tables for AUTOSAR Measurement and Calibration and Code Generation with AUTOSAR Code Replacement Library.

Enhanced AUTOSAR model creation in Simulink using Component Quick Start or Simulink Start Page

R2019a provides new resources for creating an AUTOSAR software component model in Simulink:

- AUTOSAR Component Quick Start – Provides easy, ordered steps for mapping an open Simulink model to an AUTOSAR software component. In an open model, set **System target file** to an AUTOSAR target and open Code perspective. AUTOSAR Component Quick Start opens, and you can quickly step through AUTOSAR software component configuration.
- Simulink Start Page – Provides AUTOSAR Blockset model templates. Select a Classic Platform or Adaptive Platform template, which you can use as a starting point for developing an AUTOSAR software component model.

For more information, see Create AUTOSAR Software Component in Simulink.

Reuse existing AUTOSAR elements for software components created in Simulink

R2019a enhances design and development of AUTOSAR software components created in Simulink by supporting generalized reuse of existing AUTOSAR element definitions. Changes include:

- Improved import of AUTOSAR element definitions with function `updateAUTOSARProperties`
- Option to treat imported elements as read-only (the default), preventing definition changes, or read-write

```
% Import XML definitions of AUTOSAR software address methods as read-write elements
ar = arxml.importer('SwAddressMethods.arxml');
updateAUTOSARProperties(ar,'mySWC','ReadOnly',false);
```

- AUTOSAR Dictionary displays **Exported XML File** name for packageable elements
- AUTOSAR XML export preserves ARXML file structure for imported noncomponent elements
- AUTOSAR property functions `createEnumeration` and `createNumericType` generate Simulink definitions for working with imported AUTOSAR elements
- AUTOSAR Component Quick Start supports import of AUTOSAR element definitions as part of initial configuration

Function `updateAUTOSARProperties` generalizes and replaces function `updateReferences`. If you previously used `updateReferences` to import existing AUTOSAR elements into a Simulink-originated component, consider replacing `updateReferences` function calls with `updateAUTOSARProperties` function calls.

For more information, see the `updateAUTOSARProperties` reference page, Reuse AUTOSAR Element Descriptions, and example Reuse AUTOSAR Elements in Component Model.

Incrementally auto-configure and map new Simulink elements in AUTOSAR model

R2019a enhances AUTOSAR function `autosar.api.create` so that you can incrementally configure and map Simulink elements as you modify your AUTOSAR model. When used with a mapped AUTOSAR model, function call `autosar.api.create(modelName)`:

- Preserves current model configuration and mapping.
- Finds and maps unmapped model elements.
- Updates the AUTOSAR Dictionary for deleted model elements.

For more information, see Incrementally Update AUTOSAR Mapping after Model Changes.

Compatibility Considerations

R2019a changes the default behavior of `autosar.api.create`. If you call the function without a *`mode`* argument (`init`, `default`, or `incremental`), the function behavior depends on the mapping state of the model.

- If the model is not mapped to an AUTOSAR software component, the function creates a Simulink to AUTOSAR mapping in `default` mode. In this mapping, Simulink inports and outports are mapped to AUTOSAR ports with default AUTOSAR properties.
- If the model is already mapped to an AUTOSAR software component, the function updates the existing mapping in `incremental` mode. The function finds and maps unmapped model elements, and updates the AUTOSAR Dictionary for deleted model elements.

Previously, if you called the function without a *`mode`* argument, the function created a Simulink to AUTOSAR mapping in `init` mode. In this mapping, Simulink inports and outports were not mapped to AUTOSAR ports. Also, if the model was already mapped to an AUTOSAR software component, the new mapping replaced the existing mapping.

If a MATLAB script previously called the function `autosar.api.create` without a *`mode`* argument, update the script to account for the R2019a behavior change. Consider taking advantage of the new behavior or, if you rely on the previous `init` mode behavior, specify `init` as the *`mode`* argument.

Code Perspective enhancements for mapping data stores, model workspace parameters, and internal signals and states

R2019a improves the Simulink to AUTOSAR mapping workflow by enhancing Code Mappings editor tabs and improving graphical display of model data.

Map data stores to AUTOSAR component per-instance and static memory for calibration

In R2019a, AUTOSAR Code Mappings editor adds a **Data Stores** tab. In AUTOSAR code perspective, use the tab to map internal data store elements to AUTOSAR component per-instance and static memory for calibration. Alternatively, you can use equivalent AUTOSAR map functions `getDataStore` and `mapDataStore`. For more information, see Map Data Stores to AUTOSAR Variables, Configure AUTOSAR Per-Instance Memory, and Configure AUTOSAR Static Memory.

Map model workspace parameters to AUTOSAR component instance-specific parameters for calibration

R2019a extends model workspace parameter mapping to support AUTOSAR per-instance parameters. Using Code Mappings editor, **Parameters** tab, you can map model workspace parameters that are marked as model arguments to AUTOSAR instance-specific parameters for calibration. Alternatively, you can use equivalent AUTOSAR map functions `getParameter` and `mapParameter`. For more information, see Map Model Workspace Parameters to AUTOSAR Component Internal Parameters, Shared and Per-Instance Parameters, and Configure AUTOSAR Shared or Per-Instance Parameters.

Signals and States tabs combined

R2019a combines AUTOSAR Code Mappings editor tabs **Signals** and **States** into a **Signals/States** tab. Use the **Signals/States** tab to map Simulink block signals or states to AUTOSAR variables for calibration.

AUTOSAR map functions `getSignal`, `getState`, `mapSignal`, and `mapState` are not affected by the **Signals/States** tab change.

Lookup Tables tab removed

R2019a removes the **Lookup Tables** tab from AUTOSAR Code Mappings editor. The **Lookup Tables** tab was used to map Simulink lookup table or breakpoint objects created in the base workspace to

AUTOSAR calibration parameters for integrated and distributed lookups. Instead, you can use the **Parameters** tab to map lookup table or breakpoint objects created in the model workspace. See Map Model Workspace Parameters to AUTOSAR Component Internal Parameters.

AUTOSAR map functions `getLookupTable` and `mapLookupTable` are not affected by the **Lookup Tables** tab removal. However, consider switching from using base workspace objects with functions `getLookupTable` and `mapLookupTable` to using model workspace objects with functions `getParameter` and `mapParameter`.

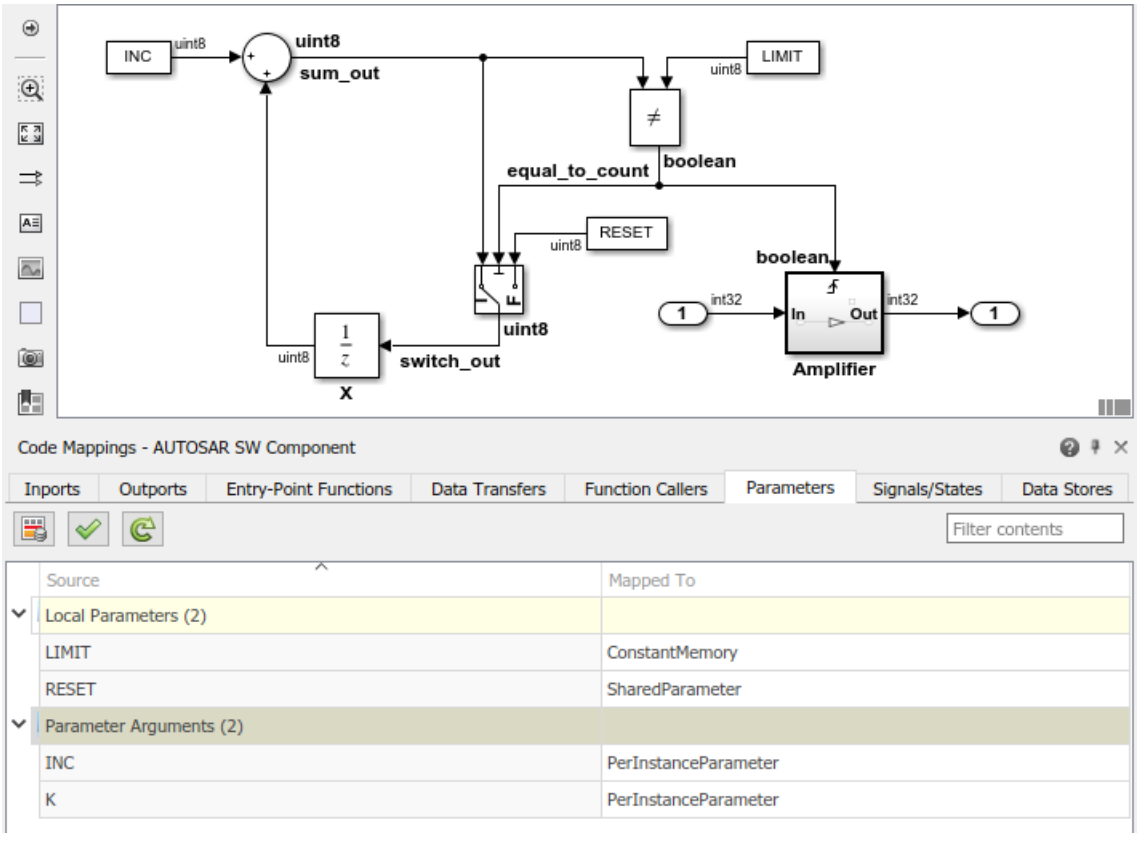
For an example of the model workspace-based workflow, see Configure Lookup Tables for AUTOSAR Measurement and Calibration.

Model data grouped in categories for easy reference

To improve navigation and usability of model data mapping tables, Code Mappings editor now groups model data into categories. The editor displays a node for each category. For example:

- The **Signals/States** tab displays nodes for **Signals** and **States**.
- The **Parameters** tab displays nodes for **Local Parameters** and **Parameter Arguments**.

To focus on entries of interest and reduce scrolling, you can expand and collapse model data categories.



Parameter and signal calibration attributes removed from AUTOSAR data objects

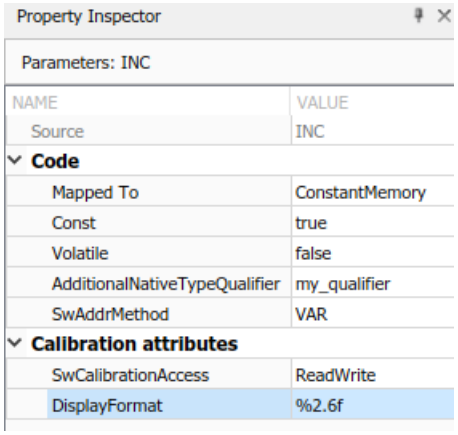
As part of the AUTOSAR parameter and variable calibration enhancements described in “Code Perspective enhancements for mapping data stores, model workspace parameters, and internal signals and states” on page 8-6, R2019a removes the calibration attributes **SwCalibrationAccess** and **DisplayFormat** from AUTOSAR parameter and signal data objects. In the dialog box views of objects such as `AUTOSAR.Parameter` and `AUTOSAR.Signal`, the calibration attributes and the **Additional attributes** tab no longer appear.

Also, the AUTOSAR parameter and signal data objects can no longer be created by using the Model Explorer. You instantiate the AUTOSAR objects in the MATLAB Command Window.

Compatibility Considerations

AUTOSAR software component models that model AUTOSAR parameters and variables by using AUTOSAR data objects in the base workspace can no longer use AUTOSAR object interfaces to set the calibration attributes **SwCalibrationAccess** and **DisplayFormat**.

Consider migrating the AUTOSAR models to the new Code Mappings editor workflow, which maps Simulink model elements to AUTOSAR component elements. In the Code Mappings workflow, AUTOSAR parameters and variables are scoped to the associated software component, and are modeled by using Simulink model-workspace parameters and internal signals, states, and data stores. When you select a parameter, signal, state, or data store in the Code Mappings editor, you can modify its AUTOSAR calibration attributes in the Property Inspector.



For more information, see Map AUTOSAR Elements for Code Generation.

Change to AUTOSAR XML import behavior for ArTypedPerInstanceMemory element with Service Dependency

In R2019a, when you import an ArTypedPerInstanceMemory element with a Service Dependency from an ARXML file into Simulink, the importer:

- Adds a Data Store Memory block to the model.
- Adds a corresponding `Simulink.Signal` object to the model workspace.

- Maps the internal data store to AUTOSAR component per-instance memory for calibration.

Previously, the importer added a Data Store Memory block to the model and added a corresponding `AUTOSAR.Signal` object to the base workspace.

Compatibility Considerations

If existing AUTOSAR infrastructure expects import of an ArTypedPerInstanceMemory element with a Service Dependency to add an `AUTOSAR.Signal` object to the base workspace, update the infrastructure to reflect the new importer behavior.

Model Advisor checks for AUTOSAR Blockset configuration and lookup table block code replacements

The following table identifies the new Model Advisor checks for AUTOSAR Blockset:

Model Advisor Check	Check ID
Check model configuration parameters for AUTOSAR compliance	mathworks.autosar.autosar_configset
Check compatibility of AUTOSAR Interpolation Routines	mathworks.autosar.lut_replacement_check

AUTOSAR contextual tab in the Simulink Toolstrip Tech Preview

In R2019a, you have the option to turn on the Simulink Toolstrip. For more information, see Simulink Toolstrip Tech Preview replaces menus and toolbars in the Simulink Desktop.

The Simulink Toolstrip includes contextual tabs, which appear only when you need them. The AUTOSAR contextual tab includes options for completing actions that apply only to AUTOSAR Blockset.

To access the AUTOSAR tab, with the Simulink Toolstrip activated, click **Apps** and then click the **AUTOSAR** button. The **AUTOSAR** tab appears.

Documentation does not reflect the addition of the AUTOSAR contextual tab.