

Simulink® Check™ Release Notes



MATLAB® & SIMULINK®



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

Simulink[®] *Check*[™] *Release Notes*

© COPYRIGHT 2017-2023 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.

SIL and PIL Code Testing Dashboards: Assess the status of code testing and coverage for compliance to standards like ISO 26262	1-2
Explore traceability relationships for design artifacts, requirements, tests, and results in the dashboards	1-3
Analyze artifacts from referenced projects in the dashboards	1-3
Trace requirement links to MATLAB code files in the Model Testing Dashboard	1-4
Troubleshoot artifact issues with Artifact Issues tab in the dashboards	1-4
View integer overflow coverage in the Model Testing Dashboard	1-5
View individual test iterations in the metrics and dashboards	1-6
Faster response time for finding clones in a model	1-7
Load a specific justifications file when running Model Advisor checks ..	1-7
JMAAB 6.0 Support: New and updated Model Advisor checks to enable compliance with JMAAB 6.0 modeling style guidelines	1-8
Analyze Simulink functions using Model Slicer	1-10
Simulink Check features available in Simulink Online	1-10
Limitations	1-10
Functionality being removed or changed	1-10
New metric IDs for test status and coverage metrics in the Model Testing Dashboard	1-10
Metric IDs removed for percentage results in the Model Testing Dashboard	1-11
metric.Result objects no longer return the fields Type and ParentType ..	1-13
Check hisl_0311: "Check safety-related diagnostic settings for Stateflow" no longer checks for the use of machine-parented data	1-13

CI/CD Automation for Simulink Check (October 2022; Version 22.2.0)	2-2
.....	
Model Maintainability Dashboard: Assess the complexity and maintainability of your design across the model development lifecycle	2-2
.....	
View external MATLAB code associated with units and components in the Model Maintainability Dashboard	2-3
.....	
Navigation enhancements for the Model Testing Dashboard and new Model Maintainability Dashboard	2-4
.....	
Identify the sources of overall achieved coverage in the Model Testing Dashboard	2-5
.....	
View internal and external test harnesses in the Model Testing Dashboard	2-6
.....	
Generated report opens automatically for the Model Testing Dashboard and Model Maintainability Dashboard	2-6
.....	
High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines	2-6
.....	
Use older Model Advisor configuration files in newer versions of MATLAB	2-6
.....	
Identify clones in a model during edit time	2-7
.....	
Identify Bus Selector and Bus Creator blocks during edit time	2-7
.....	
Enhancements to edit-time checking for numeric efficiency issues	2-7
.....	
Add bus elements as starting points using Model Slicer	2-8
.....	
Functionality being removed or changed	2-8
Metrics Dashboard user interface, metricdashboard function, and slmetric package API will be removed	2-8
Replace instances of "RequirementsBasedModelUnitTesting" with "ModelUnitTesting"	2-9
Check hisf_0007: Usage of Junction Conditions(maintaining mutual exclusions) will be removed	2-9

Toolstrip-based UI for Model Advisor	3-2
---------------------------------------------	-----

Model Advisor Check Result Statuses	3-3
Author Model Advisor checks that run at edit-time	3-3
Model Advisor disables edit-time checks with high execution times	3-4
Create help for custom Model Advisor checks	3-4
Justify Model Advisor check violations	3-5
High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines	3-5
MAB Checks: Improve quality and compliance with guidelines	3-6
Enable edit-time checking for models by using new configuration parameter	3-6
Associate a Model Advisor configuration file with a model	3-6
Enable artifact tracing to track changes and project artifacts in the Model Testing Dashboard	3-6
Include subsystem-level tests in the Model Testing Dashboard	3-7
Trace System Composer architecture models in the Model Testing Dashboard	3-7
View test harnesses in the Artifacts panel for each unit in the Model Testing Dashboard	3-7
Hide requirements metrics in the Model Testing Dashboard	3-8
Identify which artifacts contribute to metrics in the Model Testing Dashboard	3-8
Detect changes to artifact traceability and metric results in the Model Testing Dashboard	3-8
Navigate between project artifacts in the Model Testing Dashboard	3-9
Refactor similar clones across the model	3-9
Find clones by using multiple external library files	3-9
Clone Detection Exclusion Editor improvements	3-10
Inspect test cases generated in Simulink Design Verifier by using Model Slicer	3-10
Debug equivalence tests by using Model Slicer	3-10
CI/CD Automation for Simulink Check (August 2022; Version 22.1.0) ..	3-10

Functionality being removed or changed	3-11
getAvailableMetricIds function returns metrics from the Model Testing Dashboard app and Design Cost Estimation app	3-11
ModelAdvisor.CheckResult returns different values for the property status	3-11
Avoid using process callback functions	3-12

R2021b

View compliance status of metrics in the Model Testing Dashboard	4-2
Organize models using unit testing hierarchy in the Model Testing Dashboard	4-3
Measure pass and fail criteria metrics in the Model Testing Dashboard	4-4
Added functions for programmatically analyzing requirements-based testing metrics	4-4
Trace additional test results in the Model Testing Dashboard	4-5
View summary of artifacts for each unit in the model testing metrics report	4-5
Artifact tracing enhancements for the Model Testing Dashboard	4-6
Generate report from the Model Testing Dashboard	4-7
Find clones anywhere within the model	4-7
Programmatically detect clones in multiple models	4-9
Improve Code Efficiency by Merging Multiple Interpolation Using Prelookup Blocks	4-9
Improved edit-time check diagnostic interface for block constraint violations	4-10
Simplified block constraint check authoring	4-10
Additional Model Slicer support for Simulink constructs	4-10
Guideline Sub-ids for additional MAB/JMAAB checks	4-11
High Integrity Systems Modeling Checks: Improve quality and compliance to guidelines	4-11
Observe impact of Simulink parameters using Model Slicer	4-12

Additional checks to verify compliance with CERT C secure coding standards	4-13
Enhancements to edit-time checking to identify more incompatibilities	4-14
Functionality being removed or changed	4-14
ModelAdvisor.ListViewParameter class and ModelAdvisor.Check ListViewVisible property will be removed	4-14

R2021a

View detailed traceability paths and diagnostics for requirements-based testing artifacts	5-2
Generate model testing metrics report	5-2
Find artifacts in the Model Testing Dashboard by using navigation enhancements	5-2
Select software component models for analysis in the Model Testing Dashboard	5-3
Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines	5-3
Create user-defined fields and review additional information when designing block constraint checks	5-4
Add, remove, clear, get, save, and load exclusions programmatically in the Model Advisor Exclusion Editor	5-4
Additional checks for JMAAB and MAB Guidelines	5-4
Use Model Advisor Exclusion Editor with Stateflow	5-5
Enhancements to edit-time checking to identify additional incompatibility issues	5-5
Inspect enhanced MCDC objectives generated by Simulink Design Verifier using Model Slicer	5-6
Debug test failures using Model Slicer	5-6
Detect and replace clones in a model with command-line APIs	5-6
Improved clone detection for refactoring models with nested clones and masked subsystem	5-6
Edit-time checking support for library blocks	5-7

Improved analysis of models that have callbacks in the Model Testing Dashboard	5-7
---------------------------------------------------------------------------------------------	------------

R2020b

Model Testing Dashboard: Track completeness of requirements-based testing for compliance to standards such as ISO 26262	6-2
Faster calculation of Model Advisor metric checks	6-2
Debug counter examples from Design Error Detection results in Simulink Design Verifier using Model Slicer	6-3
Updates to JMAAB Checks	6-3
Test failure debugging using Model Slicer	6-4
Improvements to Model Advisor Exclusion Editor	6-4
Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines	6-5
Model Advisor checks for ISO 25119 and EN 50657 standards	6-5
Refactor models by replacing exact clones with Subsystem Reference blocks	6-6

R2020a

Model Advisor checks to support MAB v5.0 modeling guidelines	7-2
Automate checking of models to comply with JMAAB 5.1 modeling guidelines	7-7
Model Advisor checks for verifying compliance with DO-254 safety standards	7-8
Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines	7-11
Customize your Model Advisor by using the redesigned configuration editor	7-12
Removal of the AggregateComponentDetails property	7-13
Enhanced calculation of cyclomatic complexity	7-13

Removal of restore point from the Model Advisor	7-14
Removal of the Model Advisor Results Advisor View	7-14
New Customizing Model Advisor Example	7-15

R2019b

Refactor models using clone detection	8-2
Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines	8-2
Check blocks not supported for code generation by using edit-time checks	8-2
JMAAB 5.1 Support: Automate checking of models to comply with JMAAB 5.1 modeling style guidelines	8-3
Access Simulink Check capabilities from Simulink Toolstrip	8-4
Collect metric data for referenced models running in accelerated mode	8-4
Enhancement to Model Advisor compliance metrics	8-5

R2019a

Model Slicer available with Simulink Check	9-2
Hierarchical view of metrics dashboard results	9-2
Filters for metrics dashboard results	9-2
Streamline compliance with modeling guidelines by using enhanced edit-time check diagnostics	9-3
Edit-Time check diagnostics interface	9-3
Model Advisor checks evaluated by using edit-time checking	9-3
JMAAB 5.1 Support: Automate checking of models to comply with JMAAB 5.1 modeling style guidelines	9-4
Additional MAAB 3.0 Checks: Improve quality and compliance to guidelines	9-5
Added functions for edit-time checking	9-5

Model optimization by sharing prelookup operation of n-D lookup tables	9-6
Clone metrics include exact and similar clones	9-6
Model slicer support for multi-instance model reference	9-6
Model Advisor option to compile model for code generation	9-6
Updates for verifying compliance with High-Integrity Systems Modeling guidelines	9-7
MISRA C:2012 and Secure Coding checks to improve compliance of generated code	9-9
Tech Preview of model refactoring using clone detection	9-10

R2018b

Metrics Dashboard Customization: Configure compliance metrics, add metric thresholds, and customize Metrics Dashboard layout	10-2
Configure Compliance Metrics	10-2
Add Metric Thresholds	10-2
Customize Metrics Dashboard Layout	10-3
Simscape Support with Clone Detection: Detect and refactor clones in Simscape Models	10-4
JMAAB 4.01 Support: Automate checking of models to comply with JMAAB 4.01 modeling style guidelines	10-4
Additional MAAB 3.0 and High Integrity Checks: Improve quality and compliance to guidelines	10-6
High Integrity Systems Modeling Checks: Use the additional conditions to check the configuration parameters	10-12
MISRA C:2012 and Secure Coding Standards: Improve compliance of generated code by using updated Model Advisor checks	10-12
Mnemonic Support: Use keyboard shortcuts with Metrics Dashboard	10-13
Check Style for Model Advisor: Create checks that generate interactive reports	10-13
Functionality Being Removed or Changed	10-17

Additional Checks for MAAB 3.0 and JMAAB 4.0 Guidelines: Automate checking for MAAB 3.0 guidelines for Simulink, Stateflow, Variant Subsystems, and MATLAB Function Blocks and JMAAB 4.0 guidelines	11-2
MAAB Modeling Checks	11-2
JMAAB Modeling Checks	11-4
Block Constraint Authoring with Edit-Time: Define checks for supported or unsupported blocks and parameters while editing	11-6
Clone Refactoring Workflow: Apply multiple refactoring steps to the same model	11-6
Automatic Refactoring for Similar Clones: Add masks to similar clones and refactor model	11-7
Clone Detection Exclusion Editor: Exclude subsystems and referenced models from clone detection	11-7
Automatic Data Store Memory Block Elimination: Identify and refactor Data Store Memory Block blocks with Model Transformer	11-7
Grid Visualization for Metrics: View results of Model Advisor checks in a grid to identify patterns in results	11-7
MathWorks High-Integrity Guidelines and Checks: Verify compliance with safety standards by using high-integrity checks and guidelines	11-8
High-Integrity System Modeling Checks	11-8
High-Integrity Modeling Guidelines	11-10
MISRA C: 2012 Modeling Checks: Improve compliance of generated code by using MISRA C:2012 standards checks	11-12
Secure Coding Modeling Checks: Update to Secure Coding compliance checks	11-13
Enhanced Edit-Time Checking Support: Edit-time checking for blocks not recommended for C/C++ production code deployment	11-13
Model Advisor Support for Inactive Variants: Run Model Advisor checks on active and inactive variants and generate report	11-13
Metric Engine Improvement: Collect and analyze metric data faster	11-13
Model Metric APIs: Removed Model block architectural component	11-13

Simulink Verification and Validation Packaging: Moved compliance checking, model metrics, clone detection and refactoring, edit-time checking and model transformer to Simulink Check	12-2
Metrics Dashboard: Collect and view metric data for quality assessment	12-2
MathWorks High-Integrity Guidelines and Checks: Verify compliance with safety standards by using high-integrity checks and guidelines	12-2
Categorization of the Model Advisor Checks for High-Integrity Systems	12-2
High-Integrity Model Advisor Checks for DO-178C/DO-331 Standards	12-3
High-Integrity Model Advisor Checks for EN 50128, IEC 61508, IEC 62304, and ISO 26262 Standards	12-4
High-Integrity Modeling Guidelines	12-5
Modeling Support for Secure Coding Standards: Check model for compliance with secure coding requirements in CERT C, CWE, ISO/IEC TS 17961 standards to improve security of generated code	12-11
MISRA C: 2012 Modeling Checks: Improve compliance of generated code by using new MISRA C:2012 standards checks	12-13
DO-178C/DO-331 Modeling Checks: Removed Model Advisor check "Check model for block upgrade issues"	12-14
Model Metrics: Evaluate model quality by using new metric algorithms	12-14
Model Metric APIs: Create custom metrics with more detailed results and determine passed or failed compliance checks	12-14
Model Advisor Configuration Editor: Select edit-time checks from folders	12-15

R2023a

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

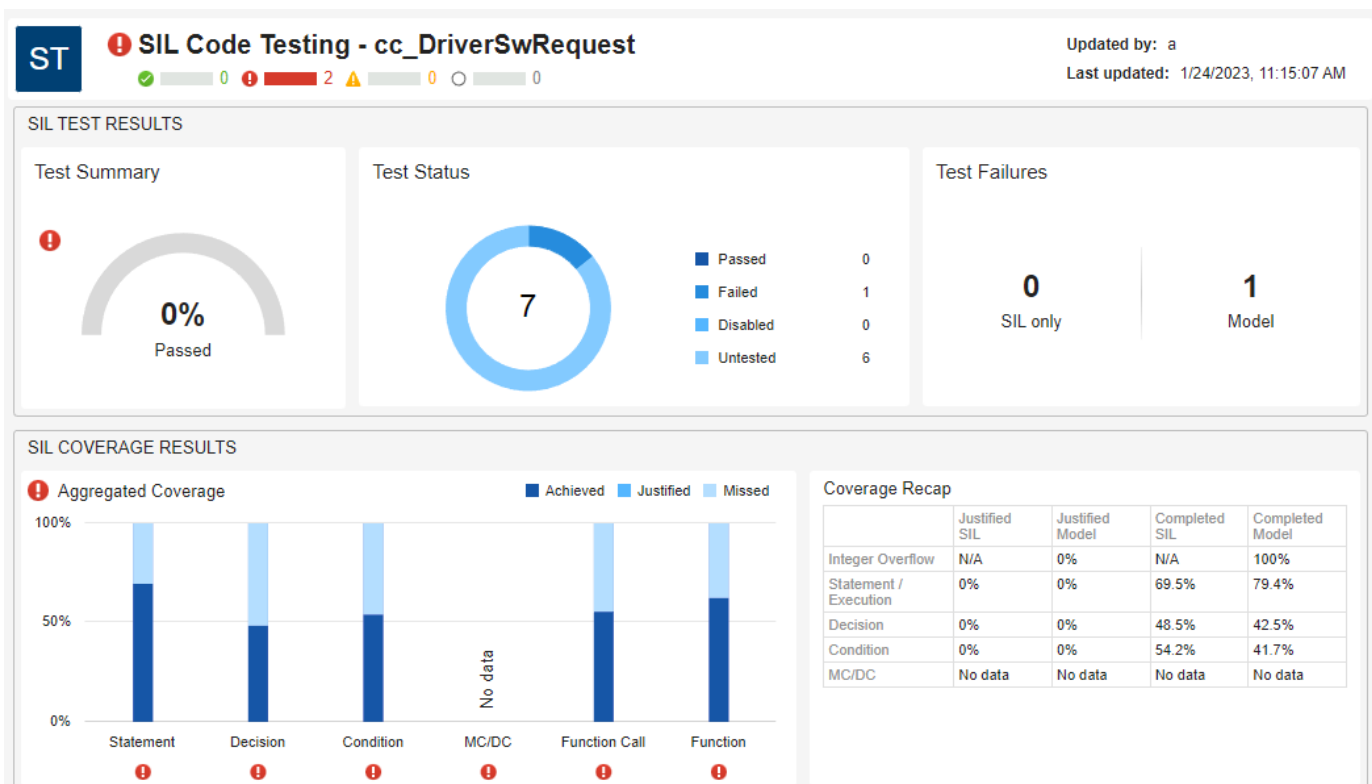
SIL and PIL Code Testing Dashboards: Assess the status of code testing and coverage for compliance to standards like ISO 26262

Assess the status and quality of your code testing by using the metric results in the **SIL Code Testing** dashboard and **PIL Code Testing** dashboard. The metrics measure different aspects of code testing completeness for software-in-the-loop (SIL) and processor-in-the-loop (PIL) tests. The metrics are based on industry-recognized standards such as ISO 26262 and DO-178C.

Use the dashboards to view the:

- Compliance status of SIL and PIL test results and coverage
- Detailed test status breakdowns
- Comparisons to relevant model testing results and coverage

Based on the metric results, you can identify and fix failing test results, gaps in coverage, and anomalies between model and code testing results. The dashboard widgets show a summary of the testing metric data for a software unit. To explore the data in more detail, click an individual widget. A table lists the artifacts in the unit and their results for the metric. The table provides hyperlinks to open each artifact so that you can view details about the artifact and address testing quality issues. For more information, see “View Status of Code Testing Activities for Software Units in Project” and “Identify and Troubleshoot Gaps in Code Testing Results and Coverage”.



You can also collect the metrics programmatically by using the same metric API as the **Model Testing Dashboard**. For more information, see `metric.Engine`.

Viewing the metric results data and details in the dashboard requires a Simulink® Check™ license. To collect results for a metric, you must have the licenses required to edit the associated artifacts, such

as Simulink Test™ and Simulink Coverage™. For more information on the metrics and the required licenses, see “Code Testing Metrics”.

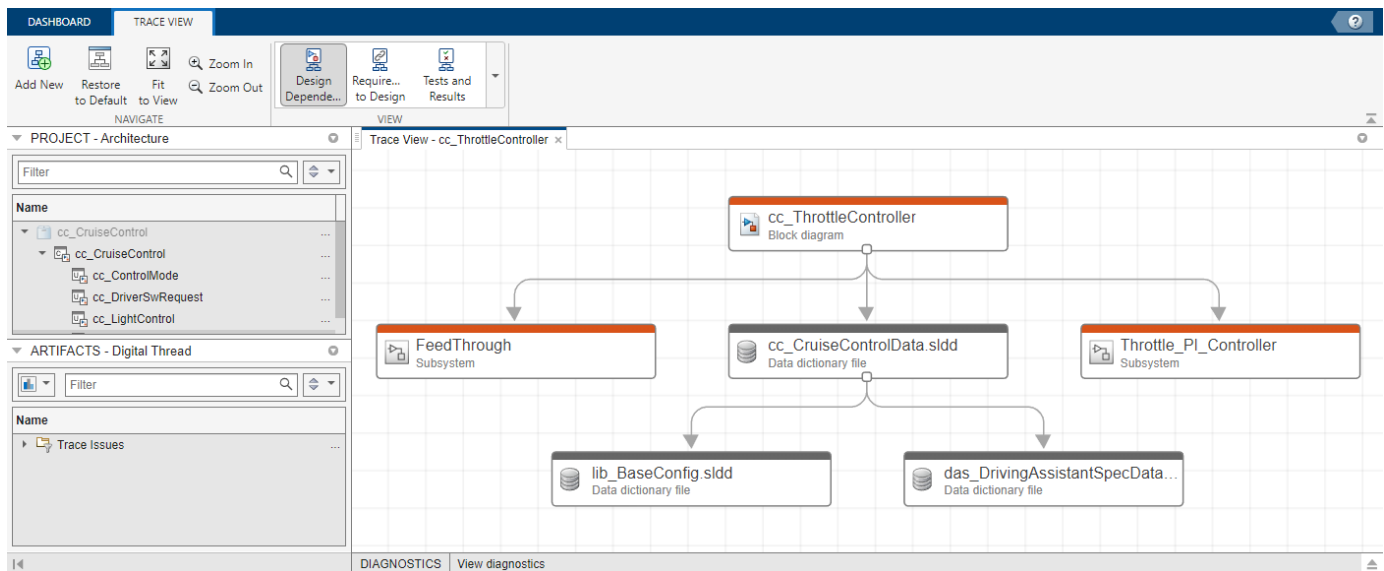
Explore traceability relationships for design artifacts, requirements, tests, and results in the dashboards

In R2023a, the dashboards provide additional and improved trace views that help you visually explore traceability information in a project. A *trace view* is an interactive diagram that shows a certain preset of traceability information for artifacts in a project. The trace views provide a detailed, tree-like structure of project artifacts and show trace relationships, individual artifact information, and a hierarchical view of trace relationships between the artifacts in the selected unit or component.

Previously, you could only see the traceability path from a specific artifact to its unit or component when you right-clicked the artifact and clicked **View trace to dashboard**.

Now, there are three different types of trace views for units and components:

- **Design Dependency** — Shows the library blocks, data dictionaries, model references, and MATLAB® files that trace to the unit or component
- **Requirement to Design** — Shows the functional requirements that trace to the unit or component
- **Tests and Results** — Shows the test cases and test results that trace to the unit or component



For more information, see “Explore Traceability Information for Units and Components”.

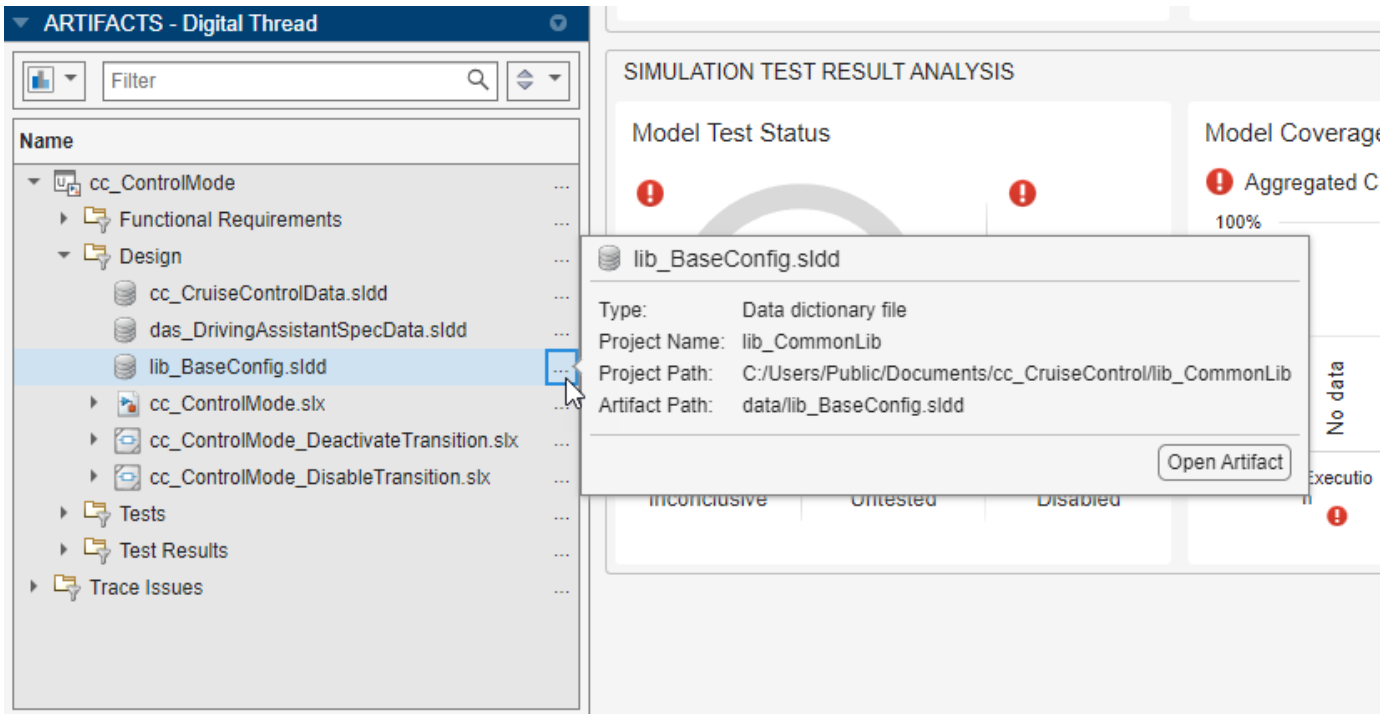
Analyze artifacts from referenced projects in the dashboards

In R2023a, the dashboards can analyze artifacts in referenced projects. The dashboards automatically analyze artifacts across project references and trace artifacts from a referenced project to the current project.

The **Project** panel now shows units and components in the current project and in referenced projects. Artifacts from referenced projects appear in the **Artifacts** panel for the relevant unit or

component that they trace to in the project. The dashboard includes these referenced project artifacts in the metric results.

To see the name and path of the project that contains an artifact, point to the artifact and view the tooltip.



You can also see artifacts from referenced projects in a trace view. For more information, see “Explore Traceability Information for Units and Components”.

Trace requirement links to MATLAB code files in the Model Testing Dashboard

In R2023a, the Model Testing Dashboard traces links between requirements and MATLAB code files. Previously, the dashboard did not support requirement links to MATLAB code files and the links did not contribute to the metric results.

For information on how to link requirements to MATLAB code, see “Requirements Traceability for MATLAB Code” (Requirements Toolbox). If you expect a requirement link to trace to a unit in the dashboard and it does not, see “Resolve Missing Artifacts, Links, and Results”.

Troubleshoot artifact issues with Artifact Issues tab in the dashboards

In R2023a, you can use the **Artifact Issues** tab to identify and fix artifact issues in your project and troubleshoot warnings and errors. Artifact issues no longer appear in the **Diagnostics** panel. The issues now appear in the **Artifact Issues** tab and persist between MATLAB sessions.

To view artifact issues in the current project, open the **Artifact Issues** tab by clicking the **Artifact Issues** button in the dashboard toolstrip. In the **Artifact Issues** tab, a table displays detailed information about each artifact issue in the project.

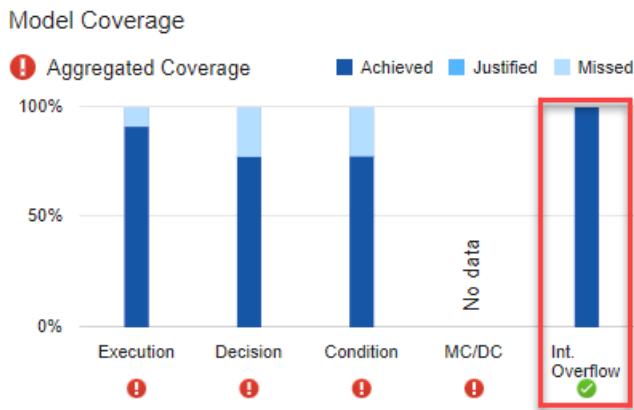
Severity	Message	Source	Message ID
Warning	Model Loading and Closing Callbacks have been deactivated while loading 'cc_CruiseControl' for analysis.	cc_CruiseControl	alm:simulink_trace_plugins:ModelCallbacksDeacti

Alternatively, you can use the new function `getArtifactIssues` on a metric engine object to return a list of the artifact issues the dashboard detects in the project.

For more information, see “View Artifact Issues in Project” and `getArtifactIssues`.

View integer overflow coverage in the Model Testing Dashboard

In R2023a, the **Model Testing Dashboard** shows integer overflow coverage, **Int. Overflow**, in the **Model Coverage** section of the dashboard.



You can also programmatically collect integer overflow coverage with the metric ID `slcomp.mt.CoverageBreakdown`. The metric value returns a structure array with a field `OverflowSaturation` for the integer overflow coverage. For example:

```
metric_engine = metric.Engine;
execute(metric_engine, "slcomp.mt.CoverageBreakdown");
res = getMetrics(metric_engine, "slcomp.mt.CoverageBreakdown");
overflowCoverage = res(1).Value.OverflowSaturation
```

`overflowCoverage =`

struct with fields:

```
Achieved: 100
Justified: 0
Missed: 0
AchievedOrJustified: 100
```

For more information, see “Model Coverage Breakdown”.

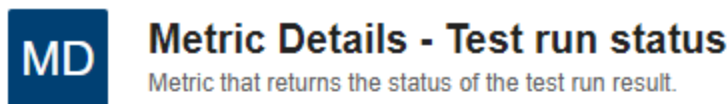
View individual test iterations in the metrics and dashboards

Previously, if a test case included multiple iterations, the dashboard and metric results reflected the status of the whole test case and did not show individual iteration results. In R2023a, the dashboards and metrics can show the results for individual tests. A *test* can be either:

- A test iteration
- A test case without iterations

For example, if the **Untested** widget in the **Model Testing Dashboard** shows 6 untested unit tests, these tests can be:

- 6 untested test iterations
- 6 untested test cases without iterations
- A combination of test iterations and test cases without iterations

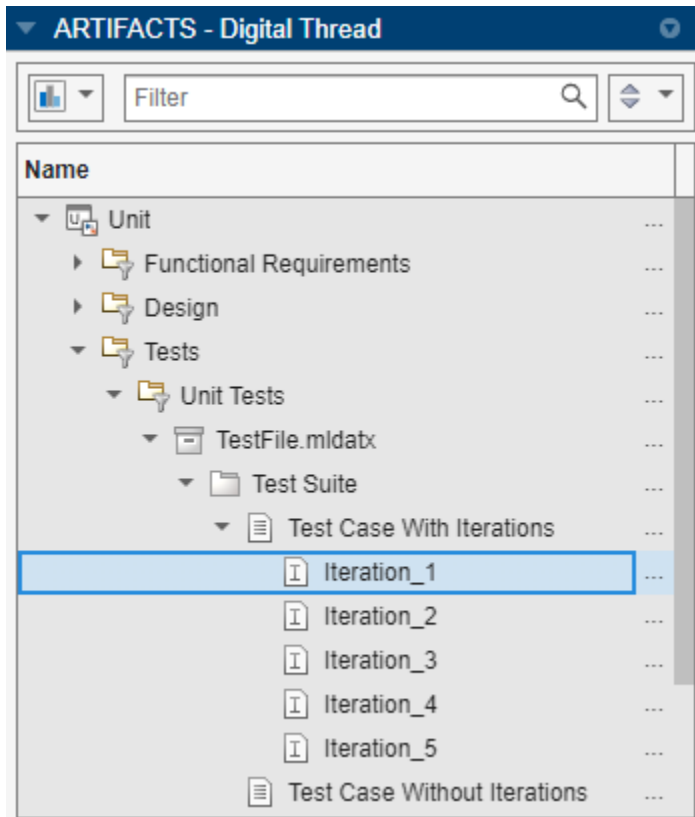


Artifact	Source	Status
Test Case With Iterations > Iteration_2	TestFile.mldatx	Untested
Test Case With Iterations > Iteration_1	TestFile.mldatx	Untested
Test Case With Iterations > Iteration_3	TestFile.mldatx	Untested
Test Case With Iterations > Iteration_5	TestFile.mldatx	Untested
Test Case With Iterations > Iteration_4	TestFile.mldatx	Untested
Test Case Without Iterations	TestFile.mldatx	Untested

In the **Artifacts** panel, in the **Tests** and **Test Result** folders, the dashboard now shows test iterations in the test file hierarchy.

Additionally, in the **Metric Details**, you can point to an artifact to view a tooltip with the:

- Location of the artifact in the project hierarchy
- Artifact type
- Project name
- Project path
- Artifact path relative to the project root



For an example, see “Explore Status and Quality of Testing Activities Using Model Testing Dashboard”.

Faster response time for finding clones in a model

Starting in R2023a, the clone detection algorithm significantly shortens the amount of time a `findClones` function takes to search for clones across the model. This is particularly visible for larger models. The optimized clone detection algorithm performs better with both the Clone Detector app and `Simulink.CloneDetection.findClones` API.

For more information, see “Find Clones Across the Model”.

Load a specific justifications file when running Model Advisor checks

Previously, the Model Advisor only loaded justifications for a model if the justifications file was named `modelName_justifications.json` and was in the current working directory. In R2023a, you can load a justifications file with any name and from any directory.

The first time that you justify a check for a model, the Model Advisor prompts you with a Save As dialog that allows you to save the justifications file with your specified file name and in your specified file directory.

You can load a justifications file for a model by using either of these approaches:

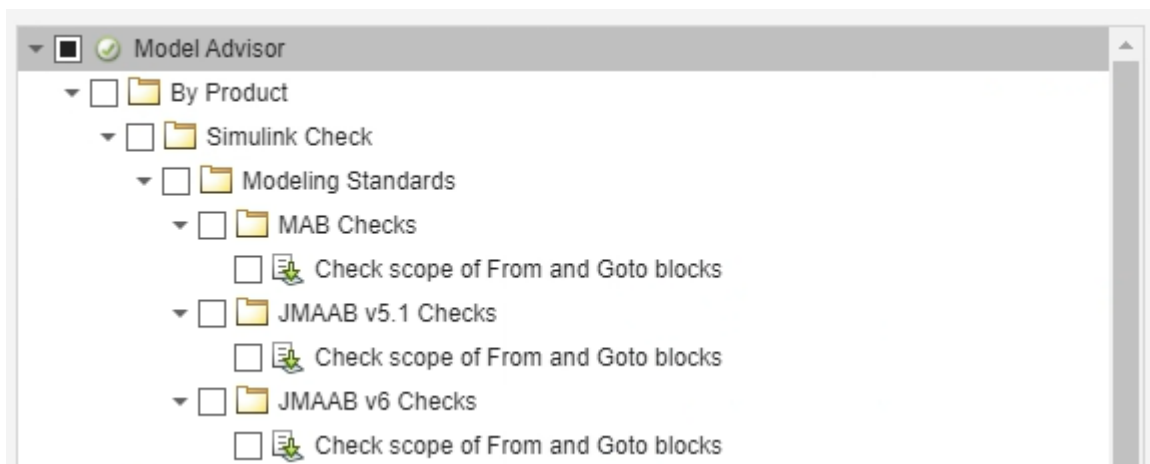
- In the Model Advisor toolstrip, click **Open > Load Justifications File** and select a justifications file.

- When you call `ModelAdvisor.run`, use the `Justifications` argument to specify the filename or path to a justifications file.

For more information, see “Justify Model Advisor Violations from Check Analysis” and `ModelAdvisor.run`.

JMAAB 6.0 Support: New and updated Model Advisor checks to enable compliance with JMAAB 6.0 modeling style guidelines

In R2023a, the Model Advisor is updated to support the new JMAAB 6.0 modeling guidelines. While a new folder containing JMAAB 6.0 checks is added in the Model Advisor, MAB 5.0 and JMAAB 5.1 checks continue to be available as before.



The JMAAB 6.0 checks are derived from the JMAAB v6.0 modeling guidelines composed by the Japan MATLAB® Automotive Advisory Board.

This table lists the new JMAAB 6.0 checks.

Model Advisor Check	Check ID
“Check bus and enumeration data type names”	<code>mathworks.jmaab_v6.jc_0900</code>
“Check length of bus and enumeration data type names”	<code>mathworks.jmaab_v6.jc_0901</code>
“Check arrowhead size of transition lines”	<code>mathworks.jmaab_v6.jc_0902</code>
“Check for prohibited overlapping or intersecting blocks and signal lines”	<code>mathworks.jmaab_v6.jc_0903</code>
“Check for prohibited overlapping of states and transition lines in Stateflow charts”	<code>mathworks.jmaab_v6.jc_0904</code>
“Check data names in MATLAB Functions”	<code>mathworks.jmaab_v6.jc_0905</code>
“Check the length of data names in MATLAB Functions”	<code>mathworks.jmaab_v6.jc_0906</code>
“Check size of junctions”	<code>mathworks.jmaab_v6.jc_0907</code>
“Check description of execution statements”	<code>mathworks.jmaab_v6.mp_0007</code>

Model Advisor Check	Check ID
"Check for spaces between function or variable names and left parenthesis symbol"	mathworks.jmaab_v6.mp_0008
"Check for operator precedence"	mathworks.jmaab_v6.mp_0010
"Check spaces in expressions"	mathworks.jmaab_v6.mp_0011
"Check description of conditional expressions"	mathworks.jmaab_v6.mp_0022
"Check relational operators usage"	mathworks.jmaab_v6.mp_0023
"Check function headers"	mathworks.jmaab_v6.mp_0032
"Check number of lines of functions"	mathworks.jmaab_v6.mp_0034
"Check for utilization of the return value of functions"	mathworks.jmaab_v6.mp_0040
"Check array indices"	mathworks.jmaab_v6.mp_0046
"Check for usage of nonempty statements"	mathworks.jmaab_v6.mp_0047
"Check folder names"	mathworks.jmaab_v6.ar_0002
"Check signal line connections"	mathworks.jmaab_v6.db_0032
"Check position of signal labels"	mathworks.jmaab_v6.db_0097
"Check definition of Stateflow data"	mathworks.jmaab_v6.db_0125
"Check for MATLAB expressions in Stateflow charts"	mathworks.jmaab_v6.db_0127
"Check for Stateflow transition appearance"	mathworks.jmaab_v6.db_0129
"Check usable characters for parameter names"	mathworks.jmaab_v6.jc_0232
"Check usage of floating-point expressions in Stateflow charts"	mathworks.jmaab_v6.jc_0481
"Check usage of Discrete-Time Integrator block"	mathworks.jmaab_v6.jc_0627
"Check settings for data ports in Multiport Switch blocks"	mathworks.jmaab_v6.jc_0630
"Check type setting by data objects"	mathworks.jmaab_v6.jc_0644
"Check Output data type of operation blocks"	mathworks.jmaab_v6.jc_0651
"Check condition actions and transition actions in Stateflow"	mathworks.jmaab_v6.jc_0753
"Check placement of Label String in Transitions"	mathworks.jmaab_v6.jc_0770
"Check for usage of events and broadcasting events in Stateflow charts"	mathworks.jmaab_v6.jm_0012
"Check scope of From and Goto blocks"	mathworks.jmaab_v6.na_0011
"Check for missing ports in Variant Subsystems"	mathworks.jmaab_v6.na_0020

The remaining checks are included as part of JMAAB 6.0, with no change to the check behavior.

Analyze Simulink functions using Model Slicer

You can now analyze models that contain Simulink functions using Model Slicer. You can also visualize the input or output dependency of a function caller block by adding the block as a starting point. For more information, see “Analyze Models Containing Simulink Functions Using Model Slicer”.

Simulink Check features available in Simulink Online

Simulink Check is now available in Simulink Online. You can access most of the features through your web browser.

Limitations

- Metrics Dashboard is not supported in Simulink Online.
- Model Advisor parallel run is not supported in Simulink Online.

Functionality being removed or changed

New metric IDs for test status and coverage metrics in the Model Testing Dashboard

Behavior change

Starting in R2023a, there are new metric IDs associated with the test status and coverage metrics in the Model Testing Dashboard. If you use the previous metric IDs, update your code to use the new metric IDs.

Previous metric ID	New metric ID	Metric updates
TestCaseStatus	slcomp.mt.TestStatus	The metric has a new metric ID, but has the same functionality. For more information, see “Model Test Status”.
TestCaseStatusDistribution	slcomp.mt.TestStatusDistribution	The metric has a new metric ID, but has the same functionality. For more information, see “Model Test Status Distribution”.
ExecutionCoverageBreakdown	slcomp.mt.CoverageBreakdown	You can now use a single metric ID to collect the aggregated coverage results for a unit. The metric value returns results in a structure array that you can use to access the coverage results for each coverage type. For more information, see “Model Coverage Breakdown”.
DecisionCoverageBreakdown		
ConditionCoverageBreakdown		
MCDCCoverageBreakdown		
ExecutionCoverageFragment	slcomp.mt.CoverageFragment	You can now use a single metric ID to collect the coverage results for each model in a unit. The metric value returns results
DecisionCoverageFragment		

Previous metric ID	New metric ID	Metric updates
ConditionCoverageFragment		in a structure array that you can use to access the coverage results for each coverage type. For more information, see "Model Coverage Fragment".
MCDCCoverageFragment		

To return the metrics from the Model Testing Dashboard, use the `getAvailableMetricIds` function:

```
modelTestingMetrics = getAvailableMetricIds(metric_engine,...
App="DashboardApp",Dashboard="ModelUnitTesting");
```

In the new metric IDs, `slcomp` refers to Simulink components and `mt` refers to model testing. For more information, see "Model Testing Metrics".

Metric IDs removed for percentage results in the Model Testing Dashboard

Starting in R2023a, there are no longer individual metric IDs for the percentage results in the Model Testing Dashboard. Instead, access the percentage results directly from the associated distribution metric. The distribution metric value contains a structure array with a field `Ratios` that contains an integer vector for the percentage results.

Previous metric ID	Associated distribution metric ID	Description
TestCaseStatusPercentage	<code>slcomp.mt.TestStatusDistribution</code>	<p>The <code>Ratios</code> field of <code>slcomp.mt.TestStatusDistribution</code> returns:</p> <ul style="list-style-type: none"> • <code>Ratios(1)</code> — Percentage of model tests that failed. • <code>Ratios(2)</code> — Percentage of model tests that passed. • <code>Ratios(3)</code> — Percentage of model tests that are disabled. • <code>Ratios(4)</code> — Percentage of model tests that are untested. <p>For more information, see "Model Test Status Distribution".</p>

Previous metric ID	Associated distribution metric ID	Description
TestCaseWithRequirementPercentage	TestCaseWithRequirementDistribution	<p>The Ratios field of TestCaseWithRequirementDistribution returns:</p> <ul style="list-style-type: none"> • Ratios(1) — Percentage of model tests missing links to requirements. • Ratios(2) — Percentage of model tests with links to requirements. <p>For more information, see “Test Case with Requirement Distribution”.</p>
RequirementWithTestCasePercentage	RequirementWithTestCaseDistribution	<p>The Ratios field of RequirementWithTestCaseDistribution returns:</p> <ul style="list-style-type: none"> • Ratios(1) — Percentage of requirements missing links to model tests. • Ratios(2) — Percentage of requirements with links to model tests. <p>For more information, see “Requirement with Test Case Distribution”.</p>

For example, if your previous code was:

```
execute(metric_engine, ["TestCaseStatusPercentage", ...
"TestCaseWithRequirementPercentage", ...
"RequirementWithTestCasePercentage"]);
```

Update to this code:

```
execute(metric_engine, ["slcomp.mt.TestStatusDistribution", ...
"TestCaseWithRequirementDistribution", ...
"RequirementWithTestCaseDistribution"]);
```

When you get the metric results, the percentage results are in the integer vector `Ratios`.

Suppose that 14.29% of tests fail, 71.43% of tests pass, 14.29% of tests are disabled, and 0% of tests are untested. The metric `slcomp.mt.TestStatusDistribution` returns `Ratios` as an integer vector with the percentages in decimal form:

```
results = getMetrics(metric_engine, "slcomp.mt.TestStatusDistribution");
results.Value.Ratios
```

```
ans =
```

```
0.1429
0.7143
0.1429
0
```

For more information on the `Ratios` field, see the associated distribution metric:

- “Model Test Status Distribution” for the percentage of failed, passed, disabled, or untested model tests
- “Test Case with Requirement Distribution” for the percentage of tests that are missing links to requirements or have links to requirements
- “Requirement with Test Case Distribution” for the percentage of requirements that are missing links to tests or have links to tests

metric.Result objects no longer return the fields Type and ParentType

Starting in R2023a, `metric.Result` objects do not return the fields `Type` and `ParentType` for the properties `Artifacts` and `Scope`.

If you use the `Type` or `ParentType` fields from the `Artifacts` and `Scope` properties of `metric.Result` objects, update your code to remove references to those fields.

Check hisl_0311: "Check safety-related diagnostic settings for Stateflow" no longer checks for the use of machine-parented data

Behavior change

Starting in R2023a, Model Advisor check "Check safety-related diagnostic settings for Stateflow" (`mathworks.hism.hisl_0311`) no longer checks that the configuration parameter **Use of machine-parented data instead of Data Store Memory** is set to `none` or `warning` because the configuration parameter has been removed.

Note that the configuration parameter was removed because Stateflow® charts no longer support machine-parented data. You can check for machine-parented data and use the Upgrade Advisor to convert machine-parented data to chart-parented data store memory. For more information, see “Consult the Upgrade Advisor” and “Check for machine-parented data”.

R2022b

Version: 6.1

New Features

Bug Fixes

Compatibility Considerations

CI/CD Automation for Simulink Check (October 2022; Version 22.2.0)

The support package CI/CD Automation for Simulink Check now supports R2022b Update 1 and later updates. The support package provides tools to help you integrate your model-based process into a Continuous Integration and Continuous Deployment (CI/CD) system.

The support package provides:

- A customizable process modeling system to define your build and verification process
- A build system that can automatically generate and efficiently execute a process in your CI system
- The **Process Advisor** app for deploying and automating your prequalification process
- Integration with common CI systems

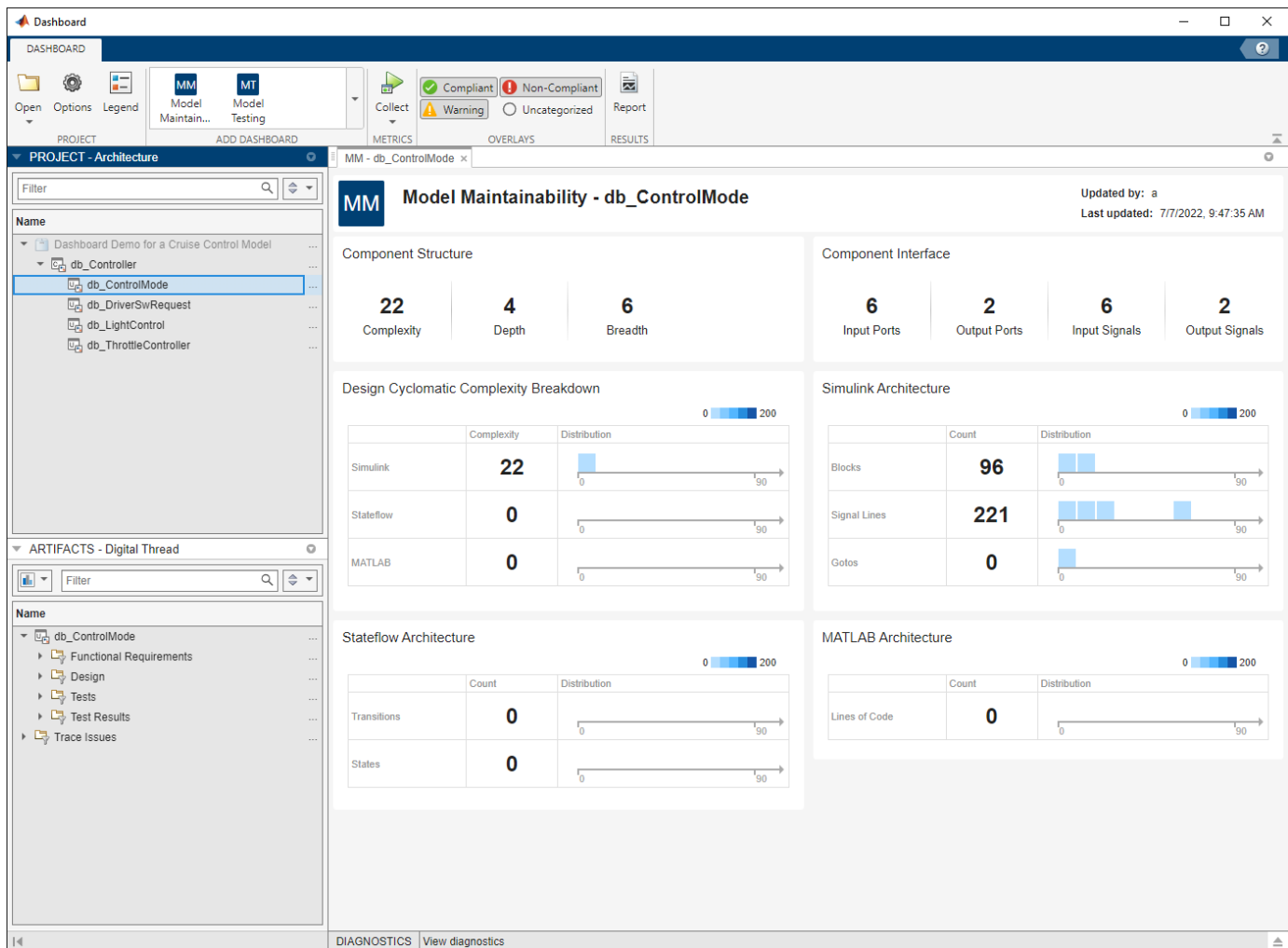
You can use the support package to help you set up a model-based design pipeline, reduce build time, reduce build failures, debug build failures, and deploy a consistent build and verification process.

For more information, see Run Tasks Locally and in CI. To download and install the support package, see CI/CD Automation for Simulink Check.

Model Maintainability Dashboard: Assess the complexity and maintainability of your design across the model development lifecycle

In R2022b, you can assess the size, architecture, and complexity of your design by using the metric data in the new **Model Maintainability Dashboard**. The metrics measure different aspects of model maintainability from model design artifacts like Simulink models, Stateflow charts, and MATLAB code. The model maintainability metrics help you determine if parts of a design are too complex and need to be refactored. A less complex design is easier to read, maintain, and test.

Use the **Model Maintainability Dashboard** to collect and explore maintainability metric data. The dashboard displays metric results related to the component structure, interface ports and signals, design cyclomatic complexity, and software architecture. For an example of how to use the **Model Maintainability Dashboard**, see Monitor the Complexity of Your Design Using the Model Maintainability Dashboard.



You can also collect the metrics programmatically by using the functions associated with the `metric.Engine` object. This is the same metric API used to programmatically collect metrics for the **Model Testing Dashboard**. When you use it to programmatically collect metrics for the **Model Maintainability Dashboard**, specify the Dashboard argument as "ModelMaintainability". For more information, see [Collect Model Maintainability Metrics Programmatically](#).

View external MATLAB code associated with units and components in the Model Maintainability Dashboard

In R2022b, the **Model Maintainability Dashboard** displays external MATLAB code associated with the units and components in your project. The dashboard displays external MATLAB code such as MATLAB functions, methods, and classes stored in MATLAB files. The code files appear in the **Design** folder of the **Artifacts** panel.

For example, if you have a unit that uses an Interpreted MATLAB Function block to call the function saved in the file `myExternalFunction.m`, the file appears in the **Artifacts** panel in the **Design** folder when you select your unit in the **Project** panel.

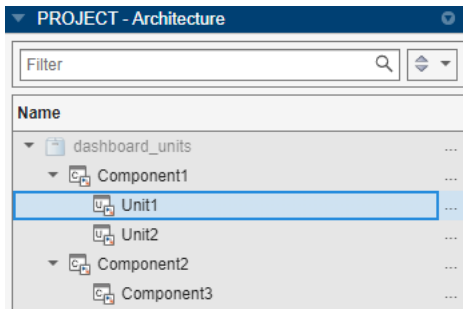
If you expect external MATLAB code to appear in the dashboard and it does not, see [External MATLAB Code Missing from Artifacts Panel](#).

Navigation enhancements for the Model Testing Dashboard and new Model Maintainability Dashboard

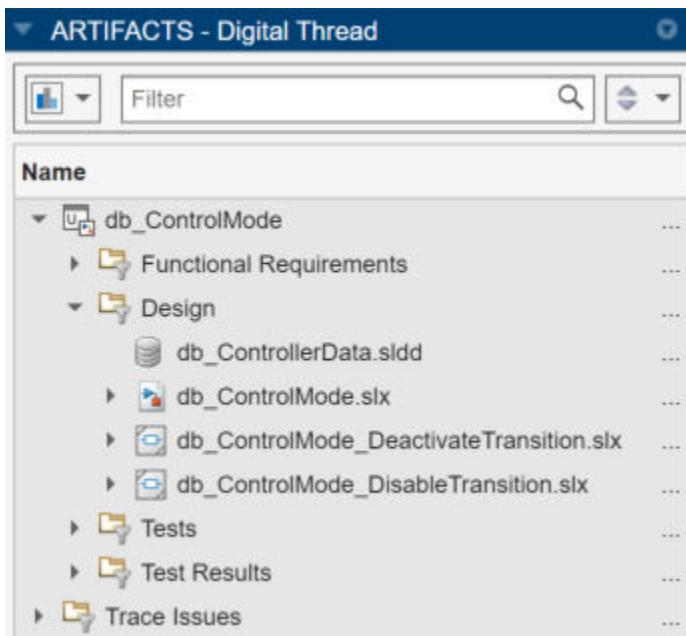
Previously, in the **Model Testing Dashboard**, you used the **Artifacts** panel both to select which unit to open the dashboard for and to explore the artifacts that traced to the units and components in the project.

In R2022b, the dashboard selection and artifact exploration are in separate panels:

- The **Project** panel shows the architecture of the software units and components in the current project. The dashboard displays metric results for the unit or component you select in the **Project** panel. If a dashboard is not available for a specific unit or component, the name of the unit or component appears dimmed.



- The **Artifacts** panel shows the **Functional Requirements**, **Design**, **Tests**, and **Test Results** folders, which contain the artifacts the dashboard traced to the current unit or component selected in the **Project** panel. For example, the **Design** folder for a unit might show the data dictionary, model, and subsystem references associated with the unit.

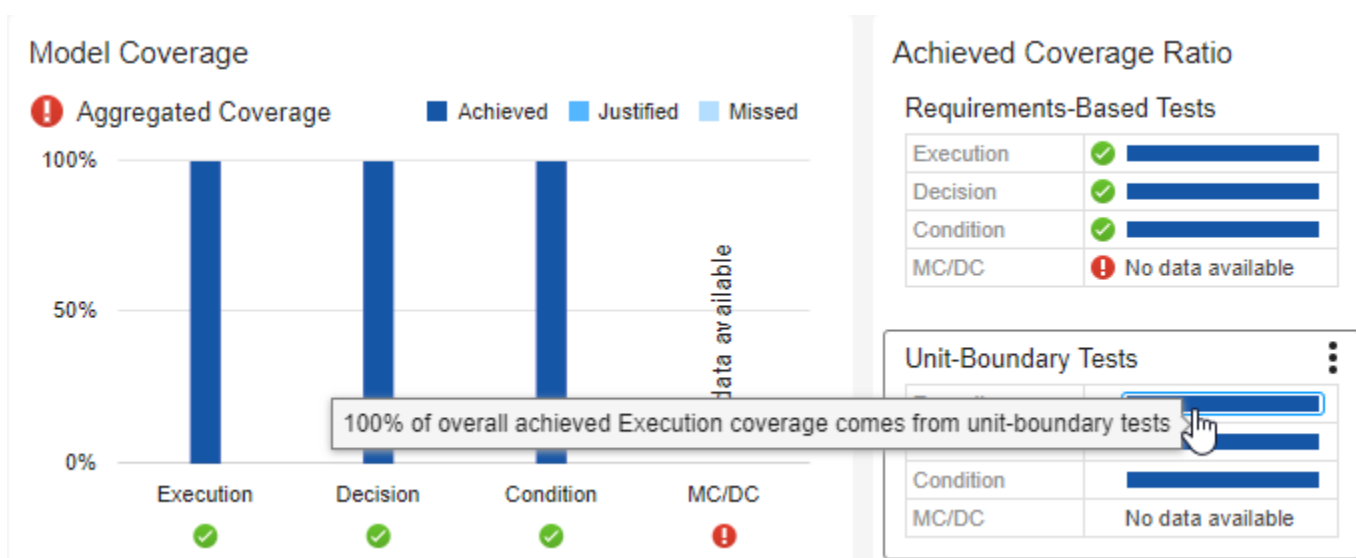


For more information, see [Monitor the Complexity of Your Design Using the Model Maintainability Dashboard](#) and [Explore Status and Quality of Testing Activities Using Model Testing Dashboard](#).

Identify the sources of overall achieved coverage in the Model Testing Dashboard

In R2022b, identify the types of tests that contribute to your overall achieved coverage in the **Model Testing Dashboard**. The dashboard identifies the percentage of the overall achieved coverage coming from requirements-based tests or unit-boundary tests, including execution, decision, condition, and modified condition / decision coverage. Requirements-based tests link to at least one requirement. Unit-boundary tests test the whole unit, not just lower-level subsystems.

Industry-recognized software development standards recommend using requirements-based, unit-boundary tests to confirm the completeness of coverage. Verify the tests that contribute to your overall achieved coverage with the dashboard metric data. In the **Model Testing Dashboard**, in the **Simulation Test Result Analysis** section, there is a new subsection, **Achieved Coverage Ratio**, with two new widgets: **Requirements-Based Tests** and **Unit-Boundary Tests**.



Additionally, you can use the functions associated with the `metric.Engine` object to programmatically collect the achieved coverage ratio metrics.

For more information on the new metrics for overall achieved coverage from requirements-based and unit-based tests, see:

- Metrics for Requirements-Based Tests for the Unit
- Metrics for Requirements-Based Tests for Each Model in the Unit
- Metrics for Unit-Boundary Tests for the Unit
- Metrics for Unit-Boundary Tests for Each Model in the Unit

For more information, see [Monitor Low-Level Test Results in the Model Testing Dashboard](#).

View internal and external test harnesses in the Model Testing Dashboard

In R2022b, the **Model Testing Dashboard** displays internal and external test harnesses in the **Artifacts** panel. Previously, the dashboard showed only externally stored test harnesses.

To view the test harnesses associated with a unit, go to the **Artifacts** panel and expand the folders **Tests > Test Harnesses**.

If you expect a test harness to appear in the dashboard and it does not, see [Resolve Missing Artifacts, Links, and Results in the Model Testing Dashboard](#).

Generated report opens automatically for the Model Testing Dashboard and Model Maintainability Dashboard

When you generate a report for your metric results, the generated report now opens automatically.

You can specify whether the report opens when you generate a report by using one of these methods:

- In the **Model Testing Dashboard** or **Model Maintainability Dashboard**, click the **Report** button on the toolstrip. In the Create Metric Result Report dialog box, in the **Output Options** section, select or clear the **Launch Report** check box.
- Use the `generateReport` function and specify the `LaunchReport` argument as `true` or `false`.

High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines

In R2022b, you can use these high-integrity modeling checks:

Model Advisor Check	Corresponding Modeling Guideline	Description of the change
Check for root Inports with missing properties	hisl_0024: Inport interface definition	Support extended for System Composer™.
Check for root Inports with missing range definitions	hisl_0025: Design min/max specification of input interfaces	Support extended for System Composer.
Check for root Outports with missing range definitions	hisl_0026: Design min/max specification of output interfaces	Support extended for System Composer.
Check for root Outports with missing properties	hisl_0077: Outport interface definition	New check for Simulink and System Composer.

For more information, see [Model Checks for High Integrity Systems Modeling](#).

Use older Model Advisor configuration files in newer versions of MATLAB

From R2022b, you can upgrade an older Model Advisor customization configuration file to the latest version of MATLAB. This enables you to view and use newly introduced or updated input parameters and check IDs. You can also delete checks that are no longer supported.

When you load an older configuration file that contains checks which are incompatible with the current release, you will get a dialog box asking whether you want to automatically fix issues in the configuration. To automatically fix issues and produce a validation summary, click **Yes**. To open the Model Configuration Editor with check issues highlighted, click **No**. You can then fix issues and validate the configuration from within the Model Advisor Configuration Editor. You can fix issues for each individual check or globally fix issues by clicking the **Validate** button. For more information, see Upgrade Incompatible Checks in Model Advisor Configuration Files.

Identify clones in a model during edit time

Simulink Check now identifies clones in models from the linked library file during edit time. This edit time check detects and highlights clones of Simulink blocks that can help you identify clone patterns earlier in the model design process.

To enable this check, in the **Modeling** tab, in the **Evaluate & Manage** section, click **Model Advisor > Configuration Editor**. Under the **Simulink Check** product, select the **Identify clones from linked library file** check box and click **Apply**.

You can also use Model Advisor to refactor the identified clones. Under the **Simulink Check** product in Model Advisor, select the **Identify clones from linked library file** check box and click the **Run** button from the Model Advisor toolstrip. The Model Advisor Report displays the identified clones. Click the **Fix** button to refactor the model.

For more information on edit-time checking, see Check Model Compliance Using Edit-Time Checking.

Identify Bus Selector and Bus Creator blocks during edit time

Simulink Check now detects Bus Selector and Bus Creator blocks in your model. To simplify your model, it is recommended to use In Bus Element and Out Bus Element blocks instead of Bus Selector blocks for inputs and Bus Creator blocks for outputs. For more information, see Simplify Subsystem and Model Interfaces with Bus Element Ports.

In the **Modeling** tab, in the **Evaluate & Manage** section, click **Model Advisor > Configuration Editor**. Under the **Simulink Check** product, select the **Refactor to simplify bus element blocks** check box and click **Apply**. This enables the edit time check to identify Bus Selector and Bus Creator blocks.

You can also use Model Advisor to refactor the identified Bus Selector and Bus Creator blocks. Under the **Simulink Check** product in Model Advisor, select the **Refactor to simplify bus element blocks** check box and click the **Run** button from the Model Advisor toolstrip. The Model Advisor Report displays the identified candidates. Click the **Fix** button to refactor the model.

For more information on edit-time checking, see Check Model Compliance Using Edit-Time Checking.

Enhancements to edit-time checking for numeric efficiency issues

You can now identify numeric efficiency issues earlier in the model design process by using edit-time checking. In R2022b, when you use edit-time checking, you can view violations of the Check usage of 'long long' data type (Embedded Coder) Model Advisor check. For more information, see the check documentation.

Add bus elements as starting points using Model Slicer

In R2022b, for bus signals, along with adding an entire bus hierarchy as a starting point, you can now select individual bus elements from the **Select Bus Element(s)** user interface using the Model Slicer context menu. You can also use the `addStartingPoint` function to specify a bus element path as an input.

To add a bus as a starting point:

- 1 Open the model in Model Slicer
- 2 Right-click the bus signal and select **Model Slicer > Add Bus as Starting Point**

To add bus elements as starting points:

- 1 Open the model in Model Slicer
- 2 Right-click the bus signal and select **Model Slicer > Select Bus Elements as Starting Points**
- 3 From the **Select Bus Element(s)** dialog box, select the elements that you want to add as starting points and then click **Add Starting Point**
- 4 Close the **Select Bus Element(s)** window

The selected bus and bus elements show up under **Starting Points** in the Model Slicer configuration window.

For more information, see `addStartingPoint` and `removeStartingPoint`.

Functionality being removed or changed

Metrics Dashboard user interface, `metricdashboard` function, and `slmetric` package API will be removed

Warns

The Metrics Dashboard user interface, `metricdashboard` function, `slmetric` package API, and corresponding customizations will be removed in a future release. In R2022b, use the **Model Maintainability Dashboard** and `metric` package API to collect size, architecture, and complexity metrics.

The Metrics Dashboard:

- Can only run on individual models
- Requires a computationally intensive, compile-based analysis for many metrics

The **Model Maintainability Dashboard** and `metric` package API:

- Runs on individual models in a project, but can also aggregate metrics across software units in software components
- Analyzes and traces dependencies between project files such as Simulink models, MATLAB code, and Stateflow objects
- Identifies outdated metric results

For the **Model Maintainability Dashboard** to analyze a model, the model needs to be stored in a project. MATLAB projects can help you organize models and interact with source control. For information on how to store your models in a project, see [Create Projects](#). For information how to use

the **Model Maintainability Dashboard** to collect size, architecture, and complexity metrics, see [Monitor the Complexity of Your Design Using the Model Maintainability Dashboard and Collect Model Maintainability Metrics Programmatically](#).

Replace instances of "RequirementsBasedModelUnitTesting" with "ModelUnitTesting"

Previously, the Dashboard argument for the `getAvailableMetricIds` function accepted the values:

- "ModelUnitTesting" — Returned only the model testing metric identifiers that are not associated with requirements metrics.
- "RequirementsBasedModelUnitTesting" — Returned each of the model testing metric identifiers, including requirements metrics.

Now the Dashboard argument for the function `getAvailableMetricIds` accepts these values:

- "ModelUnitTesting" — Return the metric identifiers associated with the **Model Testing Dashboard** for your project.

The function `getAvailableMetricIds` uses your project options to determine which model testing metrics to return. `getAvailableMetricIds` can either return each of the model testing metrics, including requirements metrics, or return only the model testing metrics that are not associated with requirements metrics. To change your Project Options, open the **Model Testing Dashboard** and click **Options** in the toolstrip. In the **Layout** section, select or clear **Hide requirements metrics** and click **Apply**. For more information, see [Hide Requirements Metrics in the Model Testing Dashboard](#) and in [API Results](#).

- "ModelMaintainability" — Return the metric identifiers associated with the **Model Maintainability Dashboard**.

Functionality	Use This Instead
When you used the syntax <code>getAvailableMetricIds(metric_engine, App="DashboardApp", Dashboard="RequirementsBasedModelUnitTesting");</code> , the function returned each of the metrics used by the Model Testing Dashboard for the metric engine object <code>metric_engine</code> .	If you want to return metrics from the Model Testing Dashboard , you must update your code: Specify the Dashboard argument as "ModelUnitTesting". <pre>testingMetrics = getAvailableMetricIds(metric_engine, App="DashboardApp", Dashboard="ModelUnitTesting");</pre>

For more information, see `getAvailableMetricIds`.

Check hisf_0007: Usage of Junction Conditions(maintaining mutual exclusions) will be removed

The hisf_0007: Usage of Junction Conditions(maintaining mutual exclusions) will be removed as it is covered under hisl_0101.

R2022a

Version: 6.0

New Features

Bug Fixes


Compatibility Considerations

Toolbar-based UI for Model Advisor

The Model Advisor user interface now includes simplified toolbar with new features.

- **Filter Checks** - Filters the checks based on their respective result statuses, such as **Failed**, **Passed**, **Justified**.
- **Justify** - Justify the violations. Justifications allow you to add a rationale for violations observed during Model Advisor analysis. For more information, see **Justify Violated Blocks** from the Model Advisor Check Analysis.
- **Fix** - Fixes the violations by setting the violated parameter values to recommended values.
- **Reports** - Export Model Advisor analysis reports in HTML, PDF, and DOCX formats.
- **Manage Configurations** - Create, load, restore, and associate configurations in simple workflows.

This table describes the changes to the menu items in the Model Advisor:

Previous UI Element	New UI Element	Description of the Change
Run Selected Checks	Run Checks	No change in functionality. Use this button to run selected checks.
Settings > (options)	Open > (options)	Some older options are no longer available. Use this button to open or customize Model Advisor Configuration Editor .
Generate Report...	Report > (options)	No change in functionality. Use this button to generate the Model Advisor analysis report in HTML format. Use the drop-down option to select PDF or WORD .
Configure Hidden input parameters	Configure input parameters in Model Advisor Configuration Editor	Configure hidden input parameters was a hyperlink in the check window. In R2022a, you can now click the Configure input parameters in Model Advisor Configuration Editor icon ()
Edit > Send Check IDs to Workspace	Right-click on any check, and select Send Check IDs to Workspace .	Use this option to send check IDs to workspace.
Edit > Send Instance IDs to Workspace	Right-click on any check, and select Send Instance IDs to Workspace .	Use this option to send instance IDs to workspace.







This table describes the navigation options removed from Model Advisor:

Navigation Options Removed
Settings > Treat as referenced model
Settings > Preferences
Edit > Reset
Switch to Model Advisor Dashboard
Run Checks in Background
Highlighting check results
Highlight exclusion results
Highlighting

For more information, see [Run Model Advisor Checks and Review Results](#).

Model Advisor Check Result Statuses

Model Advisor analysis now have new statuses to better understand the check results. This table describes the existing and new check statuses:

Check Result Status	Icon	Description
Passed		Model does not have any violations for the given check or checks.
Failed		Check has identified severe violations.
Warning		Check has identified violations.
Justified		Check violations are justified.
Not Run		Check not selected for Model Advisor analysis.
Incomplete		Check analysis is incomplete or check execution has resulted in exceptions.

For more information, see [Run Model Advisor Checks and Review Results](#).

Author Model Advisor checks that run at edit-time

Starting in R2022a, you can author Model Advisor checks that run during edit-time. Because edit-time checks appear in the model canvas while you edit your model, they can help you catch issues earlier in the model design process. You can author edit-time checks that detect and highlight issues on blocks and signals.

To create a custom edit-time check, create a MATLAB class that derives from the `ModelAdvisor.EdittimeCheck` class. For an example, see [Define Edit-Time Checks to Comply with Conditions that You Specify with the Model Advisor](#).

If you have a System Composer license, you can author custom edit-time checks that run on architecture models. For an example, see [Define Custom Edit-Time Checks that Fix Issues in Architecture Models](#).

For more on the workflow for authoring custom checks, see [Define Custom Model Advisor Checks](#).

Model Advisor disables edit-time checks with high execution times

In R2022a, the Model Advisor now disables custom edit-time checks with long execution times. The Model Advisor automatically disables custom edit-time checks if, in the current MATLAB session, the execution time of the check exceeded 500 milliseconds in at least three different models. This feature helps prevent custom edit-time checks from negatively impacting performance as you edit your model.

If the Model Advisor disables a custom edit-time check, it displays a warning on the Simulink canvas. You can re-enable the edit-time check by either:

- Clicking the hyperlink text in the warning.
- Passing the check identifier, *checkID*, to the function `edittime.enableCheck`:

```
edittime.enableCheck(checkID)
```

To prevent a custom edit-time check from being disabled, author the check so that the check executes in less than 500 milliseconds on your models.

For more information, see [Define Edit-Time Checks to Comply with Conditions that You Specify with the Model Advisor](#).

Create help for custom Model Advisor checks

You can define help files for your custom Model Advisor checks to make the checks easier to use. Custom help files allow you to verify the check capabilities and avoid potential warnings in the model. You can point the custom check help to a PDF or an HTML page of your choice. To link your custom documentation:

- 1 Open the `sl_customization.m` file.
- 2 Use `setHelp()` on the check or group object created in the `sl_customization.m` file.

```
setHelp('format','webpage','path','custom_path');
```

The supported name-value arguments are:

Format - "webpage", "pdf"

Path - Path of the user-defined help page or document

Example:

```
checkObj = ModelAdvisor.Check('SimplePassFailCheck');  
checkObj.setHelp('format','webpage','path','custom_path');
```

- 3 Close the `sl_customization.m` file.
- 4 Refresh the customizations by entering:

```
Advisor.Manager.refresh_customizations
```

To view the custom help, right-click the custom checks or the folder and click **What's This?**.

For More information, see [setHelp | setHelp | Create Help for Custom Model Advisor Checks](#).

Justify Model Advisor check violations

You can now justify and hide the Model Advisor check violations from Model Advisor check report using the justifications workflow. To justify a violation, use either of the following options:

- For violations displayed post Model Advisor check analysis:
 - 1 From the check selector section, click on the violated check(s).
 - 2 Click **Justify** icon from the toolstrip.
 - 3 Enter the rationale for justification in the **Justifications** field on the **Result Inspector** tab.
 - 4 Click **Add Justification**.
- For edit-time violations:
 - 1 On the Simulink canvas, hover over a violated block.
 - 2 Click on the warning icon displayed above the violated block.

Violation summary is displayed along with the title of the violated check.
 - 3 Click **Suppress**. A description field to enter rationale is displayed.
 - 4 Enter rationale for the justification.
 - 5 Click **Add Comment**.

For More information, see [Justify Violated Blocks from the Model Advisor Check Analysis](#).

High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines

In R2022a, you can use these high-integrity modeling checks:

Model Advisor Check	Corresponding Modeling Guideline
Check for unreachable and dead code	hisl_0101: Avoid operations that result in dead logic to improve code compliance
Check for disabled and parameterized library links	hisl_0075: Usage of library links

For more information, see [Model Checks for High Integrity Systems Modeling](#).

Compatibility Considerations

These checks were removed in R2022a:

Model Advisor Check	Corresponding Modeling Guideline
Check usage of bitwise operations in Stateflow charts	hisf_0003: Usage of bitwise operations
Check for Strong Data Typing with Simulink I/O	hisf_0009: Strong data typing (Simulink and Stateflow boundary)

MAB Checks: Improve quality and compliance with guidelines

Compatibility Considerations

These checks were removed in R2022a:

Model Advisor Check	Corresponding Modeling Guideline
Check for Strong Data Typing with Simulink I/O	db_0122: Stateflow and Simulink interface signals and parameters

Enable edit-time checking for models by using new configuration parameter

In R2022a, you can use the new configuration parameter `ShowAdvisorChecksEditTime` to enable edit-time checking for a model.

In the **Modeling** tab, in the **Evaluate & Manage** section, click **Model Advisor > Edit-Time Checks**. In the Model Advisor pane of the Configuration Parameter dialog, select the check box for **Edit-Time Checks** and click **Apply**.

Alternatively, you can enable or disable edit-time checking for your model by using `edittime.setAdvisorChecking`.

For more information, see [Show Model Advisor edit-time checks](#).

Associate a Model Advisor configuration file with a model

In R2022a, you can associate a Model Advisor configuration file with your model. In previous releases, you set a default configuration for all models. You can now specify a different Model Advisor configuration file for each model. For Model Advisor configuration files created in a previous release, open and re-save the configuration file in the R2022a Model Advisor Configuration Editor before associating the file with a model.

For more information, see [Load and Associate a Custom Configuration with a Model](#).

Enable artifact tracing to track changes and project artifacts in the Model Testing Dashboard

In R2022a, you can enable artifact tracing directly in the settings for your project. This feature allows you to set up your project to track changes to project artifacts, such as test results from Simulink Test, to detect outdated metric results.

By default, the Model Testing Dashboard prompts you to enable artifact tracing the first time you open a project in the dashboard. Click **Enable and Continue** to track tool outputs to detect outdated metric results.

You can also enable artifact tracing from the Manage Project Startup and Shutdown dialog box. In your project, in the **Project** tab, click **Startup Shutdown**. In the Manage Project Startup and Shutdown dialog box, select **Track tool outputs to detect outdated results**.

For more information, see [Enable Artifact Tracing for the Project](#).

Include subsystem-level tests in the Model Testing Dashboard

In R2022a, the Model Testing Dashboard includes subsystem-level tests, such as tests on atomic subsystems, in the metric results. Previously, the dashboard metric results included only test cases that ran on the whole unit.

The dashboard metrics now include tests defined on:

- Atomic subsystems
- Atomic subsystem references
- Atomic Stateflow charts
- Atomic MATLAB Function blocks
- Referenced models

Note that the Model Testing Dashboard cannot calculate aggregated coverage for units that only have test results at the subsystem level. In order for the dashboard to display aggregated test coverage for a unit, the unit needs to have results from top-level tests executed by the model.

For more information, see [Include Subsystem-Level Test Results in the Model Testing Dashboard](#).

Trace System Composer architecture models in the Model Testing Dashboard

In R2022a, you can use the Model Testing Dashboard to specify System Composer architecture models as components. The architecture models, and the units that trace to the models, appear in a hierarchy in the **Artifacts** panel.

Supported architectures include System Composer architecture models, System Composer software architecture models, and AUTOSAR architectures.

To add a supported architecture to the Model Testing Dashboard, label the models as components in your project and configure the Model Testing Dashboard to recognize the labels. For more information, see [Specify Models as Components and Units](#).

View test harnesses in the Artifacts panel for each unit in the Model Testing Dashboard

In R2022a, the Model Testing Dashboard organizes tests in a new hierarchy in the **Artifacts** panel, with a folder called **Tests** that includes the subfolders **Unit Tests**, **Others**, and **Test Harnesses**. The subfolder **Test Harnesses** contains externally stored test harnesses that trace to the unit or unit subsystems.

You can open a test harness directly from the Model Testing Dashboard by expanding **Test Harnesses** and double-clicking the name of the test harness.

If your model already uses internal test harnesses, you can convert the internal test harnesses to an externally stored test harness. Navigate to the top of the main model and open Simulink Test. On the **Tests** tab, click **Manage Test Harnesses > Convert to External Harnesses**. Click **Yes** to convert the affected test harnesses.

For more information, see [Manage Requirements-Based Testing Artifacts for Analysis in the Model Testing Dashboard](#).

Hide requirements metrics in the Model Testing Dashboard

In R2022a, you can view the Model Testing Dashboard without the widgets associated with requirements metrics. If you do not use requirements-based testing, this new layout can help you focus on the dashboard widgets that contribute to your design goals. The new layout shows only the widgets for **Test Case Breakdown**, **Model Test Status**, and **Model Coverage**.

To use this layout, open the Project Options dialog box. In the **Project** section, click **Options**. In the **Layout** section of the Project Options dialog box, select **Hide requirements metrics** and click **Apply**.

The **Hide requirements metrics** setting is saved in the project meta information and is shared with everyone who uses the project. If you use this layout, the function `generateReport` generates a filtered report that shows only metric results that are not associated with requirements metrics.

For more information, see [Hide Requirements Metrics in the Model Testing Dashboard](#) and in [API Results](#).

Identify which artifacts contribute to metrics in the Model Testing Dashboard

In R2022a, the **Artifacts** panel includes new folders and subfolders that indicate which artifacts contribute to the metric results.

The **Artifacts** panel shows folders for each main artifact type: **Functional Requirements**, **Design**, **Tests**, and **Test Results**. In each folder, the artifacts that contribute to the metric results are in one subfolder and the artifacts that do not contribute to metric results are in a different subfolder. For example, in the folder **Tests**, the subfolder **Unit Tests** contains the test cases that the dashboard uses in the metrics for the unit. The subfolder **Others** contains the test cases that trace to the unit, but that the dashboard does not include in the metrics for the unit. For more information, see [Manage Requirements-Based Testing Artifacts for Analysis in the Model Testing Dashboard](#).

The folder **Untraced Artifacts** is now called **Trace Issues** and contains subfolders to help you to troubleshoot why the dashboard cannot trace the artifacts. The folder **Trace Issues** contains the subfolders **Unexpected Implementation Links**, **Unresolved and Unsupported Links**, **Untraced Tests**, and **Untraced Results**. Additionally, the dashboard diagnostic messages now include a hyperlink to the affected artifact file and a suggestion for how to address the issue. If you have errors or warnings in the **Diagnostics** pane, click the hyperlink to open the affected artifact and use the suggested action to address the issue. For more information, see [Fix Requirements-Based Testing Issues](#).

Detect changes to artifact traceability and metric results in the Model Testing Dashboard

In R2022a, the Model Testing Dashboard automatically performs initial artifact tracing and shows warning banners to help you detect changes to artifact traceability and metric results.

When you open an existing project in the Model Testing Dashboard, you no longer need to click **Trace Artifacts** or **Trace and Collect All**. The dashboard automatically traces artifacts in the project and, when you select a unit, the dashboard collects any uncollected metrics for the unit.

As you make changes to the artifact files in your project, the dashboard detects the changes and automatically traces the artifacts to refresh the data in the **Artifacts** panel.

Additionally, if artifacts in the project change after you collect the results, the dashboard shows a warning banner to indicate that the metric results are outdated. The **Stale** icon also appears on dashboard widgets that might show outdated results. Click the **Collect** button on the warning banner to re-collect the metric data and to update the stale widgets with data from the current artifacts.

For more information, see [Explore Status and Quality of Testing Activities Using the Model Testing Dashboard](#).

Navigate between project artifacts in the Model Testing Dashboard

In R2022a, you can more easily navigate between widgets and data shown in the Model Testing Dashboard. At the top of the dashboard, there is now a breadcrumb trail that you can use to navigate from **Metric Details** to the dashboard for the associated unit. When you click a widget in the dashboard, the dashboard opens the **Metric Details** and shows a breadcrumb trail from the **Metric Details (MD)** back to the **Model Testing (MT)** results in the unit dashboard. Click the name of the unit, for example **db_DriverSwRequest**, to return to the **Model Testing** dashboard.

Additionally, when you open the Requirements Editor, you can now navigate to the Model Testing Dashboard by using a button in the toolstrip of the Requirements Editor. In the Requirements Editor, in the **Analysis** section, click **Model Testing Dashboard**.

For more information, see [Fix Requirements-Based Testing Issues](#).

Refactor similar clones across the model

In R2022a, you can refactor similar clones anywhere across the model programmatically or by using **Detect clones across model** property in the Clone Detector app. In R2021b, you could refactor only exact clones across the model.

For more information, see [Find Clones Across the Model](#).

Find clones by using multiple external library files

Prior to R2022a, you could search for clones in Simulink models from only a single existing library file at a time. Starting in R2022a, you can find clones by using the multiple library files in the models.

To detect clones by using external library files in Clone Detector app, click **Settings > Match Patterns with Libraries** and select the library files. Alternatively, you can use a `Simulink.CloneDetection.Settings` object to add library files to find clones programmatically.

Note You can refactor exact clones only identified from library files.

For more information, see [Identify and Replace Clones in Model Libraries](#).

Clone Detection Exclusion Editor improvements

You can now use the Clone Detection Exclusion Editor to:

- Highlight excluded blocks by clicking **Block Full Path**.
- Edit exclusions.
- Edit exclusion reason (**Rationale**).
- Delete multiple rows at a time.

For more information, see Exclude Components from Clone Detection.

Inspect test cases generated in Simulink Design Verifier by using Model Slicer

You can now use Model Slicer to inspect test cases generated through test generation analysis. Model Slicer supports these test case objective statuses:

- **Objectives Satisfied**
- **Objectives Satisfied - Needs Simulation**
- **Objectives Satisfied by Existing Testcases**
- **Objectives Undecided with Testcases**
- **Objectives Undecided due to Runtime Error**

When you set the **Model coverage objectives** parameter to Enhanced MCDC in the Configuration Parameters window and perform test generation analysis, then open Model Slicer, you can choose a configuration by setting **Slice configuration list** to:

- Configuration to inspect Enhanced MCDC objective detectability
- Configuration to inspect test generation objective

To launch Model Slicer after running a test generation analysis, in the Results window, click **Inspect**. For more information, see Inspect Enhanced MCDC Objectives using Model Slicer (Simulink Design Verifier).

Debug equivalence tests by using Model Slicer

You can now use the Model Slicer in the Simulink Test Test Manager to debug equivalence tests. You can also debug tests that compare two simulation modes if one of the modes is set to **Normal**. For information on using the Model Slicer in the Test Manager, see Debugging Equivalence Test Failures Using Model Slicer (Simulink Test).

CI/CD Automation for Simulink Check (August 2022; Version 22.1.0)

In R2022a, the support package CI/CD Automation for Simulink Check provides tools to help you integrate your model-based process into a Continuous Integration and Continuous Deployment (CI/CD) system.

The support package provides:

- A customizable process modeling system to define your build and verification process
- A build system that can automatically generate and efficiently execute a process in your CI system
- The **Process Advisor** app for deploying and automating your prequalification process
- Integration with common CI systems

You can use the support package to help you set up a model-based design pipeline, reduce build time, reduce build failures, debug build failures, and deploy a consistent build and verification process.

For more information, see <https://www.mathworks.com/matlabcentral/fileexchange/115220>.

Functionality being removed or changed

getAvailableMetricIds function returns metrics from the Model Testing Dashboard app and Design Cost Estimation app

Behavior change

The function `getAvailableMetricIds` now returns metrics from the Model Testing Dashboard app and Design Cost Estimation app if either of these conditions exists:

- You specify 'Installed' as `false`.
- You have Fixed-Point Designer™ installed on your machine.

Functionality	Use This Instead
When you used the syntax <code>getAvailableMetricIds(metric_engine)</code> , the function returned only the metrics used by the Model Testing Dashboard app for the metric engine object <code>metric_engine</code> .	<p>If you have Fixed-Point Designer installed on your machine but you want to return metrics from only the Model Testing Dashboard app, you must update your code:</p> <p>Specify the 'App' as 'DashboardApp' and the 'Dashboard' as 'RequirementsBasedModelUnitTesting'.</p> <pre>metrics = getAvailableMetricIds(metric_engine,... 'App','DashboardApp',... 'Dashboard','RequirementsBasedModelUnitTesting');</pre>

For more information, see `getAvailableMetricIds`.

ModelAdvisor.CheckResult returns different values for the property status

Behavior change

In R2022a, the `ModelAdvisor.CheckResult` object returns different values for the `status` property.

Previously, valid values for the `status` property were the character vector values:

- 'Fail'
- 'Not Run'
- 'Pass'
- 'Warn'

In R2022a, valid values for the `status` property are the enumerated values:

- Failed
- Incomplete
- Justified
- NotRun
- Passed
- Warning

If you use the status property of a `ModelAdvisor.CheckResult` object, you may need to update your code:

Previous status Value	Current status Value
'Fail'	Failed
'Not Run'	NotRun
'Pass'	Passed
'Warn'	Warning

For more information, see `ModelAdvisor.CheckResult`.

Avoid using process callback functions

Still runs

A *process callback function* is a function that configures the Model Advisor and processes check results. In R2022a and later releases, avoid using process callback functions. Remove process callback functions from your `sl_customization.m` files.

To configure the Model Advisor without a process callback function, use the Model Advisor Configuration Editor to create a custom configuration file, then use a default configuration or a model configuration to set the startup configuration. For more information, see [Use the Model Advisor Configuration Editor to Customize the Model Advisor and Load and Associate a Custom Configuration with a Model](#).

To process check results without a process callback function, open the Model Advisor and click **Report** to generate a report for the check results. You can click a check in the Model Advisor, and then open the **Results** and **Results Details** tabs to view the check results. You can also use `ModelAdvisor.run` to collect and view check results.

R2021b

Version: 5.2

New Features

Bug Fixes

Compatibility Considerations

View compliance status of metrics in the Model Testing Dashboard

In R2021b, you can now use overlays in the Model Testing Dashboard to see if your testing artifacts comply with standard requirements-based testing practices. The overlays show if the metric results for a widget are compliant, non-compliant, or generate a warning that the metric results should be reviewed. Results are compliant if they show full traceability, test completion, or model coverage.

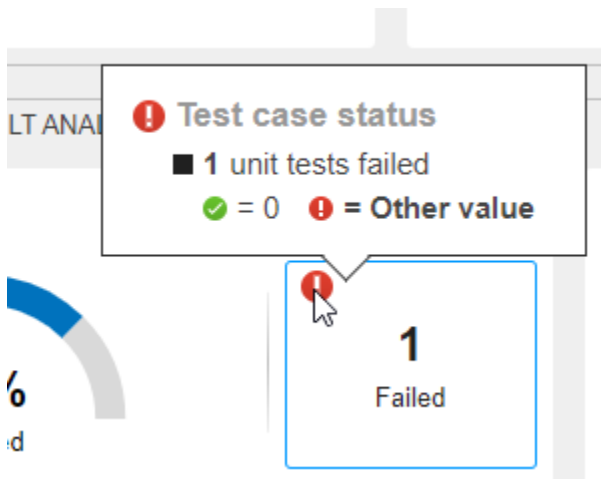
To see the overlays for a compliance category, select the category in the **Overlays** section of the dashboard toolstrip. The overlay appears on the widgets that have results in that category and the top right of the dashboard shows the number of widgets in each compliance category.

The screenshot displays the Model Testing Dashboard interface. The top toolbar includes a 'Compliant' indicator (green checkmark) and a 'Non-Compliant' indicator (red exclamation mark). The dashboard is divided into several sections:

- Navigation Pane:** Lists artifacts such as 'db_Controller', 'db_ControlMode', 'db_DriverSwRequest', and 'db_TargetSpeedThrottle'.
- Test Case Analysis:**
 - Requirements Linked to Tests:** A gauge chart showing 42.9% compliance (Requirements with Tests) and 12 Unlinked items.
 - Tests Linked to Requirements:** A gauge chart showing 87.5% compliance (Tests with Requirements) and 1 Unlinked item.
 - Tests Case Breakdown:** A table showing counts for Simulation (8), Equivalence (0), and Baseline (0).
 - Tests with Tag:** A table showing counts for tags like 1_Draft (1), 2_Under_review (2), 3_Reviewed (2), and 4_Released (3).
- Simulation Test Result Analysis:**
 - Model Test Status:** A gauge chart showing 75% Passed, 0 Inconclusive, 0 Untested, and 1 Failed.
 - Model Coverage:** A bar chart showing Aggregated Coverage for Execution (Achieved), Decision (Missed), Condition (Missed), and MC/DC (No data available).

Compliance icons are overlaid on the charts: a red exclamation mark for non-compliance and a green checkmark for compliance. The top right of the dashboard shows a summary of widget counts for each compliance category: 2 Compliant, 8 Non-Compliant, 0 Warning, and 4 Uncategorized.

To see the compliance thresholds for a metric, point to the overlay icon in the widget.



You can hide the overlay status icons by deselecting the overlays in the toolbar.

For more information on the compliance thresholds for each metric, see Model Testing Metrics.

Organize models using unit testing hierarchy in the Model Testing Dashboard

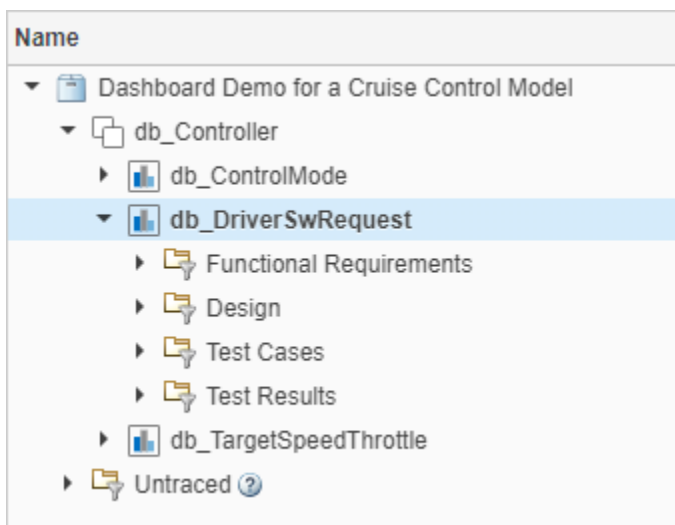
In R2021b, the Model Testing Dashboard organizes the models in a new hierarchy, with component models at the top of the hierarchy and unit models at the bottom of the hierarchy. The new hierarchy helps you to locate the models that require unit testing so you can assess their testing quality using the dashboard. The Model Testing Dashboard provides metric results for only the unit models.

By default, the Model Testing Dashboard defines models in two ways:

- Models that do not reference other models are units.
- Models that reference one or more models are components.

Alternatively, you can specify a model as a unit or a component by using labels in your project.

In the example below, the **Artifacts** pane shows that the component model `db_Controller` references the unit models `db_ControlMode`, `db_DriverSwRequest`, and `db_TargetSpeedThrottle`.



Expand a unit to see the artifacts that trace to it, organized by artifact type. For more information, see [Categorize Models in a Hierarchy as Components or Units](#).

Additionally, if you collect metric results programmatically, the `metric.Result` object has the new property `CollectionScope`, which describes the unit for which you collect metric results.

Measure pass and fail criteria metrics in the Model Testing Dashboard

In R2021b, you can use the new metrics **Test cases with pass/fail criteria** and **Test cases with pass/fail criteria distribution** to assess the quality of your requirements-based tests. The metrics determine if each test case contains pass/fail criteria such as verify statements, verification blocks, custom criteria, and logical or temporal assessments. Requirements-based tests should verify the functionality of your model using one or more of these criteria. Use the metrics to find and address tests that do not include pass/fail criteria.

To run the new metrics, in the Model Testing Dashboard, click **Collect Results**. In the **Simulation Test Result Analysis** section, the **Inconclusive** widget shows the number of tests that do not include pass/fail criteria.

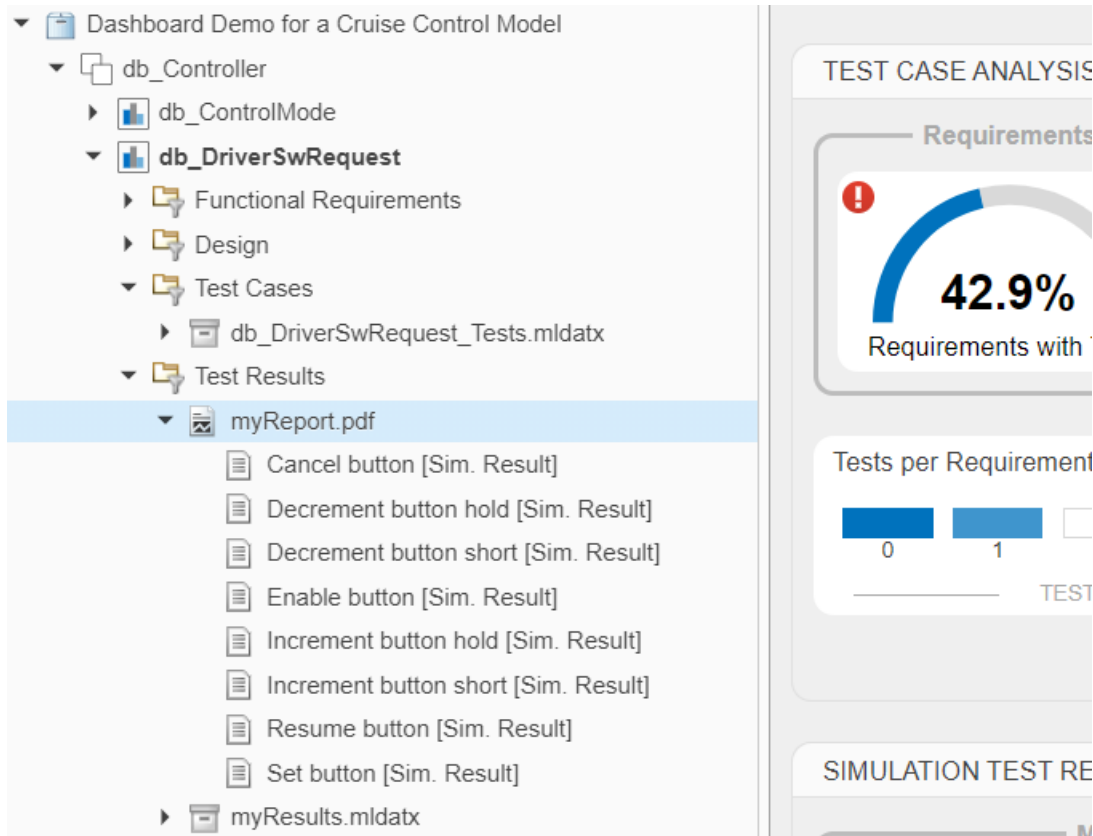
Alternatively, to run the metrics programmatically, use the `execute` function for a `metric.Engine` object and specify the identifiers `TestCaseVerificationStatus` and `TestCaseVerificationStatusDistribution`. For more information, see [Collect Metrics on Model Testing Artifacts Programmatically](#).

Added functions for programmatically analyzing requirements-based testing metrics

In R2021b, you can now use the function `updateArtifacts` to run the traceability analysis for a `metric.Engine` object and the function `getAvailableMetricIds` to get a list of the identifiers for the requirements-based testing metrics that you can collect. To collect results for all model testing metrics, pass the list into the `execute` function. For more information, see [Collect Metrics on Model Testing Artifacts Programmatically](#).

Trace additional test results in the Model Testing Dashboard

In R2021b, you can use the Model Testing Dashboard to trace test results in a PDF report, ZIP report, or DOCX report created by Simulink Test. A test results report appears in the **Test Results** section of the **Artifacts** panel of the dashboard under the model that it traces to.



To trace a report file in the **Artifacts** panel, you must open the dashboard for the project before generating the report.

Additionally, you can generate an MLDATX file of the test results before opening a project in the dashboard for the first time, then trace and do metric analysis on the test results.

For more information on the traceability of testing artifacts, see [Explore Status and Quality of Testing Activities Using the Model Testing Dashboard](#).

View summary of artifacts for each unit in the model testing metrics report

In R2021b, you can view a summary of the requirements-based testing artifacts in each unit of the model testing metrics report.

1.1. Artifact Summary

Artifact Group	Artifact Type	Number of Artifacts
Requirements	Functional requirement	23
Design	Block diagram	1
	Model reference	0
	Subsystem	22
	Stateflow chart	0
	MATLAB function	0
	Data dictionary file	1
Tests	Test case	0
	Test case result	0

After you generate a report, each unit in the report contains a table with a summary of the artifacts in that unit. The table is in the first subsection of each unit in the report. Use the summary to quickly gauge the size and structure of each unit in the report.

To generate a model testing metrics report, click **Report** in the Model Testing Dashboard or use the `generateReport` function. For an example of how to collect metrics programmatically and generate a report, see [Collect Metrics on Model Testing Artifacts Programmatically](#).

Artifact tracing enhancements for the Model Testing Dashboard

In R2021b, the Model Testing Dashboard detects changes to the MATLAB path and updates the impacted traceability information when you open the project. The dashboard updates the following relationships:

- From a model to another model, a library, or a data dictionary
- From a data dictionary to another data dictionary
- From a test to a model

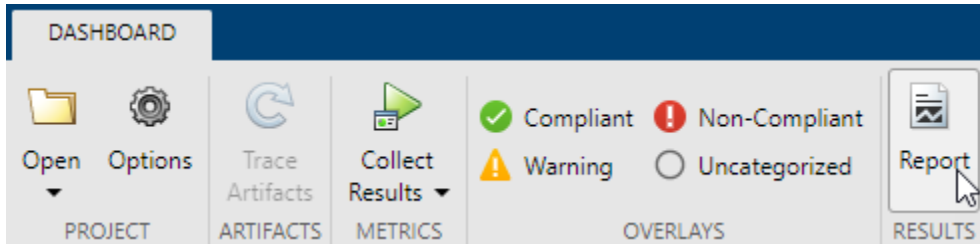
The dashboard diagnostics report on the ambiguous tracing relationships caused by file shadowing and path issues. The diagnostics also show you files that you should add to the path for tracing.

Additionally, when you trace artifacts or collect results, you can now cancel these operations by clicking the **Cancel** button under the progress bar.

For more information on artifact tracing in the Model Testing Dashboard, see [Manage Requirements-Based Testing Artifacts for Analysis in the Model Testing Dashboard](#).

Generate report from the Model Testing Dashboard

In R2021b, you can generate a requirements-based testing report from the toolstrip of the Model Testing Dashboard. Previously, to create a report, you needed to use the `generateReport` function. In the Model Testing Dashboard, collect results, then click the **Report** button.

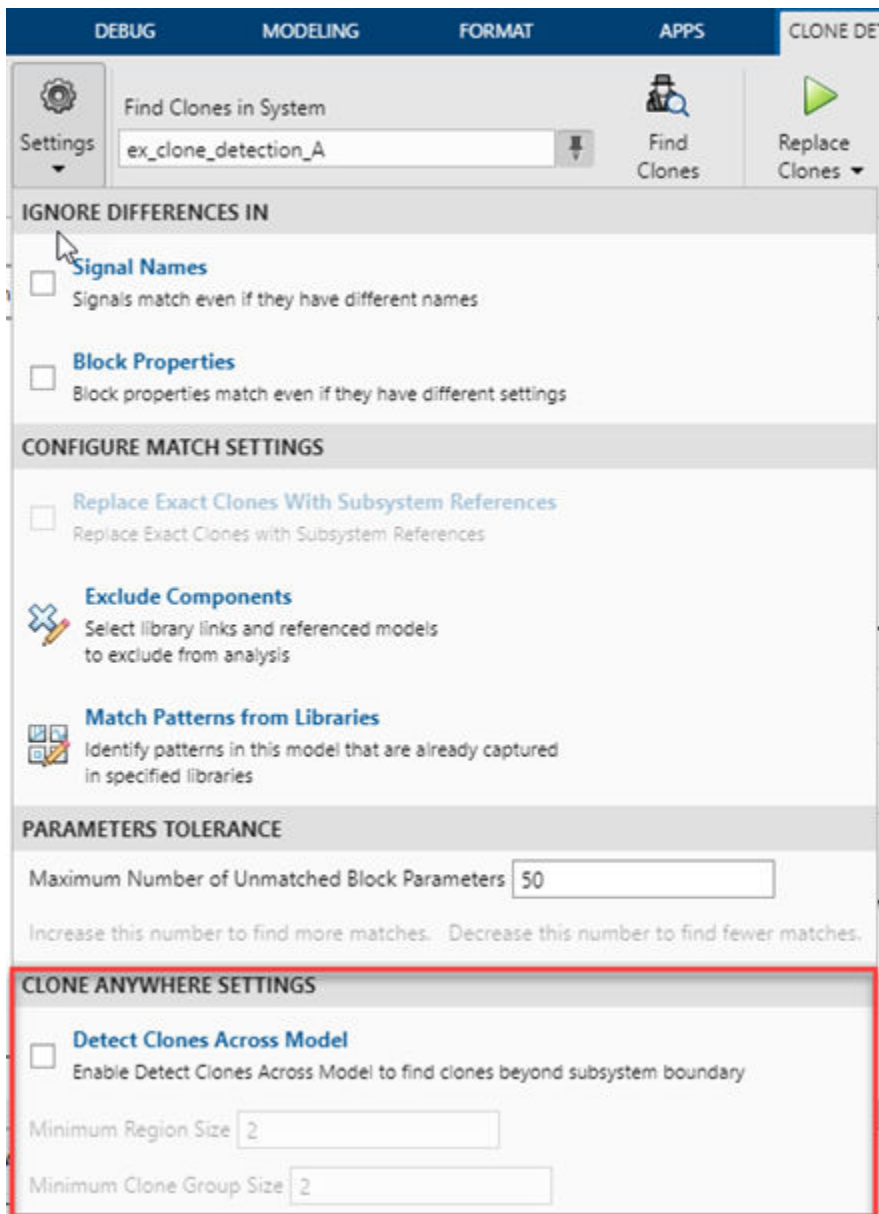


In the Create Metric Result Report dialog box, specify the file format and location for the report and click **Create**.

Find clones anywhere within the model

From R2021b, you can find exact clones beyond the boundaries of a subsystem programmatically or by using the Clone Detector app. Prior to R2021b, you could identify subsystem clones only, that is subsystems with identical region of blocks. Matching regions of blocks that were not in the subsystem were not recognized as clones.

To enable finding clones outside subsystem boundaries, in the **Clone Detector** tab, click **Settings**, then select **Detect Clones Across Model**.



To programmatically find clones outside subsystem boundaries, create an object of the `Simulink.CloneDetection.Settings` class, set the `DetectClonesAcrossModel` property to true, then pass this object to the `Simulink.CloneDetection.findClones` function to identify clones.

```
cloneDetectionSettings = Simulink.CloneDetection.Settings();
cloneDetectionSettings.DetectClonesAcrossModel = 1;
cloneDetectionSettings.MinimumRegionSize = 2;
cloneDetectionSettings.MinimumCloneGroupSize = 2;
cloneResults = Simulink.CloneDetection.findClones(cloneDetectionSettings);
```

For more information, see [Find Clones Anywhere in a Model](#).

Programmatically detect clones in multiple models

Starting in R2021b, you can detect clones programmatically in multiple Simulink models present across different folders. Prior to R2021b, you could identify clones only in a single model hierarchy.

To identify clones across multiple models, create an object of the `Simulink.CloneDetection.Settings` class, add folders containing the models to the `Folders` property, then pass this object to the `Simulink.CloneDetection.findClones` function.

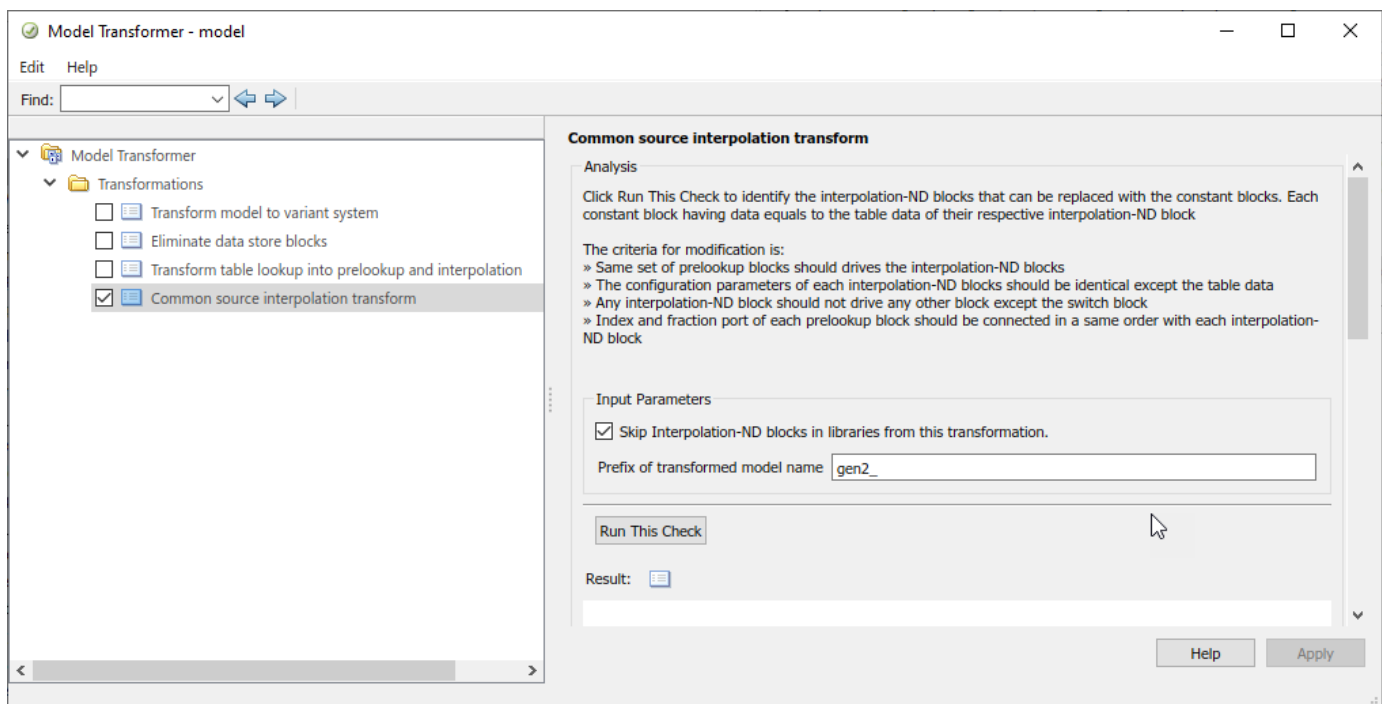
```
cloneDetectionSettings = Simulink.CloneDetection.Settings();
cloneDetectionSettings.Folders = {'Folder 1', 'Folder 2', 'Folder 3'};
cloneResults = Simulink.CloneDetection.findClones(cloneDetectionSettings);
```

For more information, see [Detect Clones Programmatically Across Folders](#).

Improve Code Efficiency by Merging Multiple Interpolation Using Prelookup Blocks

In R2021b, you can use the Model Transformer tool to replace multiple Interpolation Using Prelookup blocks that have same input signals connected from the outputs of Prelookup blocks into a single Interpolation Using Prelookup block. Reducing the number of Interpolation Using Prelookup blocks in a model reduces the number of variable assignments in the code, which improves the efficiency of the generated code. You can use the Model Transformer app or programmatic commands to refactor the model.

To optimize the model in the Model Transformer, select **Common source interpolation transform**.



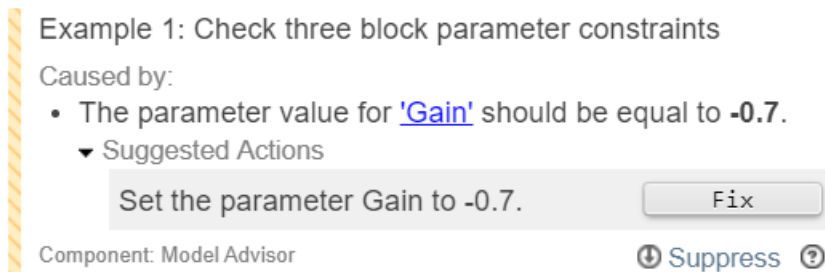
To programmatically run this check, use these MATLAB commands:

Syntax	Action
<code>Simulink.ModelTransform.CommonSourceInterpolation.identifyCandidates</code>	Identify eligible Interpolation Using Prelookup blocks to transform.
<code>Simulink.ModelTransform.CommonSourceInterpolation.refactorModel</code>	Replace Interpolation Using Prelookup blocks n-D blocks.

For more information, see [Improve Code Efficiency by Merging Multiple Interpolation Using Prelookup Blocks](#).

Improved edit-time check diagnostic interface for block constraint violations

In R2021b, the edit-time check diagnostics window now includes a **Fix** button you can use to address block constraint violations.



To enable the edit-time checking, in the **Modeling** tab, select **Model Advisor > Edit-Time Checks**.

For more information, see [Define Model Advisor Checks for Supported and Unsupported Blocks and Parameters](#)

Simplified block constraint check authoring

You can now author block constraint checks by using a new check definition format that allows you to more easily define custom Model Advisor checks that use block constraints.

In previous releases, when authoring a block constraint check, you had to create a separate XML file with the block constraints data and then specify the properties of this XML file as part of the check definition function. In R2021b, the constraint creation is part of the block constraint check definition. Consequently, the `Advisor.authoring.generateBlockConstraintsDataFile` function is no longer required and the `Advisor.authoring.createBlockConstraintCheck` function has a 'Constraints' name-value argument that accepts a callback to a constraints creation function.

For more information, see [Define Model Advisor Checks for Supported and Unsupported Blocks and Parameters](#).

Additional Model Slicer support for Simulink constructs

Model Slicer now supports the following:

- Analyze the model containing array of buses.
- Analyze the model containing Observer model elements.
- Add Virtual elements as starting point.

Guideline Sub-ids for additional MAB/JMAAB checks

In R2021b, these MAB/JMAAB checks are modified to have Guideline sub-ids:

Check	Guideline
Check default transition placement in Stateflow charts	jc_0531: Default transition

High Integrity Systems Modeling Checks: Improve quality and compliance to guidelines

In R2021b, these high-integrity modeling checks are added:

Check	Equivalent Guideline
Check for parameter tunability ignored for referenced models	hisl_0072: Usage of tunable parameters for referenced models
Check usage of bit-shift operations	hisl_0073: Usage of bit-shift operations
Check safety-related diagnostic settings for variants	hisl_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants
Check MATLAB functions not supported for code generation	himl_0012: Usage of MATLAB functions for code generation
Metrics for generated code complexity	himl_0013: Limitation of built-in MATLAB Function complexity

This table identifies modeling guidelines that were modified in R2021a.

Check	Rationale
Check usage of remainder and reciprocal operations	<ul style="list-style-type: none"> • Updated the title. • The check now supports Stateflow and MATLAB domains.
Check usage of square root operations	<ul style="list-style-type: none"> • Updated the title. • The check now supports Stateflow and MATLAB domains.
Check usage of log and log10 operations	<ul style="list-style-type: none"> • Updated the title. • The check now supports Stateflow and MATLAB domains.
Check data types for blocks with index signals	The check now analyzes external MATLAB files.
hisl_0019: Usage of bitwise operations	The check now supports Stateflow and MATLAB domains.

Compatibility Considerations

This table identifies checks that are removed from the current release:

Model Advisor Check	Corresponding Modeling Guideline
Check usage of shift operations for Stateflow data	hisf_0064: Shift operations for Stateflow data to improve code compliance
Check usage of equality operators in MATLAB Function blocks	himl_0009: MATLAB code with equal / not equal relational operators

Observe impact of Simulink parameters using Model Slicer

You can now use Model Slicer to analyze the impact of Simulink Parameters on the model. To find the impact of parameters on Simulink blocks, use the following functions:

Function	Example
parametersAffectingBlock	<p>Find Parameters Affecting a Block</p> <pre>[params, slicerObj] = parametersAffectingBlock(pd, 'blockA');</pre> <p>Input arguments:</p> <ul style="list-style-type: none"> pd is an object of <code>SLSlicerAPI.ParameterDependence</code>. <pre>pd = slicerObj.parameterDependence;</pre> <ul style="list-style-type: none"> blockA is a Simulink block path/handle/SID. <p>Output arguments:</p> <ul style="list-style-type: none"> params is an array of <code>Simulink.VariableUsage</code> objects representing parameters affecting blockA. slicerObj is an object of <code>SLSlicerAPI.SLSlicer</code> which has appropriate starting points added for user to highlight and validate.

Function	Example
blocksAffectingparameter	<p>Find Blocks Affected By a Parameter</p> <pre>[affectedBlocks, slicerObj] = blocksAffectedByParameter(pd, varUsage);</pre> <p>Input arguments:</p> <ul style="list-style-type: none"> • pd is an object of SLSlicerAPI.ParameterDependence. <pre>pd = slicerObj.parameterDependence;</pre> <ul style="list-style-type: none"> • varUsage is an object of Simulink.VariableUsage. <pre>varUsage = Simulink.VariableUsage('paramA', 'base workspace');</pre> <p>Output arguments:</p> <ul style="list-style-type: none"> • affectedBlocks is an array of block handles affected downstream by varUsage . • slicerObj is an object of SLSlicerAPI.SLSlicer which has appropriate starting points added for user to highlight and validate.

Additional checks to verify compliance with CERT C secure coding standards

From R2021b, these Model Advisor checks are now compliant with CERT C modelling guidelines:

- Check configuration parameters for MISRA C:2012
- Check usage of Abs blocks
- Check usage of remainder and reciprocal operations
- Check usage of square root operations
- Check usage of While Iterator blocks
- Check for blocks not recommended for C/C++ production code deployment
- Check data types for blocks with index signals
- Check usage of Reciprocal Sqrt blocks
- Check global variables in graphical functions
- Check usage of bit-shift operations
- Check safety-related optimization settings for data type conversions
- Check safety-related optimization settings for division arithmetic exceptions
- Check model file name

- Check model object names

Enhancements to edit-time checking to identify more incompatibilities

You can now identify compatibility issues earlier in the model design process by using edit-time checking. In R2021b, when you use edit-time checking, you can view some violations of these Model Advisor checks:

- Check usage of Stateflow states (Simulink Code Inspector)
- Check usage of User-Defined Function blocks (Simulink Code Inspector)

Edit-time checking does not flag violations for all the constraints of these checks. It flags some specific constraint violations. By clicking the warning icon, you can see information on the constraint violation. For more information, see the check documentation and Check Model Compatibility While You Edit (Simulink Code Inspector).

Functionality being removed or changed

ModelAdvisor.ListViewParameter class and ModelAdvisor.Check ListViewVisible property will be removed

Still runs

The `ModelAdvisor.ListViewParameter` class and the `ListViewVisible` property of the `ModelAdvisor.Check` class will be removed.

Functionality	Use This Instead
When you authored a custom check, the <code>ModelAdvisor.ListViewParameter</code> class enabled you to populate the Model Advisor Result Explorer. Setting the <code>ModelAdvisor.Check.ListViewVisible</code> property to <code>true</code> enabled the Model Advisor Result Explorer. The Model Advisor Result Explorer allowed you to locate, view, and change elements of a model.	Use hyperlinks in the Model Advisor results to view and modify model elements that are being flagged by the Model Advisor check.

For more information about using the recommended functionality, see [Fix a Model to Comply with Conditions that You Specify with the Model Advisor](#) and [Create and Deploy a Model Advisor Custom Configuration](#).

R2021a

Version: 5.1

New Features

Bug Fixes

Compatibility Considerations

View detailed traceability paths and diagnostics for requirements-based testing artifacts

In R2021a, you can view the traceability path from an artifact to its component in the Model Testing Dashboard. On the **Artifacts** pane, right-click the artifact and click **View trace to component**. A traceability graph opens in a new tab. The graph shows the connections and intermediate artifacts that the dashboard traced from the component to the artifact. To see the relationship between two artifacts, point to the arrow that connects the artifacts.

Additionally, when the dashboard traces your artifacts and collects metric results, you can view errors and warnings in the new diagnostic viewer. At the bottom of the Model Testing Dashboard, click **Diagnostics**. The diagnostic viewer shows warnings and errors that indicate:

- Artifacts that returned errors when loaded
- Artifacts that the dashboard does not support
- Links that the dashboard does not trace
- Model callbacks that the dashboard deactivates
- Test results without coverage
- Test results that do not include simulation results


Generate model testing metrics report

In R2021a, you can generate a PDF or HTML report of the metric results from the Model Testing Dashboard. For each software component in the project, the report shows the metric results that appear in the dashboard. Use the report to archive and share metric data on the quality and completeness of your requirements-based testing activities.

To generate a report, use the new `generateReport` function. For an example of how to collect metrics programmatically and generate a report, see [Collect Metrics on Model Testing Artifacts Programmatically](#).

Find artifacts in the Model Testing Dashboard by using navigation enhancements

In R2021a, in the Model Testing Dashboard **Artifacts** pane, you can quickly find artifacts that you want to investigate by expanding, collapsing, and scrolling to nodes in the artifact list. The menu to

the right of the search bar  enables you to:

- Collapse the expanded nodes in the list
- Collapse and expand the sections under the selected node
- Scroll to the component that the open dashboard tab reflects
- Scroll to the currently selected node

Additionally, you can right-click an artifact in the list to:

- Open the dashboard for the component that the artifact traces to
- Scroll to the component that the artifact traces to

- Scroll to the parent node that contains the artifact
- Show a file artifact in your file browser

Previously, you could view dashboard results for only one component at a time. In R2021a, you can open the dashboard for each component in a separate tab in the Model Testing Dashboard dialog. On the **Artifacts** pane, right-click a component and click **Open in new tab**.

Select software component models for analysis in the Model Testing Dashboard

Previously, the Model Testing Dashboard showed traceability and analysis results for all models in your project. In R2021a, you can specify the models that you want the dashboard to analyze by labeling the models in the project. To label the software component models and configure the dashboard to recognize them, see [Model Software Components for Requirements-Based Testing](#). In the Model Testing Dashboard, the **Artifacts** pane lists only the models that have the category and label combination that you specified. By limiting the list to only software component models, you can track your testing progress without sorting through models that do not use requirements-based testing that the dashboard analyzes.

Additionally, in R2021a, the Model Testing Dashboard now analyzes these artifacts:

- Requirement links to embedded MATLAB functions
- Custom requirements
- Requirements links that have custom types
- Data dictionaries
- Test results that you have collected in the Test Manager but not exported to a results file

For more information about artifacts and links that the dashboard does not support, see [Resolve Missing Artifacts, Links, and Results in the Model Testing Dashboard](#).

Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines

This table lists the new Model Advisor checks that verify compliance with the High-Integrity System Modeling guidelines.

Model Advisor Check	Check ID
Check for divide-by-zero calculations	mathworks.hism.hisl_0067
Check safety-related settings for hardware implementation	mathworks.hism.hisl_0071
Check usage of recursions	mathworks.hism.hisf_0004
Check for model elements that do not link to requirements	mathworks.hism.hisl_0070

Compatibility Considerations

In R2021a, due to compatibility considerations, the following checks have been removed from Model Advisor:

Model Advisor Check	Check ID
Check safety-related optimization settings for Loop unrolling threshold	mathworks.hism.hisl_0051

Create user-defined fields and review additional information when designing block constraint checks

Following enhancements are made to block constraint check authoring:

- You can create user-defined fields to identify what check failures are caused by.
- Additional information is provided on what causes a check failure in the diagnostic window.
- Diagnostic window now lists information for multiple constraint violations.
- Diagnostic window includes links to the violating block parameters.
- You can click on violating block parameters to highlight them in the Block dialog box.

Add, remove, clear, get, save, and load exclusions programmatically in the Model Advisor Exclusion Editor

The Model Advisor Exclusion Editor can now be used with the following functions:

Task	Function	Syntax
Add a new exclusion in Model Advisor.	<code>Advisor.addExclusion</code>	<code>Advisor.addExclusion('modelname', 'type', 'id')</code>
Remove exclusions from Model Advisor.	<code>Advisor.removeExclusion</code>	<code>Advisor.removeExclusion('modelname', 'type', 'id')</code>
Clear all exclusions from Model Advisor.	<code>Advisor.clearExclusion</code>	<code>Advisor.clearExclusion('modelname')</code>
Get exclusions for a model or a filter.	<code>Advisor.getExclusion</code>	<code>Advisor.getExclusion('modelname')</code>
Save exclusions to the default option or to a new file.	<code>Advisor.saveExclusion</code>	<code>Advisor.saveExclusion('modelname')</code>
Load default exclusions stored inside the model or according to the path settings.	<code>Advisor.loadExclusion</code>	<code>Advisor.loadExclusion('modelname', 'filepath')</code>

For more information, see [Exclude Blocks From the Model Advisor Check Analysis](#).

Additional checks for JMAAB and MAB Guidelines

This table identifies the new Model Advisor checks that verify compliance with the MAB and JMAAB Modeling guidelines.

Model Advisor Check	Check ID	MAB/JMAAB
Check usage of graphical functions in Stateflow	mathworks.jmaab.jc_0804	MAB and JMAAB

Model Advisor Check	Check ID	MAB/JMAAB
Check for division by zero in Simulink	mathworks.jmaab.jc_0794	MAB and JMAAB
Check lines of code in MATLAB Functions	mathworks.jmaab.na_0016	MAB
Check nested conditions in MATLAB Functions	mathworks.jmaab.na_0018	MAB

For more information, see Model Checks for MAB and JMAAB Compliance.

Use Model Advisor Exclusion Editor with Stateflow

From R2021a, the following enhancements to Model Advisor Exclusion Editor are added for Stateflow:

- Model Advisor exclusions can be applied to Stateflow states, transitions, junctions, MATLAB functions, graphical functions, Simulink functions, Simulink based states, and truth tables.
- You can launch Model Advisor Exclusion Editor from the Stateflow context menu.
- You can launch Model Advisor Check Selector from the Stateflow context menu.
- You can launch Model Advisor from the Stateflow context menu.

For more information, see Exclude Blocks From the Model Advisor Check Analysis.

Enhancements to edit-time checking to identify additional incompatibility issues

You can now identify compatibility issues earlier in the model design process by using edit-time checking for Simulink Code Inspector™ checks. In R2021a, when you use edit-time checking, you can view some violations of these Model Advisor checks:

- Check for unsupported blocks (Simulink Code Inspector)
- Check model for reusable subsystems that use the same function interfaces (Simulink Code Inspector)
- Check usage of Sources blocks (Simulink Code Inspector)
- Check usage of Signal Routing blocks (Simulink Code Inspector)
- Check usage of Math Operations blocks (Simulink Code Inspector)
- Check usage of Signal Attributes blocks (Simulink Code Inspector)
- Check usage of Logical and Bit Operations blocks (Simulink Code Inspector)
- Check usage of Lookup Tables blocks (Simulink Code Inspector)
- Check usage of Ports and Subsystems blocks (Simulink Code Inspector)
- Check usage of Discontinuities blocks (Simulink Code Inspector)
- Check usage of Sinks blocks (Simulink Code Inspector)
- Check usage of Discrete blocks (Simulink Code Inspector)
- Check usage of Stateflow charts (Simulink Code Inspector)

Edit-time checking does not flag violations for all the constraints of these checks. It flags some specific constraint violations. For more information, see the check documentation and Check Model Compatibility While You Edit (Simulink Code Inspector).

Inspect enhanced MCDC objectives generated by Simulink Design Verifier using Model Slicer

You can now launch Model Slicer for satisfied enhanced MCDC objectives to verify that the test case generated by Simulink Design Verifier™ impacts the detection site without being masked.

For more information, see Enhanced MCDC Coverage in Simulink Design Verifier (Simulink Design Verifier).

Debug test failures using Model Slicer

You can now use Model Slicer in the Simulink Test Manager to debug failed verify signals. When you simulate the model in debug mode, Model Slicer highlights the model components that impact the failed signals at each time step. You can move between failure regions to debug the cause of the verification failures.

Detect and replace clones in a model with command-line APIs

In R2021a, you can programmatically identify clones in a Simulink model and replace the clones with links to library blocks.

Syntax	Purpose
<code>Simulink.CloneDetection.findClones</code>	Find clones in a model. Optionally, you can constrain the threshold for matching a subsystem by using <code>Settings</code> class.
<code>Simulink.CloneDetection.replaceClones</code>	Replace the clones in a model with links to library blocks. Optionally, you can replace clones with reference blocks by using the <code>ReplacementConfig</code> class.
<code>Simulink.CloneDetection.checkEquivalency</code>	Check the equivalency of the original model with the model created after clone replacement.

For more information, see Detect and Replace Clones Programmatically.

Improved clone detection for refactoring models with nested clones and masked subsystem

In R2021a, you can now use clone detection to refactor models that contain clone groups under another clone group. This feature also supports the refactoring and replacement of clones in masked subsystems.

For more information, see Clone Detector and Enable Component Reuse by Using Clone Detection.

Edit-time checking support for library blocks

In R2021a, edit-time checks identify modeling issues for library blocks in a model. For more information on edit-time checking, see [Check Model Compliance by Using the Model Advisor](#).

Improved analysis of models that have callbacks in the Model Testing Dashboard

Previously, when the Model Testing Dashboard analyzed a model, any loading and closing model callbacks were executed. Model callbacks that changed the model resulted in an error or stale results in the Model Testing Dashboard. In R2021a, when the dashboard analyzes a model, loading and closing callbacks are not executed.

R2020b

Version: 5.0

New Features

Bug Fixes

Compatibility Considerations

Model Testing Dashboard: Track completeness of requirements-based testing for compliance to standards such as ISO 26262

Assess the status and quality of your requirements-based testing by using metric data in the new Model Testing Dashboard. The metrics in the dashboard measure different aspects of model testing completeness and quality based on industry-recognized standards such as ISO 26262 and DO-178. Use the dashboard to view:

- Summary data for requirements, tests, and traceability between requirements and tests
- Status and results of the latest test runs
- Model coverage measurements achieved by tests and justifications
- A list of the artifacts in the project, organized by the associated models

The dashboard widgets show a summary of the testing metric data for one component. To explore the data in more detail, click an individual widget. A table lists the artifacts in the component and their results for the metric. The table provides hyperlinks to open each artifact so that you can view details about the artifact and address testing quality issues. For more information about using the Model Testing Dashboard, see [Explore Status and Quality of Testing Activities Using the Model Testing Dashboard](#).

In R2020b, you can also collect the metrics programmatically by using the new metric API. To collect metrics for a project, use these new objects and functions.

Object	Description
<code>metric.Engine</code>	Collect testing metric data for a project
<code>metric.Result</code>	Access results for a testing metric

Function	Description
<code>execute</code>	Collect testing metric data for a metric engine
<code>getMetrics</code>	Return testing metric results from a metric engine
<code>openArtifact</code>	Open the artifact that a metric result references
<code>getArtifactErrors</code>	View errors that occurred during artifact analysis

Viewing the metric results data and details in the dashboard requires a Simulink Check license. To collect results for a metric, you must have the licenses required to edit the associated artifacts, such as Simulink Requirements™, Simulink Test, or Simulink Coverage. For more information on the metrics and the required licenses, see [Model Testing Metrics](#).

Faster calculation of Model Advisor metric checks

In R2020b, the Model Advisor collects and analyzes data for Model Metrics checks faster than in R2020a.

Previously, the **Subsystem depth metric** check returned results for both the **Subsystem Depth** and **Subsystem Level** for a component. In R2020b, the metric is simplified to return only the **Subsystem Depth** result. The result is the depth of a component relative to the top level of the hierarchy, which is the analysis root. For more information, see [Subsystem depth metric](#).

Additionally, the **Cyclomatic complexity metric** check no longer shows aggregated results for a component and its descendants. Instead, the metric shows only local results for the component. For more information, see Cyclomatic complexity metric.

Compatibility Considerations

Previously, the **Subsystem depth metric** check returned these results for the components in a hierarchy:

- **Value** — The maximum depth of all hierarchical children of a component. In the Model Advisor, this result was labelled **Subsystem Depth**.
- **Measure** — The depth of a component relative to the top level of the hierarchy (the analysis root). In the Model Advisor, this result was labelled **Subsystem Level**.

In R2020b, the metric no longer returns the maximum depth of hierarchical children of a component, which was previously returned in the **Value**. Instead, the **Value** result is the depth of a component relative to the top level of the hierarchy. This data was previously returned as the **Measure** result. The metric no longer returns a **measure** result.

Debug counter examples from Design Error Detection results in Simulink Design Verifier using Model Slicer

You can now use Model Slicer along with Simulink Design Verifier to debug the Design Error Detection violations. Following are the design errors supported:

- Division by zero
- Integer Overflow
- Non-Finite and NaN (Not a Number) floating-point values
- Specified minimum and maximum value violations
- Datastore access violations
- Specified block input range violations

For more information, see [Debug Integer Overflow Design Error Detection using Model Slicer \(Simulink Design Verifier\)](#).

Updates to JMAAB Checks

This table lists the JMAAB checks that can now be run by selecting the Guideline Sub-IDs.

Model Advisor Check	Check ID
Check usable characters for signal names and bus names	mathworks.jmaab.jc_0222
Check usable characters for parameter names	mathworks.jmaab.jc_0232
Check usage of Lookup Tables	mathworks.jmaab.jc_0626
Check usage of Discrete-Time Integrator block	mathworks.jmaab.jc_0627
Check usable characters for Stateflow data names	mathworks.jmaab.jc_0795

Model Advisor Check	Check ID
Check for unconnected signal lines and blocks	mathworks.jmaab.db_0081
Check transitions in Stateflow Flow charts	mathworks.jmaab.db_0132
Check for propagated signal labels	mathworks.jmaab.jc_0009

The following table lists the JMAAB checks that are now edit-time compliant.

Model Advisor Check	Check ID
Check usage of parallel states	mathworks.jmaab.jc_0721
Check usage of transitions to external states	mathworks.jmaab.jc_0723
Check starting point of internal transition in Stateflow	mathworks.jmaab.jc_0760
Check prohibited combination of state action and flow chart	mathworks.jmaab.jc_0762
Check Stateflow chart action language	mathworks.jmaab.jc_0790
Check for state in state machines	mathworks.jmaab.db_0137
Check for MATLAB expressions in Stateflow charts	mathworks.jmaab.db_0127
Check transitions in Stateflow Flow charts	mathworks.jmaab.db_0132
Check entry formatting in State blocks in Stateflow charts	mathworks.jmaab.jc_0501

For more information, see Model Checks for MAB and JMAAB Compliance.

Test failure debugging using Model Slicer

You can now use the Model Slicer in the Test Manager to debug a failed signal in a baseline test failure. When you simulate the model in debug mode, the model components that impact the failed signal are highlighted for each time step. You can move between failure regions to debug the cause of the baseline differences. To use this feature, you must have a Simulink Check license.

For more information, see Debugging Baseline Test Failures Using Model Slicer (Simulink Test).

Improvements to Model Advisor Exclusion Editor

Model Advisor Exclusion Editor is now revamped to be a customizable editor with the following improvements:

- Trace model object from the exclusion editor.
- Edit exclusions from the exclusion editor view.
- Select multiple checks together to perform group actions through Check Selector window.
- Display details of the checks in resizable columns.
- Display Check titles and Check groups.
- Check Selector with grouping mechanism (tree hierarchy).
- Sorting mechanism for Exclusion Editor columns.

- Improvements to Save and Search mechanism.
- Search option in Check Selector window.

For more information, see Exclude Blocks From the Model Advisor Check Analysis.

Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines

This table identifies the new Model Advisor checks that verify compliance with High-Integrity System Modeling guidelines. Running these checks triggers an extensive analysis using Simulink Design Verifier.

Model Advisor Check	Check ID
Check usage of Sqrt blocks	mathworks.hism.hisl_0003
Check usage of Reciprocal Sqrt blocks	mathworks.hism.hisl_0028
Check usage of Math Function blocks (rem and reciprocal functions)	mathworks.sldv.hism.hisl_0002
Check usage of Math Function blocks (log and log10 functions)	mathworks.sldv.hism.hisl_0004

In R2020b, due to compatibility considerations, the following checks are removed from Model Advisor:

Model Advisor Check	Check ID
Check usage of Math Function blocks (rem and reciprocal functions)	mathworks.hism.hisl_0002
Check usage of Math Function blocks (log and log10 functions)	mathworks.hism.hisl_0004

Model Advisor checks for ISO 25119 and EN 50657 standards

A new check group was added to the Model Advisor. You can use these checks to verify compliance with ISO 25119 standards. To execute these check, open the Model Advisor and click on:

- **By Product > Simulink Check > Modeling Standards > IEC 61508, IEC 62304, ISO 26262, ISO 25119, EN 50128, and EN 50657 Checks**
- **By Task > Modeling Standards for ISO 25119**

You can use the EN 50128 checks to verify compliance with EN 50657:2017 standards. To reflect this change, these Model Advisor folders have been updated:

- **By Product > Simulink Check > Modeling Standards > IEC 61508, IEC 62304, ISO 26262, ISO 25119, EN 50128, and EN 50657 Checks**
- **By Task > Modeling Standards for ISO 50128/EN 50657**

For more information about the checks, see IEC 61508, IEC 62304, ISO 26262, ISO 25119, and EN 50128/EN 50657 Checks.

Refactor models by replacing exact clones with Subsystem Reference blocks

In R2020a, you could refactor a model by replacing similar and exact clones with library links. In R2020b, in addition to library links, you can replace exact clones with Subsystem Reference blocks.

To enable model refactoring, open a model and select **Configure Match Settings > Replace Exact Clones with Subsystem References** under the **Settings** drop-down menu.

For more information, see [Replace Exact Clones with Subsystem Reference](#).

R2020a

Version: 4.5

New Features

Bug Fixes

Compatibility Considerations

Model Advisor checks to support MAB v5.0 modeling guidelines

In R2020a, the MathWorks® Automotive Advisory Board (MAAB) modeling guidelines is reintroduced as the MathWorks Advisory Board (MAB) modeling guidelines, version 5.0.

The Model Advisor is updated to support the new modeling guidelines. To view the MAB checks, use either:

- **By Product > Simulink Check > Modeling Standards > MAB Checks**
- **By Task > Modeling Standards for MAB**

This table identifies the new Model Advisor checks for the MAB v5.0 modeling guidelines.

Model Advisor Check	Check ID
Check length of model file name	mathworks.jmaab.jc_0241
Check length of folder name at every level of model path	mathworks.jmaab.jc_0242
Check length of subsystem names	mathworks.jmaab.jc_0243
Check length of block names	mathworks.jmaab.jc_0247
Check length of Inport and Outport names	mathworks.jmaab.jc_0244
Check usable characters for signal names and bus names	mathworks.jmaab.jc_0222
Check usable characters for parameter names	mathworks.jmaab.jc_0232
Check length of signal and bus names	mathworks.jmaab.jc_0245
Check length of parameter names	mathworks.jmaab.jc_0246
Check usable characters for Stateflow data names	mathworks.jmaab.jc_0795
Check length of Stateflow data name	mathworks.jmaab.jc_0796
Check duplication of Simulink data names	mathworks.jmaab.jc_0791
Check unused data in Simulink Model	mathworks.jmaab.jc_0792
Check for unused data in Stateflow Charts	mathworks.jmaab.jc_0700
Check Signed Integer Division Rounding mode	mathworks.jmaab.jc_0642
Check diagnostic settings for incorrect calculation results	mathworks.jmaab.jc_0806
Check Model Description	mathworks.jmaab.jc_0603
Check if blocks are shaded in the model	mathworks.jmaab.jc_0604
Check block orientation	mathworks.jmaab.jc_0110
Check for consistency in model element names	mathworks.jmaab.jc_0602
Check for avoiding algebraic loops between subsystems	mathworks.jmaab.jc_0653
Check signal line labels	mathworks.jmaab.jc_0008
Check for propagated signal labels	mathworks.jmaab.jc_0009
Check signal flow in model	mathworks.jmaab.db_0097

Model Advisor Check	Check ID
Check if tunable block parameters are defined as named constants	mathworks.jmaab.jc_0645
Check for sample time setting	mathworks.jmaab.jc_0641
Check usage of fixed-point data type with non-zero bias	mathworks.jmaab.jc_0643
Check type setting by data objects	mathworks.jmaab.jc_0644
Check undefined initial output for conditional subsystems	mathworks.jmaab.jc_0640
Check usage of Merge block	mathworks.jmaab.jc_0659
Check default/else case in Switch Case blocks and If blocks	mathworks.jmaab.jc_0656
Check operator order of Product blocks	mathworks.jmaab.jc_0610
Check signs of input signals in product blocks	mathworks.jmaab.jc_0611
Check for parentheses in Fcn block expressions	mathworks.jmaab.jc_0622
Check icon shape of Logical Operator blocks	mathworks.jmaab.jc_0621
Comparing floating point types in Simulink	mathworks.jmaab.jc_0800
Check usage of Lookup Tables	mathworks.jmaab.jc_0626
Check usage of Memory and Unit Delay blocks	mathworks.jmaab.jc_0623
Check for cascaded Unit Delay blocks	mathworks.jmaab.jc_0624
Check usage of Discrete-Time Integrator block	mathworks.jmaab.jc_0627
Check usage of the Saturation blocks	mathworks.jmaab.jc_0628
Check output data type of operation blocks	mathworks.jmaab.jc_0651
Check display for port blocks	mathworks.maab.jc_0081
Check for usage of Data Store Memory blocks	mathworks.jmaab.jc_0161
Check input and output datatype for Switch blocks	mathworks.jmaab.jc_0650
Check settings for data ports in Multiport Switch blocks	mathworks.jmaab.jc_0630
Check definition of Stateflow events	mathworks.jmaab.db_0126
Check usable number for first index	mathworks.jmaab.jc_0701
Check execution timing for default transition path	mathworks.jmaab.jc_0712
Check scope of data in parallel states	mathworks.jmaab.jc_0722
Check for unconnected objects in Stateflow Charts	mathworks.jmaab.jc_0797
Check usage of exclusive and default states in state machines	mathworks.maab.db_0137
Check for parallel Stateflow state used for grouping	mathworks.jmaab.jc_0721
Check usage of transitions to external states	mathworks.jmaab.jc_0723

Model Advisor Check	Check ID
Check for unexpected backtracking in state transitions	mathworks.jmaab.jc_0751
Check starting point of internal transition in Stateflow	mathworks.jmaab.jc_0760
Check usage of internal transitions in Stateflow states	mathworks.jmaab.jc_0763
Check prohibited combination of state action and flow chart	mathworks.jmaab.jc_0762
Check transition orientations in flow charts	mathworks.maab.db_0132
Check usage of unconditional transitions in flow charts	mathworks.jmaab.jc_0773
Check terminal junctions in Stateflow	mathworks.jmaab.jc_0775
Check usage of Stateflow comments	mathworks.jmaab.jc_0738
Check Stateflow chart action language	mathworks.jmaab.jc_0790
Check usage of numeric literals in Stateflow	mathworks.jmaab.jc_0702
Check order of state action types	mathworks.jmaab.jc_0733
Check repetition of Action types	mathworks.jmaab.jc_0734
Check if state action type 'exit' is used in the model	mathworks.jmaab.jc_0740
Check updates to variables used in state transition conditions	mathworks.jmaab.jc_0741
Check usage of transition conditions in Stateflow transitions	mathworks.jmaab.jc_0772
Check condition actions and transition actions in Stateflow	mathworks.jmaab.jc_0753
Check prohibited comparison operation of logical type signals	mathworks.jmaab.jc_0655
Check for implicit type casting in Stateflow	mathworks.jmaab.jc_0802
Check uniqueness of Stateflow State and Data names	mathworks.jmaab.jc_0732
Check uniqueness of State names	mathworks.jmaab.jc_0730
Check usage of State names	mathworks.jmaab.jc_0731
Check indentation of code in Stateflow states	mathworks.jmaab.jc_0736
Check for usage of text inside states	mathworks.jmaab.jc_0739
Check placement of Label String in Transitions	mathworks.jmaab.jc_0770
Check position of comments in transition labels	mathworks.jmaab.jc_0771
Check usage of parentheses in Stateflow transitions	mathworks.jmaab.jc_0752
Check for comments in unconditional transitions	mathworks.jmaab.jc_0774
Check usage of Simulink function in Stateflow	mathworks.jmaab.na_0042

Compatibility Considerations

In some instances, checks already existed but their IDs and/or titles have changed. This table maps the previous check ID to their new check ID. When a check name has changed, the "Model Advisor Check Name" column specifies the check name for both R2019b and R2020a.

Model Advisor Check Name	R2019b Check ID	R2020a Check ID
Check file names	mathworks.maab.ar_0001	mathworks.jmaab.ar_0001
Check folder names	mathworks.maab.ar_0002	mathworks.jmaab.ar_0002
Check Subsystem names	mathworks.maab.jc_0201	mathworks.jmaab.jc_0201
Check port block names	mathworks.maab.jc_0231	mathworks.jmaab.jc_0211
Check Model font settings (R2019b) Check font formatting	mathworks.maab.db_0043	mathworks.jmaab.db_0043
Check signal line connections (R2019b) Check Simulink signal appearance	mathworks.maab.db_0032	mathworks.jmaab.db_0032
Check connections between structural subsystems (R2019b) Check usage of Goto and From blocks between Subsystems	mathworks.maab.jc_0171	mathworks.jmaab.jc_0171
Check trigger signal names (R2019b) Check Trigger and Enable block names	mathworks.maab.jc_0281	mathworks.jmaab.jc_0281
Check usage of vector and bus signals (R2019b) Check usage of buses and Mux blocks	mathworks.maab.na_0010	mathworks.jmaab.na_0010
Check position of conditional blocks and iterator blocks (R2019b) Check position of Trigger and Enable blocks	mathworks.maab.db_0146	mathworks.jmaab.db_0146
Check fundamental logical and numerical operations	mathworks.maab.na_0002	mathworks.jmaab.na_0002
Check usage of Sum blocks	mathworks.maab.jc_0121	mathworks.jmaab.jc_0121
Check position of Inport and Outport blocks	mathworks.maab.db_0042	mathworks.jmaab.db_0042

Model Advisor Check Name	R2019b Check ID	R2020a Check ID
Check for missing ports in Variant Subsystems (R2019b) Check unused ports in Variant Subsystems	mathworks.maab.na_0020	mathworks.jmaab.na_0020
Check for Strong Data Typing with Simulink I/O	mathworks.maab.db_0122	mathworks.jmaab.db_0122
Check definition of Stateflow data (R2019b) Check Stateflow data objects with local scope	mathworks.maab.db_0125	mathworks.jmaab.db_0125
Check for Stateflow transition appearance	mathworks.maab.db_0129	mathworks.jmaab.db_0129
Check default transition placement in Stateflow charts	mathworks.maab.jc_0531	mathworks.jmaab.jc_0531
Check for use of C-style comment symbols	mathworks.maab.jc_0801	mathworks.jmaab.jc_0801
Check for usage of events and broadcasting events in Stateflow charts (R2019b) Check for event broadcasts in Stateflow charts	mathworks.maab.jm_0012	mathworks.jmaab.jm_0012
Check for MATLAB expressions in Stateflow charts	mathworks.maab.db_0127	mathworks.jmaab.db_0127
Check Stateflow operators (R2019b) Check for bitwise operations in Stateflow charts	mathworks.maab.na_0001	mathworks.jmaab.na_0001

This table identifies the Model Advisor checks that are no longer available in the Model Advisor because the corresponding modeling guideline is removed in R2020a. You will receive a warning when you run these checks under the following circumstances:

- When you load/save a configuration that includes these checks: Model Advisor checks for MAAB and JMAAB are updated to support the MAB modeling guidelines. For more information, see the R2020a Release Notes.
- When you run the checks by using `ModelAdvisor.run()` on the MATLAB command line:

In R2020a, MAAB modeling guidelines were updated and renamed to the MAB modeling guidelines. As a result, the guideline for check “<check name” (ID: <check ID>) was removed and the check is no longer needed to verify compliance.

For more information about modeling guidelines that were removed from the MAB 5.0 guidelines, see Support for MathWorks Advisory Board (MAB) 5.0 modeling guidelines

Model Advisor Check	Check ID
Check orientation of Subsystem blocks	mathworks.maab.jc_0111
Check character usage in signal labels	mathworks.maab.jc_0221
Check reuse of variables within a Stateflow scope	mathworks.maab.jc_0491
Check usage of return values from Stateflow graphical functions	mathworks.maab.jc_0521
Check for prohibited blocks in discrete controllers	mathworks.maab.jm_0001
Check for matching port and signal names	mathworks.maab.jm_0010
Check visibility of block port names	mathworks.maab.na_0005
Check for comparison operations in Stateflow charts	mathworks.maab.na_0013
Check usage of non-compliant blocks	mathworks.maab.na_0027
Check Simulink bus signal names	mathworks.maab.na_0030
Check usage of merge blocks	mathworks.maab.na_0032
Check transition actions in Stateflow charts	mathworks.maab.db_0151
Check nested states in Stateflow charts	mathworks.maab.na_0038
Check number of Stateflow states per container	mathworks.maab.na_0040

Automate checking of models to comply with JMAAB 5.1 modeling guidelines

The following table identifies the new and updated checks to the JMAAB group for verifying compliance with JMAAB 5.1.

To execute these checks, open Model Advisor and select **By Task > Modeling Standards for JMAAB**.

Model Advisor Check	Check ID
Check signal line connections	mathworks.jmaab.db_0032
Check position of Inport and Outport blocks	mathworks.jmaab.db_0042
Check Model font settings	mathworks.jmaab.db_0043
Check Indexing Mode	mathworks.jmaab.db_0112
Check definition of Stateflow data	mathworks.jmaab.db_0125
Check definition of Stateflow events	mathworks.jmaab.db_0126
Check for MATLAB expressions in Stateflow charts	mathworks.jmaab.db_0127
Check for Stateflow transition appearance	mathworks.jmaab.db_0129
Check usage of exclusive and default states in state machines	mathworks.maab.db_0137
Check position of conditional blocks and iterator blocks	mathworks.jmaab.db_0146

Model Advisor Check	Check ID
Check usage of Sum blocks	mathworks.jmaab.jc_0121
Check for usage of Data Store Memory blocks	mathworks.jmaab.jc_0161
Check connections between structural subsystems	mathworks.jmaab.jc_0171
Check entry formatting in State blocks in Stateflow charts	mathworks.maab.jc_0501
Check for cascaded Unit Delay blocks	mathworks.jmaab.jc_0624
Check condition actions and transition actions in Stateflow	mathworks.jmaab.jc_0753
Check duplication of Simulink data names	mathworks.jmaab.jc_0791
Check for usage of events and broadcasting events in Stateflow charts	mathworks.jmaab.jm_0012
Check Stateflow operators	mathworks.jmaab.na_0001
Check fundamental logical and numerical operations	mathworks.jmaab.na_0002
Check usage of vector and bus signals	mathworks.jmaab.na_0010
Check logical expressions in If blocks	mathworks.maab.na_0003
Check scope of From and Goto blocks	mathworks.maab.na_0011
Check for missing ports in Variant Subsystems	mathworks.jmaab.na_0020
Check usage of character vector inside MATLAB Function block	mathworks.maab.na_0021
Check usage of enumerated values	mathworks.maab.na_0031
Check use of default variants	mathworks.maab.na_0036
Check use of single variable variant conditionals	mathworks.maab.na_0037
Check use of Simulink in Stateflow charts	mathworks.maab.na_0039
Check usage of Simulink function in Stateflow	mathworks.jmaab.na_0042
Check Subsystem names	mathworks.jmaab.jc_0201

Model Advisor checks for verifying compliance with DO-254 safety standards

In R2020a, Model Advisor introduces checks that verify compliance with DO-254 safety standards. To check compliance with the DO-254 standards, open the Model Advisor and run the checks in **By Task > Modeling Standards for DO-254**. For more information, see Model Checks for DO-254 Standard Compliance.

The following table lists the DO-254 checks:

DO-254 Checks
Display model version information
Identify disabled library links

DO-254 Checks
Identify parameterized library links
Identify unresolved library links
Check for model reference configuration mismatch
Identify requirement links that specify invalid locations within documents
Identify requirement links with missing documents
Identify requirement links with path type inconsistent with preferences
Identify selection-based links having descriptions that do not match their requirements document text

The following table lists the High Integrity System Model checks that are supported by DO-254 safety standards:

High Integrity System Model Checks
Check for inconsistent vector indexing methods
Check for variant blocks with 'Generate preprocessor conditionals' active
Check for root Inports with missing properties
Check for Relational Operator blocks that equate floating-point types
Check usage of Relational Operator blocks
Check usage of Logical Operator blocks
Check sample time-dependent blocks
Check safety-related block reduction optimization settings
Check usage of Abs blocks
Check usage of Assignment blocks
Check for root Inports with missing range definitions
Check for root Outports with missing range definitions
Check Stateflow charts for transition paths that cross parallel state boundaries
Check Stateflow charts for ordering of states and transitions
Check Stateflow debugging options
Check Stateflow charts for uniquely defined data objects
Check usage of shift operations for Stateflow data
Check Stateflow charts for unary operators
Check for Strong Data Typing with Simulink I/O
Check MATLAB Code Analyzer messages
Check safety-related model referencing settings
Check safety-related diagnostic settings for parameters
Check safety-related diagnostic settings for type conversions
Check safety-related diagnostic settings for signal connectivity
Check safety-related diagnostic settings for bus connectivity

High Integrity System Model Checks
Check safety-related diagnostic settings for model initialization
Check safety-related diagnostic settings for model referencing
Check safety-related diagnostic settings for saving
Check safety-related diagnostic settings for Stateflow
Check model object names
Check for model elements that do not link to requirements
Check for inappropriate use of transition paths
Check usage of Bitwise Operator block
Check data types for blocks with index signals
Check model file name
Check if/elseif/else patterns in MATLAB Function blocks
Check switch statements in MATLAB Function blocks
Check global variables in graphical functions
Check for length of user-defined object names
Check usage of conditionally executed subsystems
Check usage of standardized MATLAB function headers
Check usage of relational operators in MATLAB Function blocks
Check usage of equality operators in MATLAB Function blocks
Check usage of logical operators and functions in MATLAB Function blocks
Check naming of ports in Stateflow charts
Check scoping of Stateflow data objects
Check usage of Gain blocks
Check usage of bitwise operations in Stateflow charts
Check data type of loop control variables

The following table lists the HDL Coder checks that are supported by DO-254 safety standards:

HDL Code Advisor Checks
Check for infinite and continuous sample time sources (HDL Coder)
Check for unsupported blocks (HDL Coder)
Check for large matrix operations (HDL Coder)
Identify unconnected lines, input ports, and output ports (Simulink)
Identify disabled library links (Simulink)
Identify unresolved library links (Simulink)
Check for MATLAB Function block settings (HDL Coder)
Check for Stateflow chart settings (HDL Coder)
Check Delay, Unit Delay and Zero-Order Hold blocks for rate transition (Simulink)
Check for unsupported storage class for signal objects (HDL Coder)

HDL Code Advisor Checks
Check VHDL file extension (HDL Coder)
Check naming conventions (HDL Coder)
Check top-level subsystem/port names (HDL Coder)
Check module/entity names (HDL Coder)
Check signal and port names (HDL Coder)
Check package file names (HDL Coder)
Check generics (HDL Coder)
Check clock, reset, and enable signals (HDL Coder)
Check architecture name (HDL Coder)
Check entity and architecture (HDL Coder)
Check clock settings (HDL Coder)
Check model for foreign characters (Simulink)
Check for global reset setting for Xilinx and Altera devices (HDL Coder)
Check inline configurations setting (HDL Coder)
Check algebraic loops (HDL Coder)
Check for visualization settings (HDL Coder)
Check delay balancing setting (HDL Coder)
Check for safe model parameters (HDL Coder)
Check for double datatypes in the model with Native Floating Point (HDL Coder)
Check for Data Type Conversion blocks with incompatible settings (HDL Coder)
Check for HDL Reciprocal block usage (HDL Coder)
Check for Relational Operator block usage (HDL Coder)
Check for unsupported blocks with Native Floating Point (HDL Coder)
Check for blocks with nonzero output latency (HDL Coder)
Check blocks with nonzero ulp error (HDL Coder)
Check for single datatypes in the model (HDL Coder)
Check initial conditions of enabled and triggered subsystems (HDL Coder)
Check for invalid top level subsystem (HDL Coder)

Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines

This table identifies the new and updated Model Advisor checks that verify compliance with High-Integrity System Modeling guidelines.

Model Advisor Check	Check ID	Description of Change
Check safety-related diagnostic settings for model referencing	mathworks.hism.hisl_0310	<p>In R2020a, configuration parameter Model block version mismatch is removed from the check.</p> <p>The parameter has been removed from corresponding guideline hisl_0310: Configuration Parameters > Diagnostics > Model Referencing (Simulink).</p>

Customize your Model Advisor by using the redesigned configuration editor

In R2020a, the Model Advisor Configuration Editor user interface is redesigned to provide a more flexible and streamline way for you to create custom Model Advisor configurations and modify existing Model Advisor configurations.

To open the Model Advisor Configuration Editor from the Simulink toolstrip:

- In the **Model Advisor** gallery of the **Modeling** tab, select **Model Advisor Configuration Editor**.
- In the **Modeling** tab, select **Model Advisor**. From the Model Advisor dialog, select **Settings > Open Configuration Editor**.

For more information about creating a custom configuration, see Use the Model Advisor Configuration Editor to Customize the Model Advisor

Format of the Model Advisor Configuration File

Previously, Model Advisor configuration files were saved as .mat files. Starting in R2020a, configuration files are saved as .json files.

Warning Message for Missing Checks When Loading Configuration

Previously, you were automatically warned of checks that were missing in the Model Advisor when loading the Model Advisor configuration. In R2020a, you can suppress this message by highlighting the top-level folder in the check hierarchy and selecting the **Suppress warning messages for missing checks when loading configuration** button in the Model Advisor Configuration Editor.

To programmatically suppress the warning for Model Advisor configurations, in the MATLAB command line, enter
`warning('off', 'Simulink:tools:MALoadConfigMissCorrespondCheck')`.

For more information, see Suppress Warning Message for Missing Checks.

Specify Results Severity for Failed Checks

You can now specify whether you want the check to be marked as a warning or failure when the check flags an issue in your model. In the Model Advisor Configuration Editor, from the **Check result when issues are flagged** list, select Warning or Fail. For more information, see Customize the Model Advisor Configuration.

For custom Model Advisor checks, you can specify the result severity for the check by using the new `ModelAdvisor.Check.ErrorSeverity` property.

Programmatically Set the Default Configuration

You can use function `ModelAdvisor.setDefaultConfiguration` to programmatically set the default configuration that is used by the Model Advisor. Using these methods is equivalent to selecting **Set as Default** in the Model Advisor Configuration Editor. For more information, see [Specify a Default Configuration File](#).

Removal of the `AggregateComponentDetails` property

In R2020a, the `slmetric.metric.Metric` class Boolean property `AggregateComponentDetails` has been removed.

Compatibility Considerations

For custom metrics, the removal of this property means that you can no longer specify whether to aggregate noncomponent values to the parent component. Remove the `AggregateComponentDetails` property from your metric algorithm functions. Noncomponents are Simulink objects that are not:

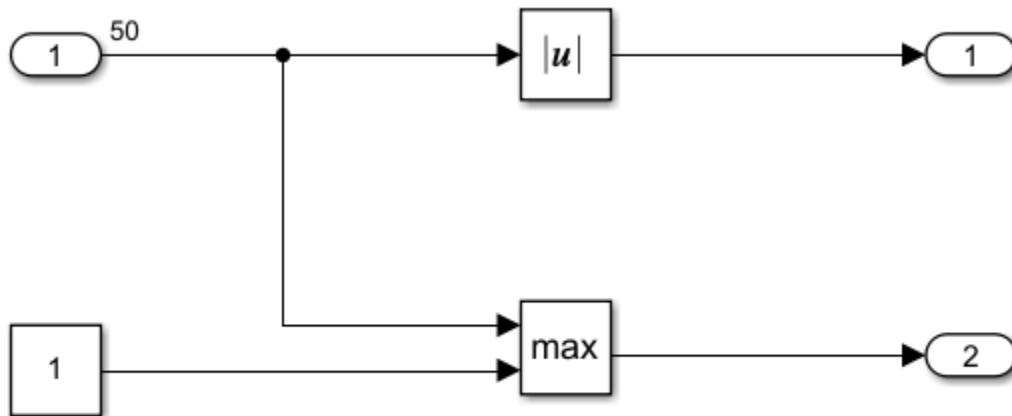
- Models
- Subsystem blocks
- Charts
- MATLAB Function blocks
- Protected models
- Library blocks

Because of this change, the cyclomatic complexity metric no longer includes chart states in its calculation. For more information, see [Cyclomatic complexity metric](#).

Enhanced calculation of cyclomatic complexity

In R2019b, the cyclomatic complexity metric calculation counted each element of a vectorized operation as a separate decision point. As a result, for a simple model with a signal that is a vector or matrix, the Metrics Dashboard might have reported a cyclomatic complexity much higher than 30—the threshold value between the compliant and warning range. In R2020a, the cyclomatic complexity metric calculation considers a vectorized operation as a single decision point, thereby lowering the cyclomatic complexity value for some models.

For example, consider the following simple model, which has a vector signal of size 50. This signal branches to two signals. Two blocks perform calculations on each signal.



In R2019b, this model had a cyclomatic complexity of 101. In R2020a, this model has a cyclomatic complexity of 3. This is a more reasonable value for a simple model. For more information, see [Cyclomatic complexity metric](#).

Compatibility Considerations

Previously, models with vectorized operations had higher cyclomatic complexity values than they have in R2020a.

Removal of restore point from the Model Advisor

Prior to 2020a, you could use the Model Advisor to save a restore point, which is a snapshot in time of your model, base workspace, and Model Advisor. You could then load the restore point to revert changes that you made to your model in response to the check results.

In R2020a, the ability to save and load a restore point by using these Model Advisor menu options has been removed:

- **File > Save Restore Point As**
- **File > Save Restore Point**
- **File > Load Restore Point**

Removal of the Model Advisor Results Advisor View

Some Model Advisor checks have an **Explore Results** button that opens the Model Advisor Result Explorer. The Model Advisor Results Explorer allows you to quickly locate, view, and change elements of a model. In R2020a, **Explore Results** has been removed from the checks.

Compatibility Considerations

You can use the hyperlinks in the Model Advisor results to view and modify model elements that are being flagged by the Model Advisor check. For more information, see [Address Model Check Results \(Simulink\)](#). You can also apply batch changes to model parameters from the command line using the `set_param` command.

For custom checks, set the `ModelAdvisor.Check.CallbackStyle` property to `'DetailStyle'`.

New Customizing Model Advisor Example

In R2020a, Simulink Check contains a new model and example that demonstrate how to author custom Model Advisor checks and deploy a custom Model Advisor configuration. The name of the new model is `AdvisorExampleCustomization.slx`. This model replaces the previous `slvndemo_mdadv` model. To access the example, model and supporting files, see [Create and Deploy a Model Advisor Custom Configuration](#).

R2019b

Version: 4.4

New Features

Bug Fixes

Compatibility Considerations

Refactor models using clone detection

In R2019b, to detect clones in your model, use the new Clone Detector app. This app is available in the **Apps** tab in the Simulink Toolstrip. The Clone Detector helps you to refactor your model and replace the clones with library links.

The Clone Detector interface steps you through the process of identifying and refactoring clones. You can:

- Access clone reuse percentages in the model to determine the benefits of refactoring.
- Vary parameter tolerance to identify similar and exact clones.
- Identify the differences in a clone from the baseline subsystem.
- Access a log of clone detection results.
- Use a Simulink Test license to access an embedded Test Manager that allows you to verify the equivalence of the refactored model and the original model.

For more information, see [Clone Detector and Enable Component Reuse by Using Clone Detection](#).

Model Advisor checks for verifying compliance with High-Integrity System Modeling guidelines

This table identifies the new and updated Model Advisor checks that verify compliance with High-Integrity System Modeling guidelines.

Model Advisor Check	Check ID	Description of Change
Check type and size of conditional expressions	mathworks.hism.himl_0011	New check that verifies the use of logical scalars for data type and size in condition expressions.
Check safety-related code generation identifier settings	mathworks.hism.hisl_0049	In R2019b, the configuration parameters pane Code Generation > Symbols is renamed as Code Generation > Identifiers . The Model Advisor check was updated to reflect this change.

Check blocks not supported for code generation by using edit-time checks

In R2019b, when you open the C Code app (available with Embedded Coder® or Simulink Coder™), you can specify whether you want to view modeling issues for code generation on the canvas while editing by selecting the **C/C++ Code Advisor > Edit-Time Checks** box.

For a GRT-based targets, edit-time checking identifies blocks not supported for code generation.

For ERT-based targets, edit-time checking identifies:

- Blocks not supported for code generation

- Blocks not recommended for code generation

Note In R2019a, violations of the ERT-based edit-time check were flagged automatically without the edit-time checks functionality being enabled. In R2019b, you must enable edit-time checks to view these violations on the canvas.

For more information, see:

- Check Your Model by Using Edit Time Checks
- Blocks and Products Supported for Code Generation (Embedded Coder)
- Configure a System Target File (Embedded Coder)
- Embedded Coder app

JMAAB 5.1 Support: Automate checking of models to comply with JMAAB 5.1 modeling style guidelines

The following table identifies the new and updated checks to the JMAAB group for verifying compliance with JMAAB 5.1. For more information, see Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance.

To execute these checks, open Model Advisor and select **By Task > Modeling Standards for JMAAB**.

Model Advisor Check	Check ID
Check file names	mathworks.jmaab.ar_0001
Check folder names	mathworks.jmaab.ar_0002
Check subsystem names	mathworks.jmaab.jc_0201
Check port block names	mathworks.jmaab.jc_0211
Check character usage in block names	mathworks.jmaab.jc_0231
Check trigger signal names	mathworks.jmaab.jc_0281
Check for consistency in model element names	mathworks.jmaab.jc_0602
Check Model Description	mathworks.jmaab.jc_0603
Check operator order of Product blocks	mathworks.jmaab.jc_0610
Check for sample time setting	mathworks.jmaab.jc_0641
Check output data type of operation blocks	mathworks.jmaab.jc_0651
Check usable number for first index	mathworks.jmaab.jc_0701
Check usage of numeric literals in Stateflow	mathworks.jmaab.jc_0702
Check usage of transitions to external states	mathworks.jmaab.jc_0723
Check order of state action types	mathworks.jmaab.jc_0733
Check indentation of code in Stateflow states	mathworks.jmaab.jc_0736
Check usage of Stateflow comments	mathworks.jmaab.jc_0738

Model Advisor Check	Check ID
Check if state action type 'exit' is used in the model	mathworks.jmaab.jc_0740
Check usage of internal transitions in Stateflow states	mathworks.jmaab.jc_0763
Check placement of Label String in Transitions	mathworks.jmaab.jc_0770
Check position of comments in transition labels	mathworks.jmaab.jc_0771
Check usage of unconditional transitions in flow charts	mathworks.jmaab.jc_0773
Check for comments in unconditional transitions	mathworks.jmaab.jc_0774
Check terminal junctions in Stateflow	mathworks.jmaab.jc_0775
Check unused data in Simulink Model	mathworks.jmaab.jc_0792
Check for use of C-style comment symbols	mathworks.jmaab.jc_0801
Check for implicit type casting in Stateflow	mathworks.jmaab.jc_0802
Check diagnostic settings for incorrect calculation results	mathworks.jmaab.jc_0806

Access Simulink Check capabilities from Simulink Toolstrip

In R2019b, these Simulink Check capabilities are in the **Apps** tab, in the **Verification, Validation, and Test** section:

- Model Slicer
- Metrics Dashboard
- Clone Detector
- Model Transformer

The **Modeling** tab contains a **Model Advisor** gallery where you can access the Model Advisor Configuration Editor.

Collect metric data for referenced models running in accelerated mode

Previously, you could collect metric data for only referenced models running in normal simulation mode. On the Metrics Dashboard, you collected this data by selecting **Options > Include Model References**.

In R2019b, you can collect metric data for referenced models running in accelerated mode. To collect these metrics, open the Metrics Dashboard and select the new **Options > Include referenced models (all modes)** button. This setting is on by default. There are two new buttons for collecting metrics for referenced models running only in normal simulation mode or to exclude referenced models from the analysis.

In R2019b, you can programmatically collect metric data for referenced models running in accelerated mode. For an `slmetric.Engine` object, you collect this data by setting the new `ModelReferencesSimulationMode` property to `AllModes`. This new property has these settings.

Setting	Description
None	Metric engine does not collect metric data for referenced models.
NormalModeOnly	Metric engine collects metric data only for referenced models running in normal simulation mode.
AllModes	Metric engine collects metric data for referenced models running in normal and accelerated simulation modes.

Compatibility Considerations

Previously, to collect metric data for referenced models, you set the `AnalyzeModelReferences` property to 1 (default). In R2019b, you get a warning stating that this property will be removed in a future release. To avoid getting this warning, use the new `ModelReferencesSimulationMode` property.

For more information, see [Collect Model Metric Data by Using the Metrics Dashboard and `slmetric.Engine`](#).

Enhancement to Model Advisor compliance metrics

In R2019a, the Model Advisor Check Issues for High-Integrity Systems metric (`mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178`) and the Model Advisor Check Issues for MAAB Standards metric (`mathworks.metrics.ModelAdvisorCheckIssues.maab`) did not count model advisor checks that contained warnings or failures if these checks did not contain a link to a block or signal in the model. In R2019b, these metrics count each Model Advisor check that produces a warning or failure. If a check contains links to blocks, this metric counts one issue for each linked block. Checks with links to the model are highlighted in the Simulink Editor. For more information, see [Model Metrics and Collect and Explore Metric Data by Using the Metrics Dashboard](#).

R2019a

Version: 4.3

New Features

Bug Fixes

Model Slicer available with Simulink Check

Previously, the Model Slicer was available with Simulink Design Verifier. In R2019a, the tool is available with Simulink Check. Use the Model Slicer to determine the interdependencies of blocks, signals, and model components throughout a model. Also, use Model Slicer to create simplified standalone models that are easier to understand and analyze yet retain their original context. To access the Model Slicer, select **Analysis > Model Slicer**. For more information, see Model Simplification with Dependency Analysis.

Hierarchical view of metrics dashboard results

In R2019a, you can view metric data for related components together by viewing Metric Dashboard results in a model hierarchical view. You can then better understand aggregated model metric data. To use the model hierarchical view, run the Metrics Dashboard on your model. Click a widget and then click the new **Tree** button. The figure shows metric results for the model `sf_car` in the Tree view.

Cyclomatic complexity
Metric that calculates the cyclomatic complexity for model, subsystems and charts.

COMPONENT	TYPE	MODEL COMPLEXITY	MODEL COMPLEXITY (INCL. DESCENDANTS)
▼ sf_car	Model	✔ 1	32
▼ transmission	SubSystem	✔ 0	5
Torque Converter	SubSystem	✔ 0	0
▼ transmission ratio	SubSystem	✔ 0	5
Look-Up Table	MATLABFunction	✔ 5	5
Vehicle	SubSystem	✔ 0	0
▼ shift_logic	Chart	✔ 20	26
▼ selection_state.calc_th	SubSystem	✔ 1	6
Look-Up	MATLABFunction	✔ 5	5
Engine	SubSystem	✔ 0	0

For more information on the Metrics Dashboard, see [Collect and Explore Metric Data by Using the Metrics Dashboard](#).

Filters for metrics dashboard results

In R2019a, you can filter Metrics Dashboard results by component type, name, and path. After you run the Metrics Dashboard, to drill into a widget's metric details, select that widget. In the **Table** view, select the context menu on the right side of the **TYPE**, **COMPONENT**, and **PATH** column headers. From the **TYPE** menu, select applicable components. From the **COMPONENT** and **PATH** menus, type a component name or path in the search bar. The Metrics Dashboard saves the filters for a widget, so you can view metric details for other widgets and return the filtered results. You cannot specify filters on the **High Integrity** and **MAAB** compliance widgets.

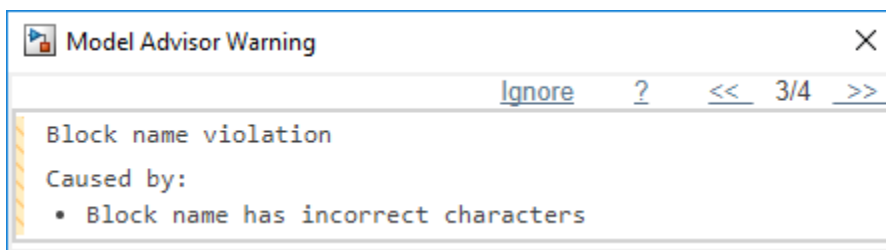
Filters enable you to quickly locate results. For example, to locate results for only Stateflow Charts, you can select that component from the **TYPE** menu. To analyze results for one Stateflow Chart, type its name or path in the **COMPONENT** or **PATH** search bar. For more information, see [Collect Model Metric Data by Using the Metrics Dashboard](#).

Streamline compliance with modeling guidelines by using enhanced edit-time check diagnostics

Edit-Time check diagnostics interface

Enhancements to the edit-time checking capability make it easier for you to identify and resolve compatibility issues earlier in the model design process.

In R2019a, when you click the warning icon that appears over the highlighted block, the Model Advisor identifies compliance issues in the block that violate edit-time checks. When a block has multiple check violations, you can move between the violations by using the << and >> buttons. The cause of each issue is provided.



For information on checking your model while you edit, see [Check Model Compliance by Using the Model Advisor](#).

Model Advisor checks evaluated by using edit-time checking

In R2019a, when you use edit-time checking, you can view violations of these Model Advisor checks. See the check documentation for additional information and limitations.

- Check usage of non-compliant blocks
- Check length of subsystem names
- Check length of Inport and Outport names
- Check length of block names
- Check operator order of Product blocks
- Check icon shape of Logical Operator blocks
- Check settings for data ports in Multiport Switch blocks
- Check usage of Switch blocks
- Check for unsupported block names
- Check for blocks not recommended for MISRA C:2012
- Check for switch case expressions without a default case
- Check usage of Assignment blocks
- Check for blocks not recommended for secure coding standards
- Identify lookup table blocks that generate expensive out-of-range checking code

JMAAB 5.1 Support: Automate checking of models to comply with JMAAB 5.1 modeling style guidelines

The JMAAB check group has been updated to support the latest release of the Japan MATLAB Automotive Advisory Board (JMAAB) guidelines 5.1. New checks have been added to support new guideline rules and removed when no longer applicable.

The following table identifies the new checks added to the JMAAB group for verifying compliance JMAAB 5.1. For more information, see Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance.

To execute these checks, open Model Advisor and select **By Task > Modeling Standards for JMAAB**.

Model Advisor Check	Check ID
Check default transition placement in Stateflow charts (JMAAB)	mathworks.jmaab.jc_0531
Check duplication of Simulink data names	mathworks.jmaab.jc_0791
Check for avoiding algebraic loops between subsystems	mathworks.jmaab.jc_0653
Check for unconnected objects in Stateflow Charts	mathworks.jmaab.jc_0797
Check for usage of text inside states	mathworks.jmaab.jc_0739
Check length of Stateflow data names	mathworks.jmaab.jc_0796
Check placement of Label String in Transitions	mathworks.jmaab.jc_0770
Check settings for data ports in Multiport Switch blocks	mathworks.jmaab.jc_0630
Check Stateflow chart action language	mathworks.jmaab.jc_0790
Check usable characters for Stateflow data names	mathworks.jmaab.jc_0795
Comparing floating point types in Simulink	mathworks.jmaab.jc_0800
Check signal line labels (JMAAB)	mathworks.jmaab.jc_0008
Check for propagated signal labels	mathworks.jmaab.jc_0009

The following table identifies checks that are no longer applicable to JMAAB 5.1 and have been removed from the JMAAB check group.

Check ID	Model Advisor Check
mathworks.jmaab.jc_0735	Check if each action in state label ends with a semicolon
mathworks.jmaab.jc_0737	Check if operators (ActionLanguage) for States and/or Transitions in Stateflow have uniform spaces around them.
mathworks.jmaab.jc_0742	Guidelines for writing boolean operations in condition labels in Stateflow transitions

Check ID	Model Advisor Check
mathworks.jmaab.jc_0743	Guidelines for writing condition actions in Stateflow transitions
mathworks.jmaab.jc_0756	Prohibited use of operation expressions in array indices
mathworks.maab.db_0151	Check transition actions in Stateflow charts
mathworks.maab.hd_0001	Check for prohibited sink blocks
mathworks.maab.himl_0003	Check MATLAB Function metrics
mathworks.maab.jc_0111	Check orientation of Subsystem blocks
mathworks.maab.jc_0221	Check character usage in signal labels
mathworks.maab.jm_0001	Check for prohibited blocks in discrete controllers
mathworks.maab.na_0008	Check signal line labels
mathworks.maab.na_0009	Check for propagated signal labels
mathworks.maab.na_0013	Check for comparison operations in Stateflow charts
mathworks.maab.jc_0281	Check Trigger and Enable block names

Additional MAAB 3.0 Checks: Improve quality and compliance to guidelines

This table identifies the new checks for verifying compliance with MATLAB Automotive Advisory Board Checks 3.0 (MAAB) guidelines. For information about MAAB guidelines, see Model Advisor Checks for MAAB Guidelines.

Model Advisor Check	Check ID
Check Simulink signal appearance	mathworks.maab.db_0032
Check for Stateflow transition appearance	mathworks.maab.db_0129
Check usage of Goto and From blocks between Subsystems	mathworks.maab.jc_0171
Check reuse of Variables within a Stateflow scope	mathworks.maab.jc_0491
Check usage of non-compliant blocks	mathworks.maab.na_0027
Check usage of enumerated values	mathworks.maab.na_0031

For more information, see Model Checks for MathWorks Automotive Advisory Board (MAAB) Guideline Compliance.

Added functions for edit-time checking

In R2019a, there are two new edit-time checking functions. Use the new `edittime.setAdvisorChecking` method to check for Model Advisor violations while you edit. The `edittime.setAdvisorChecking` method is equivalent to selecting **Analysis > Model Advisor > Display Advisor Checks in Editor**. Use the new `edittime.getAdvisorChecking` method to find out if Edit-time checking is on or not. For more information on edit-time checking, see Check Model Compliance by Using the Model Advisor.

Model optimization by sharing prelookup operation of n-D lookup tables

In R2019a, you can use the Model Transformer tool to improve the simulation speed for models with n-D Lookup table blocks by reusing the Prelookup operation with multiple blocks. This transformation results in decreased ROM usage and increased execution speed in the generated code.

You can run the **Transform table lookup into prelookup and interpolation** check on the Model Transformer tool to convert models.

A limitation is that the model transformation is confined to a single subsystem and does not extend beyond the subsystem boundaries. For further details, refer to [Improve Efficiency of Simulation by Optimizing Prelookup Operation of Lookup Table Blocks](#)

Clone metrics include exact and similar clones

Previously, when you ran the Metrics Dashboard on a model, the tool reported only exact clones in the **Potential Reuse** widget. Exact clones have identical block types, connections, and parameter values. When you clicked **Open Conversion Tool**, the Clone Detection tool ran and reported only exact clones because the **Maximum number of different parameters** parameter was set to 0 (default value). The Clone content and the Clone detection metrics included only exact clones as part of their calculations.

In R2019a, the Metrics Dashboard reports exact and similar clones in the **Potential Reuse** widget. Similar clones have identical block types and connections, but they can have different block parameter values. When you click **Open Conversion Tool**, the Clone Detection tool also reports exact and similar clones because the **Maximum number of different parameters** parameter is set to a high value. The Clone content and the Clone detection metrics include exact and similar clones as part of their calculations. For more information on the Metrics Dashboard, see [Collect Model Metric Data by Using the Metrics Dashboard](#).

Model slicer support for multi-instance model reference

You can now use model slicer on a Simulink model that contains multiple references to the same model in normal simulation mode.

Model Advisor option to compile model for code generation

In R2019a, the value 'PostCompileForCodegen' is added to the `ModelAdvisor.Check.CallbackContext` property. Use this option when developing custom Model Advisor checks to ensure code generation readiness of the model.

When using the existing 'PostCompile' option, the Model Advisor updates the model diagram and simulates the model. These Model Advisor checks do not flag modeling issues that fail during code generation because these issues do not affect the simulated model.

With the 'PostCompileForCodegen' option, the Model Advisor updates the model diagram specifically for code generation and does not assume that the model is being simulated. As a result, custom Model Advisor checks can identify code generation setup issues in a model at an earlier stage, avoiding unexpected errors during code generation.

The 'PostCompileForCodegen' option compiles for all variant choices in a model. This enables you to analyze possible issues present in the generated code for both active and inactive variant paths in the model. To compile for all variant choices, on the Variant system block, select the Analyze all choices during update diagram and generate preprocessor conditionals parameter.

Using the 'PostCompileForCodegen' option can increase the amount of time to execute your Model Advisor checks.

For additional information, see:

- ModelAdvisor.Check.CallbackContext
- Define the Compile Option for Custom Checks
- Variant Systems (Simulink)

Updates for verifying compliance with High-Integrity Systems Modeling guidelines

This table identifies the updates to the checks that verify compliance with High-Integrity Systems Modeling guidelines.

Model Advisor Check	Check ID	Description of Change
Check usage of relational operators in MATLAB Function blocks	mathworks.hism.himl_0008	Flags the MATLAB relational operators that are used in functional format. For example:
Check usage of equality operators in MATLAB Function blocks	mathworks.hism.himl_0009	<ul style="list-style-type: none"> • lt - Less than • ne - not equal to
Check usage of Bitwise Operator block	mathworks.hism.hisl_0019	Checks the Bitwise Operations for Signed data types .
Check usage of bitwise operations in Stateflow charts	mathworks.hism.hisf_0003	
Check Stateflow debugging options	mathworks.hism.hisf_0011	Checks the parameter Underspecified and Overspecified in the Settings menu of the Truth Table Editor , to Error .

Model Advisor Check	Check ID	Description of Change
Check usage of conditionally executed subsystems	mathworks.hism.hisl_0012	Extends support on the following sample-time dependent blocks: <ul style="list-style-type: none"> • Discrete Filter • Discrete Fir • Discrete State Space • Discrete Transfer Fcn • Discrete Zero Pole • Discrete Integrator • First Order Transfer Fcn • Lead or Lag Compensator • Transfer Fcn Real Zero • Discrete Derivative • Discrete Transfer Function with Initial Outputs • Discrete Transfer Function with Initial States • Discrete Zero-Pole with Initial Outputs • Discrete Zero-Pole with Initial States
Check usage of Merge blocks	mathworks.hism.hisl_0015	Checks the Outport block parameter Output when disabled to held for each conditionally executed subsystem being merged.

Model Advisor Check	Check ID	Description of Change
Check for inconsistent vector indexing methods	mathworks.hism.hisl_002 1	<p>Extends support on the following:</p> <p>Blocks that support configurable indexing:</p> <ul style="list-style-type: none"> • Index Vector • Multiport Switch • Assignment • Selector • For Iterator <p>Blocks that support only one-based indexing:</p> <ul style="list-style-type: none"> • Fcn • MATLAB Function • MATLAB System • State Transition Table • Test Sequence • Truth Table block • Stateflow chart with MATLAB action language • Truth Table function with MATLAB action language <p>Blocks that supports only zero-based indexing:</p> <ul style="list-style-type: none"> • Stateflow chart with C action language • Truth Table function with C action language
Check usage of shift operations for Stateflow data	mathworks.hism.hisf_006 4	Does not check the negative values.

For more information, see Model Checks for High Integrity Systems Modeling Checks.

MISRA C:2012 and Secure Coding checks to improve compliance of generated code

Modifications to existing Model Advisor checks that you use to verify compliance with MISRA C:2012 and Secure Coding standards are outlined in this table.

Model Advisor Check	Description of Change
Check configuration parameters for MISRA C:2012	Checks now analyze the setting for these configuration parameters:
Check configuration parameters for secure coding standards	<ul style="list-style-type: none"> • Include comments (GenerateComments) • MATLAB user comments (MATLABFcnDesc)
Check for missing error ports for AUTOSAR receiver interfaces	<p>When an error port is missing, the check flags receiver interface ports with these AUTOSAR data access mode types:</p> <ul style="list-style-type: none"> • ImplicitReceive • ExplicitReceive • EndToEndRead

Tech Preview of model refactoring using clone detection

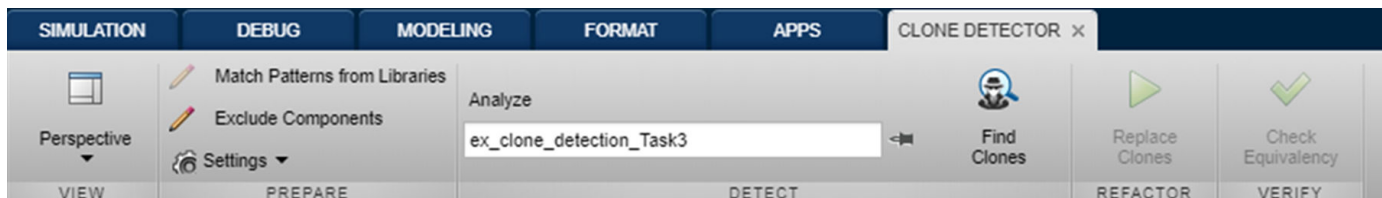
In R2019a, you can use the Clone Detector app that has a new and improved user interface. This user interface facilitates:

- Display of clones detected hierarchically as model view, tree view, list view, and clones plot.
- Displays the report from the clone refactorization and enables you to restore the model to its original state.

The Clone Detector app is a technical preview in R2019a. To try it out, turn on the Simulink Toolstrip. For more information, see “Simulink Toolstrip Tech Preview replaces menus and toolbars in the Simulink Desktop”.

The documentation does not reflect the addition of the Clone Detection app. There is documentation for the existing Clone Detection tool. For more information, see Enable Component Reuse by Using Clone Detection.

The new user interface contains 5 distinct sections representing the workflow as shown below :



To access the parameter settings for help, results, and model properties of the Clone Detector app, click the **Perspective** menu.

You can use the settings present on the **PREPARE** tab to set the conditions for clone detection.

To identify clones in the model, click the **Find Clones** icon. The percentage of overall, exact, and similar clones that can be replaced by library blocks is displayed.

To refactor the models by creating links to library blocks of similar clones, click the **Replace clones** button. Backup models are created in case you want to restore models to their original configurations.

To open the Test Manager, that tests the equivalence between the original and modified model,click the **Check Equivalency** button.

R2018b

Version: 4.2

New Features

Bug Fixes

Compatibility Considerations

Metrics Dashboard Customization: Configure compliance metrics, add metric thresholds, and customize Metrics Dashboard layout

Configure Compliance Metrics

In R2018b, you can use the Metrics Dashboard and metric APIs to obtain compliance and issues data on your Model Advisor configuration. To set up your Model Advisor configuration, see [Organize Checks and Folders Using the Model Advisor Configuration Editor](#). After you have set up your configuration, create an `slmetric.config.Configuration` object. Use the `setMetricFamilyParameterValues` method to specify the check group for which you want to obtain compliance and issues data.

This code sample is for one check group.

```
CONF = slmetric.config.Configuration.new('Name', 'config.xml');
CONF.setMetricFamilyParameterValues('ModelAdvisorStandard',...
    {'SysRoot_$optimization_checks'})
CONF.save('FileName', 'config.xml')
slmetric.config.setActiveConfiguration(fullfile(pwd, 'config.xml'));
```

`SysRoot_$optimization_checks` is the Check Group ID. In the Model Advisor Configuration Editor, this ID is on your custom folder. `ModelAdvisorStandard` is a standard string that you must specify as an input to the `setMetricFamilyParameterValues` method. To visualize results, use the new “Customize Metrics Dashboard Layout” on page 10-3 feature to add custom widgets to the Metrics Dashboard. You can also run the metric engine using APIs to obtain results.

Add Metric Thresholds

In R2018a, you could run the Metrics Dashboard to collect metric data on your model and use the various widgets to explore this data in more detail. Starting in R2018b, you can apply thresholds for categorizing metric data and display these thresholds on the Metrics Dashboard. Setting these thresholds enables you to assess the quality of your model. You use new APIs to specify threshold values corresponding to these three categories:

- Compliant — Metric data that is in an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

After you specify thresholds and update the active configuration, you can run the Metrics Dashboard. To specify metric thresholds, use these five new classes and two new functions:

Class	Description
<code>slmetric.config.Configuration</code>	Specify metric data categories and custom metric families
<code>slmetric.config.ThresholdConfiguration</code>	Object for holding a collection of metric data thresholds
<code>slmetric.config.Threshold</code>	Specify metric and <code>slmetric.metric.Result</code> property for thresholding
<code>slmetric.config.Classification</code>	Specify categorical metric data ranges
<code>slmetric.metric.ResultClassification</code>	Access metric data threshold values

Function	Description
<code>slmetric.config.getActiveConfiguration</code>	Obtain file path and name of XML file containing active Metrics Dashboard custom configuration
<code>slmetric.config.setActiveConfiguration</code>	Activate custom configuration for metric engine to use

The `slmetric.metric.Result` class contains two new properties: `Category` and `Classifications`. The `slmetric.metric.ResultCollection` class contains the new property `Category`.

Customize Metrics Dashboard Layout

In R2018b, you can customize the layout and functionality of the Metrics Dashboard. For custom metrics, you can visualize metric results by adding widgets to the Metrics Dashboard. You can choose among these visualization techniques:

- Radial gauge
- Single value
- Bar chart
- Distribution heatmap

You can also configure where existing, shipped widgets appear on the Metrics Dashboard or remove these widgets.

To add custom metrics and configure existing metrics to display on the Metrics Dashboard, use these six new classes and two new functions :

Class	Description
<code>slmetric.dashboard.Configuration</code>	Specify custom Metrics Dashboard configuration.
<code>slmetric.dashboard.Layout</code>	Add or remove widgets from the Metrics Dashboard.
<code>slmetric.dashboard.Container</code>	Add widgets next to each other on the Metrics Dashboard.
<code>slmetric.dashboard.Group</code>	Add a title to a group of widgets and place them next to each other on the Metrics Dashboard.
<code>slmetric.dashboard.Widget</code>	Object that holds a Library Reuse, System Interface, or System Info widget. Library Reuse and System Interface widgets are for existing, shipped metrics, so you cannot change the visualization property or metric ID. You cannot assign a metric ID to a System Info widget because it is not a metric.
<code>slmetric.dashboard.CustomWidget</code>	Create widgets in which you specify a visualization property and metric ID.

Class	Description
<code>slmetric.metric.MetaInformation</code>	The <code>slmetric.metric.MetaInformation</code> class properties contain metric metadata. On the Metrics Dashboard, when you click the widget for an individual metric, this metadata is in the table. For custom metrics, when you create a custom metric class, you specify the <code>slmetric.metric.MetaInformation</code> properties. To obtain metric metadata, use the new <code>getMetricMetaInformation</code> method. This method is for the <code>slmetric.Engine</code> object.

Function	Description
<code>slmetric.dashboard.getActiveConfiguration</code>	Return <code>slmetric.dashboard.Configuration</code> object corresponding to active Metrics Dashboard configuration.
<code>slmetric.dashboard.setActiveConfiguration</code>	Set active Metrics Dashboard configuration.

To create widgets for custom metrics, use the `slmetric.dashboard.CustomWidget` class. Add these widgets and shipped widgets in groups and containers or by themselves on the Metrics Dashboard.

For an example of all three of the preceding features, see [Customize Metrics Dashboard Layout and Functionality](#).

Simscape Support with Clone Detection: Detect and refactor clones in Simscape Models

In R2018b, you can run the **Identify Modeling Clones** tool on Simscape models.

JMAAB 4.01 Support: Automate checking of models to comply with JMAAB 4.01 modeling style guidelines

Use these new checks to verify compliance with Japan MATLAB Automotive Advisory Board Checks (JMAAB) guidelines. To execute these checks, open Model Advisor and select **By Task > Modeling Standards for JMAAB**.

Model Advisor Check	Check ID
Check block orientation	mathworks.jmaab.jc_0110
Check usable characters for signal names and bus names	mathworks.jmaab.jc_0222
Check usable characters for parameter names	mathworks.jmaab.jc_0232
Check length of model file name	mathworks.jmaab.jc_0241
Check length of folder name at every level of model path	mathworks.jmaab.jc_0242
Check length of subsystem names	mathworks.jmaab.jc_0243
Check length of Inport and Outport names	mathworks.jmaab.jc_0244
Check length of signal and bus names	mathworks.jmaab.jc_0245

Model Advisor Check	Check ID
Check length of parameter names	mathworks.jmaab.jc_0246
Check length of block names	mathworks.jmaab.jc_0247
Check if blocks are shaded in the model	mathworks.jmaab.jc_0604
Check operator order of Product blocks	mathworks.jmaab.jc_0610
Check icon shape of Logical Operator blocks	mathworks.jmaab.jc_0621
Check for parentheses in Fcn block expressions	mathworks.jmaab.jc_0622
Check usage of Memory and Unit Delay blocks	mathworks.jmaab.jc_0623
Check usage of Lookup Tables	mathworks.jmaab.jc_0626
Check usage of the Saturation blocks	mathworks.jmaab.jc_0628
Check Signed Integer Division Rounding mode	mathworks.jmaab.jc_0642
Check type setting by data objects	mathworks.jmaab.jc_0644
Check if tunable block parameters are defined as named constants	mathworks.jmaab.jc_0645
Check prohibited comparison operation of logical type signals	mathworks.jmaab.jc_0655
Check default/else case in Switch Case blocks and If blocks	mathworks.jmaab.jc_0656
Check usage of Merge block	mathworks.jmaab.jc_0659
Check for unused data in Stateflow Charts	mathworks.jmaab.jc_0700
Check first index of arrays in Stateflow	mathworks.jmaab.jc_0701
Check execution timing for default transition path	mathworks.jmaab.jc_0712
Check for parallel Stateflow state used for grouping	mathworks.jmaab.jc_0721
Check scope of data in parallel states	mathworks.jmaab.jc_0722
Check uniqueness of State names	mathworks.jmaab.jc_0730
Check usage of State names	mathworks.jmaab.jc_0731
Check uniqueness of Stateflow State and Data names	mathworks.jmaab.jc_0732
Check repetition of Action types	mathworks.jmaab.jc_0734
Check if each action in state label ends with a semicolon	mathworks.jmaab.jc_0735
Check indentation of Stateflow blocks	mathworks.jmaab.jc_0736
Check uniform spaces before and after operators	mathworks.jmaab.jc_0737
Check comments in state actions	mathworks.jmaab.jc_0738
Check updates to variables used in state transition conditions	mathworks.jmaab.jc_0741
Check boolean operations in condition labels	mathworks.jmaab.jc_0742
Check condition actions in Stateflow transitions	mathworks.jmaab.jc_0743

Model Advisor Check	Check ID
Check for unexpected backtracking in state transitions	mathworks.jmaab.jc_0751
Check usage of parentheses in Stateflow transitions	mathworks.jmaab.jc_0752
Check condition actions and transition actions in Stateflow	mathworks.jmaab.jc_0753
Check prohibited use of operation expressions in array indices	mathworks.jmaab.jc_0756
Check starting point of internal transition in Stateflow	mathworks.jmaab.jc_0760
Check prohibited combination of state action and flow chart	mathworks.jmaab.jc_0762
Check usage of internal transitions in Stateflow states	mathworks.jmaab.jc_0763
Check usage of transition conditions in Stateflow transitions	mathworks.jmaab.jc_0772

For more information, see Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance.

Additional MAAB 3.0 and High Integrity Checks: Improve quality and compliance to guidelines

The following table identifies the new and updated checks to verify compliance with MAAB 3.0 guidelines. For information about MAAB guidelines, see Model Advisor Checks for MAAB Guidelines.

Model Advisor Check	Description of Change
Check for blocks not recommended for C/C++ production code deployment	Removed from the Model Advisor By Task > Modeling Standards for MAAB checks. Check is not applicable for verifying compliance with MAAB modeling guidelines.
Check the number of function calls in MATLAB Function blocks	Use this new check to verify compliance with MathWorks® Automotive Advisory Board (MAAB) guidelines. To execute these check, open Model Advisor and select By Task > Modeling Standards for MAAB .

In R2018b, existing DO-178C/DO-331, and EN 50128, IEC 61508, IEC 62304, and ISO 26262 check IDs are renamed for better usability and consistency. For more information, see the information in this table.

Model Advisor Check	DO-178C/DO-331	IEC 61508, IEC 62304, ISO 26262, EN 50128	Check ID in R2018b
Check for MATLAB Function interfaces with inherited properties	mathworks.do178.himl_0002	mathworks.iec61508.himl_0002	mathworks.hism.himl_0002

Model Advisor Check	DO-178C/DO-331	IEC 61508, IEC 62304, ISO 26262, EN 50128	Check ID in R2018b
Check MATLAB Function metrics	mathworks.do178.himl_0003	mathworks.iec61508.himl_0003	mathworks.hism.himl_0003
Check MATLAB Code Analyzer messages	mathworks.do178.himl_0004	mathworks.iec61508.himl_0004	mathworks.hism.himl_0004
Check state machine type of Stateflow charts	mathworks.do178.hisf_0001	mathworks.iec61508.hisf_0001	mathworks.hism.hisf_0001
Check Stateflow charts for ordering of states and transitions	mathworks.do178.hisf_0002	mathworks.do178.hisf_0002	mathworks.hism.hisf_0002
Check for Strong Data Typing with Simulink I/O	mathworks.iec61508.StateflowProperUsage	mathworks.iec61508.StateflowProperUsage	mathworks.hism.hisf_0009
Check Stateflow debugging options	mathworks.do178.hisf_0011	mathworks.do178.hisf_0011	mathworks.hism.hisf_0011
Check Stateflow charts for transition paths that cross parallel state boundaries	mathworks.do178.hisf_0013	mathworks.iec61508.hisf_0013	mathworks.hism.hisf_0013
Check Stateflow charts for strong data typing	mathworks.do178.hisf_0015	mathworks.iec61508.hisf_0015	mathworks.hism.hisf_0015
Check usage of shift operations for Stateflow data	mathworks.do178.hisf_0064	mathworks.iec61508.hisf_0064	mathworks.hism.hisf_0064
Check assignment operations in Stateflow charts	mathworks.do178.hisf_0065	mathworks.iec61508.hisf_0065	mathworks.hism.hisf_0065
Check Stateflow charts for unary operators	mathworks.do178.hisf_0211	mathworks.iec61508.hisf_0211	mathworks.hism.hisf_0211
Check safety-related diagnostic settings for data store memory	mathworks.do178.DataStoreMemoryDiagnosticSet	mathworks.do178.DataStoreMemoryDiagnosticSet	mathworks.hism.hisl_0013
Check for inconsistent vector indexing methods	mathworks.do178.hisl_0021	mathworks.iec61508.hisl_0021	mathworks.hism.hisl_0021
Check for variant blocks with 'Generate preprocessor conditionals' active	mathworks.do178.VariantBlock	mathworks.do178.VariantBlock	mathworks.hism.hisl_0023
Check for root Inports with missing properties	mathworks.iec61508.RootLevelInports	mathworks.iec61508.RootLevelInports	mathworks.hism.hisl_0024
Check for root Inports with missing range definitions	mathworks.iec61508.InportRange	mathworks.iec61508.InportRange	mathworks.hism.hisl_0025

Model Advisor Check	DO-178C/DO-331	IEC 61508, IEC 62304, ISO 26262, EN 50128	Check ID in R2018b
Check for root Outports with missing range definitions	mathworks.iec61508.OutportRange	mathworks.iec61508.OutportRange	mathworks.hism.hisl_0026
Check model object names	mathworks.do178.hisl_0032	mathworks.iec61508.hisl_0032	mathworks.hism.hisl_0032
Check usage of lookup table blocks	mathworks.do178.LUTRangeCheckCode	mathworks.do178.LUTRangeCheckCode	mathworks.hism.hisl_0033
Check usage of Signal Routing blocks	mathworks.do178.SignalRoutingBlockUsage	mathworks.iec61508.SignalRoutingBlockUsage	mathworks.hism.hisl_0034
Check safety-related diagnostic settings for saving	mathworks.do178.SavingDiagnosticsSet	mathworks.do178.SavingDiagnosticsSet	mathworks.hism.hisl_0036
Check safety-related model referencing settings	mathworks.do178.MdlrefOptSet	mathworks.do178.MdlrefOptSet	mathworks.hism.hisl_0037
Check safety-related solver settings for simulation time	mathworks.iec61508.SimulationTimeOptions	mathworks.iec61508.SimulationTimeOptions	mathworks.hism.hisl_0040
Check safety-related solver settings for solver options	mathworks.iec61508.hisl_0041	mathworks.iec61508.hisl_0041	mathworks.hism.hisl_0041
Check safety-related solver settings for tasking and sample-time	mathworks.do178.hisl_0042	mathworks.iec61508.hisl_0042	mathworks.hism.hisl_0042
Check safety-related diagnostic settings for solvers	mathworks.do178.SolverDiagnosticsSet	mathworks.do178.SolverDiagnosticsSet	mathworks.hism.hisl_0043
Check safety-related diagnostic settings for sample time	mathworks.do178.SampleTimeDiagnosticsSet	mathworks.do178.SampleTimeDiagnosticsSet	mathworks.hism.hisl_0044
Check safety-related optimization settings for loop unrolling threshold	mathworks.iec61508.hisl_0051	mathworks.iec61508.hisl_0051	mathworks.hism.hisl_0051
Check Stateflow charts for uniquely defined data objects	mathworks.do178.hisl_0061	mathworks.do178.hisl_0061	mathworks.hism.hisl_0061
Check for model elements that do not link to requirements	mathworks.do178.RequirementInfo	mathworks.iec61508.RequirementInfo	mathworks.hism.hisl_0070
Check safety-related diagnostic settings for compatibility	mathworks.do178.CompatibilityDiagnosticsSet	mathworks.do178.CompatibilityDiagnosticsSet	mathworks.hism.hisl_0301

Model Advisor Check	DO-178C/DO-331	IEC 61508, IEC 62304, ISO 26262, EN 50128	Check ID in R2018b
Check safety-related diagnostic settings for parameters	mathworks.do178.DataValidityParamDiagnosticSet	mathworks.do178.DataValidityParamDiagnosticSet	mathworks.hism.hisl_0302
Check safety-related diagnostic settings for Merge blocks	mathworks.do178.hisl_0303	mathworks.iec61508.hisl_0303	mathworks.hism.hisl_0303
Check safety-related diagnostic settings for model initialization	mathworicsSet	mathworks.do178.InitDiagnosticsSet	mathworks.hism.hisl_0304
Check safety-related diagnostic settings for data used for debugging	mathworks.do178.DataValidityDebugDiagnosticSet	mathworks.do178.DataValidityDebugDiagnosticSet	mathworks.hism.hisl_0305
Check safety-related diagnostic settings for signal connectivity	mathworks.do178.ConnectivitySignalsDiagnosticsSet	mathworks.do178.ConnectivitySignalsDiagnosticsSet	mathworks.hism.hisl_0306
Check safety-related diagnostic settings for bus connectivity	mathworks.do178.ConnectivityBussesDiagnosticsSet	mathworks.do178.ConnectivityBussesDiagnosticsSet	mathworks.hism.hisl_0307
Check safety-related diagnostic settings that apply to function-call connectivity	mathworks.do178.FcnCallDiagnosticsSet	mathworks.do178.FcnCallDiagnosticsSet	mathworks.hism.hisl_0308
Check safety-related diagnostic settings for type conversions	mathworks.do178.TypeConversionDiagnosticsSet	mathworks.iec61508.hisl_0309	mathworks.hism.hisl_0309
Check safety-related diagnostic settings for model referencing	mathworks.do178.MdlrefDiagnosticsSet	mathworks.do178.MdlrefDiagnosticsSet	mathworks.hism.hisl_0310
Check safety-related diagnostic settings for Stateflow	mathworks.do178.hisl_0311	mathworks.iec61508.hisl_0311	mathworks.hism.hisl_0311

In R2018b, the following High-Integrity Systems Modeling checks are split into multiple checks based on functionality.

Model Advisor Check	Split Check Titles	Check ID
Check usage of Math Operations blocks	Check usage of Abs blocks	mathworks.hism.hisl_0001
	Check usage of Math Function blocks (rem and reciprocal functions)	mathworks.hism.hisl_0002
	Check usage of Math Function blocks (log and log10 functions)	mathworks.hism.hisl_0004
	Check usage of Assignment blocks	mathworks.hism.hisl_0029

Model Advisor Check	Split Check Titles	Check ID
Check usage of Logic and Bit Operations blocks	Check for Relational Operator blocks that equate floating-point types	mathworks.hism.hisl_0016
	Check usage of Relational Operator blocks	mathworks.hism.hisl_0017
	Check usage of Logical Operator blocks	mathworks.hism.hisl_0018
Check usage of Ports and Subsystems blocks	Check usage of While Iterator blocks	mathworks.hism.hisl_0006
	Check sample time-dependent blocks	mathworks.hism.hisl_0007
	Check usage of For Iterator blocks	mathworks.hism.hisl_0008
	Check usage of If blocks and If Action Subsystem blocks	mathworks.hism.hisl_0010
	Check usage Switch Case blocks and Switch Case Action Subsystem blocks	mathworks.hism.hisl_0011
Check safety-related code generation settings	Check safety-related code generation settings for comments	mathworks.hism.hisl_0038
	Check safety-related code generation interface settings	mathworks.hism.hisl_0039
	Check safety-related code generation settings for code style	mathworks.hism.hisl_0047
	Check safety-related code generation symbols settings	mathworks.hism.hisl_0049
Check usage of Stateflow constructs	Check for Strong Data Typing with Simulink I/O	mathworks.hism.hisf_0009
	Check Stateflow charts for ordering of states and transitions	mathworks.hism.hisf_0002
	Check Stateflow debugging options	mathworks.hism.hisf_0011
	Check Stateflow charts for uniquely defined data objects	mathworks.hism.hisl_0061
Check safety-related optimization settings	Check safety-related optimization settings for logic signals	mathworks.hism.hisl_0045
	Check safety-related block reduction optimization settings	mathworks.hism.hisl_0046

Model Advisor Check	Split Check Titles	Check ID
	Check safety-related optimization settings for application lifespan	mathworks.hism.hisl_0048
	Check safety-related optimization settings for data initialization	mathworks.hism.hisl_0052
	Check safety-related optimization settings for data type conversions	mathworks.hism.hisl_0053
	Check safety-related optimization settings for division arithmetic exceptions	mathworks.hism.hisl_0054

The following table includes new checks for High-Integrity Systems Modeling in R2018b.

Model Advisor Check	Check ID
Check usage of standardized MATLAB function headers	mathworks.hism.himl_0001
Check if/elseif/else patterns in MATLAB Function blocks	mathworks.hism.himl_0006
Check switch statements in MATLAB Function blocks	mathworks.hism.himl_0007
Check usage of relational operators in MATLAB Function blocks	mathworks.hism.himl_0008
Check usage of equality operators in MATLAB Function blocks	mathworks.hism.himl_0009
Check usage of logical operators and functions in MATLAB Function blocks	mathworks.hism.himl_0010
Check for inappropriate use of transition paths	mathworks.hism.hisf_0014
Check naming of ports in Stateflow charts	mathworks.hism.hisf_0016
Check scoping of Stateflow data objects	mathworks.hism.hisf_0017
Check usage of conditionally executed subsystems	mathworks.hism.hisl_0012
Check usage of Merge blocks	mathworks.hism.hisl_0015
Check usage of Bitwise Operator block	mathworks.hism.hisl_0019
Check data types for blocks with index signals	mathworks.hism.hisl_0022
Check model file name	mathworks.hism.hisl_0031
Check global variables in graphical functions	mathworks.hism.hisl_0062
Check for length of user-defined object names	mathworks.hism.hisl_0063
Check usage of Gain blocks	mathworks.hism.hisl_0066
Check data type of loop control variables	mathworks.hism.hisl_0102

For more information, see:

- Model Checks for DO-178C/DO-331 Standard Compliance
- Model Checks for IEC 61508, IEC 62304, ISO 26262, and EN 50128 Standard Compliance

Compatibility Considerations

The old check IDs continue to work with the Model Advisor API. In a future release, the old check IDs will be removed. It is recommended that you update your scripts to use the new check IDs.

If you set a default configuration for when the Model Advisor opens, the default configuration continues to run the old check IDs.

High Integrity Systems Modeling Checks: Use the additional conditions to check the configuration parameters

From R2018b, you can use these conditions to check the configuration parameters:

Check Name	Parameter	Parameter Values
Check safety-related diagnostic settings for Stateflow	Undirected event broadcasts	'none' 'warning' (default) 'error' Check passes when value is set to 'error'.
	Transition action specified before condition action	'none' 'warning' (default) 'error' Check passes when value is set to 'error'.
Check safety-related diagnostic settings for sample time	Single task rate transition	'none' (default) 'warning' 'error' Check passes when value is set to 'error'.
	Tasks with equal priority	'none' 'warning' (default) 'error' Check passes when value is set to 'error'.
	Unspecified inheritability of sample time	'none' 'warning' (default) 'error' Check passes when value is set to 'error'.

MISRA C:2012 and Secure Coding Standards: Improve compliance of generated code by using updated Model Advisor checks

Modifications to existing Model Advisor checks that you use to verify compliance with MISRA C:2012 and Secure Coding standards are outlined in this table.

Model Advisor Check	Description of Change
Check configuration parameters for MISRA C:2012 Check configuration parameters for secure coding standards	Checks now analyze the setting for these configuration parameters: <ul style="list-style-type: none"> • External mode • Undirected event broadcasts • Compile-time recursion limit for MATLAB functions • Enable run-time recursion for MATLAB functions
Check for blocks not recommended for MISRA C:2012 Check for blocks not recommended for secure coding standards	Checks now flag the usage of these blocks in a model or subsystem: <ul style="list-style-type: none"> • Compose String • Scan String • String to Double • String to Single • To String

Mnemonic Support: Use keyboard shortcuts with Metrics Dashboard

In R2018b, you can use your keyboard to choose from the actions on the Metric Dashboard toolbar. To enable keyboard shortcuts, on Windows® and Linux® machines, type **Alt + M**. Mac computers do not support keyboard shortcuts.

Check Style for Model Advisor: Create checks that generate interactive reports


In R2018b, the Model Advisor has two new report styles that you can use to view the check results. In addition to Recommended Action, you can now view the results for some of the Model Advisor checks by:

- **Subsystem**. This view organizes the flagged model components by subsystem. The report provides a recommended action for each flagged issue. You can click the hyperlink path to open the affected model component in the model editor.

Check whether block names appear below blocks (recommended check style)

Analysis

Example new style callback (recommended check style)

Result:  Warning View by

Identify blocks where the name is not displayed below the block.

Warning

The following blocks have names that do not display below the blocks:

Subsystem	Block Path
example_sl-demo_fuelsys	.../Throttle Angle Fault Switch
example_sl-demo_fuelsys/fuel_rate_control	.../fuel_rate_control/validate_sample_time

Recommended Action

Change the location such that the block name is below the block.

Action

Click the button to place block names below blocks

- **Block.** This view organizes the flagged model components by block. The report provides a recommended action for each flagged issue. You can click the hyperlink path to open the affected model component in the model editor.

Check whether block names appear below blocks (recommended check style)

Analysis

Example new style callback (recommended check style)

Run This Check

Result:  Warning

View by

Issues for block `example_sl-demo_fuelsys/Throttle Angle Fault Switch`:

Identify blocks where the name is not displayed below the block.

Warning

The following blocks have names that do not display below the blocks:

- [example_sl-demo_fuelsys/Throttle Angle Fault Switch](#)

Recommended Action

Change the location such that the block name is below the block.

Issues for block `example_sl-demo_fuelsys/fuel_rate_control/validate_sample_time`:

Identify blocks where the name is not displayed below the block.

Warning

The following blocks have names that do not display below the blocks:

- [example_sl-demo_fuelsys/fuel_rate_control/validate_sample_time](#)

Recommended Action

Change the location such that the block name is below the block.

Action

Click the button to place block names below blocks

Make block names appear below blocks

In R2018b, new report styles are available for the following Model Advisor checks:

- Identify blocks generating inefficient algorithms
- Check state machine type of Stateflow charts
- Check usage of Gain blocks
- Check for indexing in blocks
- Check for prohibited blocks in discrete controllers
- Check for prohibited sink blocks
- Check positioning and configuration of ports
- Check for matching port and signal names
- Check whether block names appear below blocks
- Check for mixing basic blocks and subsystems

- Check for unconnected ports and signal lines
- Check position of Trigger and Enable blocks
- Check usage of tunable parameters in blocks
- Check the display attributes of block names
- Check display for port blocks
- Check orientation of Subsystem blocks
- Check usage of Relational Operator blocks
- Check use of Simulink in Stateflow charts
- Check use of default variants
- Check usage of Discrete-Time Integrator block
- Check usage of State names
- Check uniform spaces before and after operators
- Check comments in state actions
- Check prohibited comparison operation of logical type signals
- Check usage of internal transitions in Stateflow states
- Check usage of transition conditions in Stateflow transitions
- Check block orientation
- Check usage of parentheses in Stateflow transitions
- Check boolean operations in condition labels
- Check usage of transition conditions in Stateflow transitions
- Check prohibited use of operation expressions in array indices
- Check if each action in state label ends with a semicolon
- Check prohibited combination of state action and flow chart
- Check updates to variables used in state transition conditions
- Check condition actions in Stateflow transitions
- Check starting point of internal transition in Stateflow
- Check usage of Lookup Tables
- Check for parentheses in Fcn block expressions
- Check for blocks not supported for row-major code generation
- Identify TLC S-Functions with unset array layout
- Check input and output datatype for Switch blocks
- Check type setting by data objects
- Check for the Saturation and Saturation Dynamic blocks that perform type casting
- Check usage of fixed-point data type with non-zero bias

To apply the new report formats to your custom Model Advisor checks, use the classes and function listed in this table.

Class	Description
ModelAdvisor.Check	<p>New method setResultDetails associates ResultDetailObjs and ElementResults with the check (CheckObj) in the check callback function.</p> <p>New property ModelAdvisor.Check.ResultDetails stores the ResultDetailObjs.</p> <p>In the setCallbackFcn method, input argument 'DetailStyle' specifies the callback function for the new report style. Define the new 'DetailStyle' callback function type in the ModelAdvisor.Check.CallbackStyle property.</p>
ModelAdvisor.ResultDetail	<p>Defines the result detail objects for a specific check object. Each object stores the result details for one model element, such as a block that violates a check.</p>

Functionality Being Removed or Changed

Instances of slmetric.metric.Result for the mathworks.metrics.CloneContent and mathworks.metrics.LibraryContent metrics contain these differences between R2018a and R2018b:

- Previously, for the mathworks.metrics.CloneContent metric, the Value property provided the number of components in a clone. The Measures property was not applicable. In R2018b, the Value property provides the fraction of the total number of subcomponents that are clones. The Measures property is a vector containing the number of clones, total number of components, and the clone group number.
- Previously, for the mathworks.metrics.LibraryContent metric, the Value property provided the number of components involved in a library, excluding clones. The Measures property was not applicable. In R2018b, the Value property provides the fraction of the total number of components that are linked-library blocks. The Measures property is a vector containing the number of linked-library blocks and total number of components.

The Metrics Dashboard incorporates these changes. Previously, the **Library Reuse** widget displayed percentages that were a combination of five metrics. The widget directly used the Clone detection (mathworks.metrics.CloneDetection) and Library link (mathworks.metrics.LibraryCount) metrics. To calculate percentages, the widget indirectly used the MATLAB Function count (mathworks.metrics.MatlabFunctionCount), the Chart count (mathworks.metrics.StateflowChartCount), and the Subsystem count (mathworks.metrics.SubSystemCount) metrics.

In R2018b, The Library Reuse widget is **Potential Reuse** and **Actual Reuse** bars. The **Potential Reuse** bar displays the mathworks.metrics.CloneContent Value as a percentage. The **Actual Reuse** bar displays the mathworks.metrics.LibraryContent Value as a percentage.

For the mathworks.metrics.CloneContent and mathworks.metrics.LibraryCount metrics, changing the slmetric.metric.Result Measures and Values properties supports displaying

metric threshold values on the Metrics Dashboard because you specify metric thresholds on a single metric and not on a combination of metrics. Metric thresholds is a new R2018b feature. For more information, see [Customize Metrics Dashboard Layout and Functionality](#).

R2018a

Version: 4.1

New Features

Bug Fixes

Additional Checks for MAAB 3.0 and JMAAB 4.0 Guidelines: Automate checking for MAAB 3.0 guidelines for Simulink, Stateflow, Variant Subsystems, and MATLAB Function Blocks and JMAAB 4.0 guidelines

MAAB Modeling Checks

Use these new checks to verify compliance with MathWorks Automotive Advisory Board (MAAB) guidelines. To execute these checks, open Model Advisor (Simulink) and select **By Task > Modeling Standards for MAAB**.

For information about MAAB® guidelines, see Model Advisor Checks for MAAB Guidelines (Simulink).

By Task > Modeling Standards for MAAB subfolder	Model Advisor Check	Addresses Guideline
Naming Convention	Check Simulink bus signal names	na_0030: Usable characters for Simulink Bus names
Model Architecture	Check unused ports in Variant Subsystems	na_0020: Number of inputs to variant subsystems
Model Architecture	Check use of default variants	na_0036: Default variant
Model Architecture	Check use of single variable variant conditionals	na_0037: Use of single variable variant conditionals
Model Architecture	Check number of Stateflow states per container	na_0040: Number of states per container
Simulink	Check fundamental logical and numerical operations	na_0002: Appropriate implementation of fundamental logical and numerical operations
Simulink	Check usage of merge blocks	na_0032: Use of merge blocks
Simulink	Check logical expressions in 'If' blocks	na_0003: Simple logical expressions in If Condition block
Stateflow	Check nested states in Stateflow charts	na_0038: Levels in Stateflow charts
Stateflow	Check use of Simulink in Stateflow charts	na_0039: Use of Simulink in Stateflow charts
MATLAB Functions	Check usage of reserved keywords in Simulink	na_0019: Restricted Variable Names
MATLAB Functions	Check usage of character vector inside MATLAB Function block	na_0021: Strings
MATLAB Functions	Check Recommended patterns for Switch/Case Statements	na_0022: Recommended patterns for Switch/Case statements

Modifications to existing MAAB checks are outlined in this table.

Model Advisor Check	Description of Change
<p>Check entry formatting in State blocks in Stateflow charts</p> <p>Check for mismatches between names of Stateflow ports and associated signals</p> <p>Check for Strong Data Typing with Simulink I/O</p> <p>Check for indexing in blocks</p> <p>Check for MATLAB expressions in Stateflow charts</p> <p>Check transition orientations in flow charts</p> <p>Check usage of exclusive and default states in state machines</p> <p>Check transition actions in Stateflow charts</p> <p>Check for unary minus operations on unsigned integers in Stateflow charts</p> <p>Check for equality operations between floating-point expressions in Stateflow charts</p> <p>Check return value assignments of graphical functions in Stateflow charts</p> <p>Check usage of return values from a graphical function in Stateflow charts</p> <p>Check default transition placement in Stateflow charts</p> <p>Check for pointers in Stateflow charts</p> <p>Check for event broadcasts in Stateflow charts</p> <p>Check for bitwise operations in Stateflow charts</p> <p>Check for comparison operations in Stateflow charts</p>	<p>These checks now require a Stateflow license. These checks are included in the Model Advisor interface only when a Stateflow license is detected.</p>
<p>Check the display attributes of block names</p>	<p>You can now use the input parameters in the Model Advisor Configuration Editor to customize the blocks and masks to be analyzed the check.</p>

Model Advisor Check	Description of Change
Check position of Trigger and Enable blocks	Check now verifies that For Each, For Iterator, and While Iterator blocks are in a uniform location on the subsystem diagram.

JMAAB Modeling Checks

Checks that verify compliance with Japan MATLAB Automotive Advisory Board (JMAAB) guidelines are now available in the Model Advisor under the new menu item **By Task > Modeling Standards for JMAAB**.

The following table identifies new checks to verify compliance with JMAAB 4.0 guidelines. For information about JMAAB guidelines, see Model Advisor Checks for MAAB Guidelines (Simulink).

By Task > Modeling Standards for JMAAB Subfolder	Model Advisor Check	Addresses Guideline
Simulink	Check usage of Discrete-Time Integrator block	jc_0627: Guideline for using the Discrete-Time Integrator block
Simulink	Check for blocks with a fixed-point data type whose bias is not zero	jc_0643: Fixed-point setting
Simulink	Check input and output datatype for Switch blocks	jc_0650: Block input/output data type with switching function
Simulink	Check input signal data types in product blocks that perform division	jc_0611: Input signal sign during product block division

The following table identifies MAAB checks are also applicable to JMAAB 4.0 guidelines. These checks are available in the Model Advisor (Simulink) under **By Task > Modeling Standards for JMAAB**. There are no changes to the check IDs.

By Task > Modeling Standards for JMAAB Subfolder	Model Advisor Check
Naming Conventions	Check file names
	Check folder names
	Check subsystem names
	Check port block names
	Check character usage in signal labels
	Check character usage in block names
Model Architecture	Check for mixing basic blocks and subsystems
Model Configuration Options	Check Implement logic signals as Boolean data (vs. double)
Simulink	Check for Simulink diagrams using nonstandard display attributes
	Check font formatting

By Task > Modeling Standards for JMAAB Subfolder	Model Advisor Check
	Check positioning and configuration of ports Check whether block names appear below blocks Check the display attributes of block names Check position of Trigger and Enable blocks Check for nondefault block attributes Check Trigger and Enable block names Check signal line labels Check for propagated signal labels Check for unconnected ports and signal lines Check for prohibited blocks in discrete controllers Check for prohibited sink blocks Check usage of Switch blocks Check usage of Relational Operator blocks Check for indexing in blocks Check usage of tunable parameters in blocks Check orientation of Subsystem blocks
Stateflow	Check transition orientations in flow charts Check return value assignments of graphical functions in Stateflow charts Check default transition placement in Stateflow charts Check for Strong Data Typing with Simulink I/O Check Stateflow data objects with local scope Check usage of return values from a graphical function in Stateflow charts Check for MATLAB expressions in Stateflow charts Check for pointers in Stateflow charts Check for event broadcasts in Stateflow charts Check transition actions in Stateflow charts Check for bitwise operations in Stateflow charts Check for unary minus operations on unsigned integers in Stateflow charts Check for comparison operations in Stateflow charts Check for mismatches between names of Stateflow ports and associated signals
MATLAB Function	Check input and output settings of MATLAB Functions Check MATLAB Function metrics Check MATLAB code for global variables

Block Constraint Authoring with Edit-Time: Define checks for supported or unsupported blocks and parameters while editing

In R2018a, there are seven new classes and two new functions that you can use to create block and parameter constraints. You can use the `sl_customization` function template to create basic Model Advisor checks. These checks include these constraints and a check algorithm callback. You can check your model as you edit or run the checks interactively after you complete your model design.

The new classes and their descriptions are in the table.

Class	Description
<code>Advisor.authoring.PositiveBlockParameterConstraint</code>	Check for supported block parameter values.
<code>Advisor.authoring.NegativeBlockParameterConstraint</code>	Check for unsupported block parameter values.
<code>Advisor.authoring.PositiveModelParameterConstraint</code>	Check for supported model parameter values.
<code>Advisor.authoring.NegativeModelParameterConstraint</code>	Check for unsupported model parameter values.
<code>Advisor.authoring.PositiveBlockTypeConstraint</code>	Check for supported blocks.
<code>Advisor.authoring.NegativeBlockTypeConstraint</code>	Check for unsupported blocks.
<code>Advisor.authoring.CompositeConstraint</code>	Check whether blocks or parameters meet a combination of constraints.

The new functions and their descriptions are in the table.

Function	Description
<code>Advisor.authoring.generateBlockConstraintsDataFile</code>	Generate XML data file for custom check for block constraints.
<code>Advisor.authoring.createBlockConstraintCheck</code>	Create Model Advisor check for registering block constraints.

You can also create constraints that check for pre-requisite constraints before checking the actual constraint. For more information, see [Define Checks for Supported or Unsupported Blocks and Parameters](#).

Clone Refactoring Workflow: Apply multiple refactoring steps to the same model

In R2017b, for each refactoring step, the Identify Modeling Clones tool created a new model. In R2018a, you can apply multiple refactoring steps to the same model. For each step, the tool creates a back-up model. The back-up models are in the folder that has the prefix `m2m_` plus the model name. For a single model, this enhanced functionality makes it easier for you to replace clones with links to library blocks. For more information, see [Enable Component Reuse by Using Clone Detection](#).

Automatic Refactoring for Similar Clones: Add masks to similar clones and refactor model

In R2017b, you could use the Identify Modeling Clones tool to identify exact clones and replace them with links to library blocks. The tool also identified similar clones (that is, clones that had identical block types and connections but different parameter settings or values). The tool identified similar and exact clones as part of different steps.

In R2018a, for similar clones, the Identify Modeling Clones tool creates a masked library subsystem. The refactored model contains links from the clone instances to this masked library subsystem. The tool identifies similar and exact clones as part of the same steps. You specify which clones you want to detect by setting the value of the **Maximum number of different parameters** parameter. A value of 0 indicates that you want the tool to identify only exact clones.

Replacing clones with links to library blocks enables component reuse. If you have Simulink Coder or Embedded Coder software, you can generate reusable code for library subsystems. For more information, see [Enable Component Reuse by Using Clone Detection](#).

Clone Detection Exclusion Editor: Exclude subsystems and referenced models from clone detection

In R2018a, there is a new Clone Detection Exclusion Editor that you can use to exclude a subsystem or referenced model from the Identify Modeling Clones tool. For subsystems, right-click the subsystem and select **Identify Modeling Clones > Subsystem and Its Contents > Add to exclusions**. For referenced models, right-click the Model block and select **Identify Modeling Clones > Model Reference > Add to exclusions**. You can use the Exclusion Editor to specify a rationale for excluding subsystems and referenced models and whether to store exclusions in a model file. For more information, see [Enable Component Reuse by Using Clone Detection](#).

Automatic Data Store Memory Block Elimination: Identify and refactor Data Store Memory Block blocks with Model Transformer

In R2018a, use the Model Transformer tool to refactor a model to eliminate Data Store Memory, Data Store Read, and Data Store Write blocks. Eliminating these blocks improves model readability by making data-dependency explicit. If you have Simulink Coder, eliminating these blocks may improve the efficiency of the generated code by reducing the number of global variables, the corresponding reads and writes to these global variables, and stack size. For more information, see [Improve Model Readability by Eliminating Local Data Store Blocks](#).

Grid Visualization for Metrics: View results of Model Advisor checks in a grid to identify patterns in results

In R2017b, after collecting metric data by using the Metric Dashboard, you could view results for High Integrity and MAAB Compliance metrics in tabular format. You viewed this data by clicking the widgets in the **MODELING GUIDELINE COMPLIANCE** section.

In R2018a, when you click the **High Integrity Compliance** and **MAAB Compliance** widgets, you can view results in a table or a grid. In the toolbar, you change views by clicking **Table** or **Grid**.

Viewing results in a grid enables you to identify compliance check issues and failure patterns quickly. The grid contains a row for each model component and a column for each check. The colors in each grid cell indicate this status.

Color	Check Status
Red	Fail
Orange	Warning
Green	Pass
Gray	Not run

The colors in the row and column headers indicate the worst status for that component or check. For example, for a row, if the worst status is a failure, the row header is red.

Placing your cursor over a cell displays the component, check status, and check name. You can click individual cells to navigate to the corresponding block and identify compliance issues for that block. To navigate to the corresponding check in the Model Advisor, click a column header. To navigate to the corresponding model component, click a row header. For more information, see [Collect and Explore Metric Data by Using the Metrics Dashboard](#).

MathWorks High-Integrity Guidelines and Checks: Verify compliance with safety standards by using high-integrity checks and guidelines

High-Integrity System Modeling Checks

This table identifies modifications to existing high-integrity system modeling checks.

Model Advisor Check	Description of Change
<p>DO-178C/DO-331 checks:</p> <ul style="list-style-type: none"> • Check usage of shift operations for Stateflow data • Check assignment operations in Stateflow Charts • Check state machine type of Stateflow charts • Check Stateflow charts for ordering of states and transitions • Check Stateflow debugging options • Check for inconsistent vector indexing methods • Check Stateflow charts for strong data typing • Check Stateflow charts for unary operators • Check Stateflow charts for uniquely defined data objects <p>IEC 61508, IEC 62304, EN 50128, and ISO 26262 checks:</p> <ul style="list-style-type: none"> • Check usage of shift operations for Stateflow data • Check assignment operations in Stateflow Charts • Check state machine type of Stateflow charts • Check Stateflow charts for ordering of states and transitions • Check Stateflow debugging options • Check for inconsistent vector indexing methods • Check Stateflow charts for strong data typing • Check Stateflow charts for unary operators • Check Stateflow charts for uniquely defined data objects • Display model metrics and complexity report 	<p>Checks require a Stateflow license. These checks are available in the Model Advisor only when a Stateflow license is detected.</p>

Model Advisor Check	Description of Change
DO-178C/DO-331: Check for model elements that do not link to requirements IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check for model elements that do not link to requirements	Checks require a Simulink Requirements license. These checks are available in the Model Advisor only when a Simulink Requirements license is detected.
Check for blocks not recommended for MISRA C:2012 Check for blocks not recommended for MISRA C:2012	Checks analysis now includes the content in library linked blocks and masked subsystems.
Check for blocks not recommended for C/C++ production code deployment Check for blocks not recommended for C/C++ production code deployment	Checks analysis now includes the content in library linked blocks.

High-Integrity Modeling Guidelines

These high-integrity system modeling guidelines are introduced in R2018a:

- hisl_0056: Configuration Parameters > Optimization > Optimize using the specified minimum and maximum values
- hisl_0066: Usage of Gain blocks
- hisl_0314: Configuration Parameters > Diagnostics > Data Validity > Signals
- hisf_0016: Stateflow port names
- hisf_0017: Stateflow data object scoping

This table identifies removed and modified high-integrity system modeling guidelines. For a complete list of high-integrity system modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines (Simulink).

High-Integrity Modeling Guideline	Rationale
hisl_0202: Use of data conversion blocks to improve MISRA C:2012 compliance	Removed - guideline no longer applies. The code generation process does not produce this MISRA violation in the generated code.
hisl_0401: Encapsulation of code to improve MISRA C:2012 compliance	Removed - not a modeling guideline. The guideline addresses issues in source code external to a model. Embedded Coder does not directly call assembly language code.
hisl_0402: Use of custom #pragma to improve MISRA C:2012 compliance	Removed - not a modeling guideline. The guideline addresses issues in source code external to a model.
hisl_0403: Use of char data type to improve MISRA C:2012 compliance	Removed - not a modeling guideline. The guideline addresses issues in source code external to a model. Embedded Coder does not directly create data of type char.

High-Integrity Modeling Guideline	Rationale
hisl_0055: Prioritization of code generation objectives for high-integrity systems	Removed - guideline is redundant. Model configuration parameter considerations are covered by existing high-integrity systems guidelines.
hisl_0042: Configuration Parameters > Solver > Tasking and sample time options	<p>Removed criteria A, "Periodic sample time constraint to specified and assign values to Sample time properties". Criteria A no longer applies because there is no known safety issue with the periodic sample time constraint values.</p> <p>Change affects checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related solver settings for tasking and sample-time • EC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related solver settings for tasking and sample-time
hisl_0310: Configuration Parameters > Diagnostics > Model Referencing	Set configuration parameter Model block version mismatch to none. This change reflects behavior of the corresponding Model Advisor check.
hisl_0016: Usage of blocks that compute relational operators	Add the If block to the list of blocks that compute relational operations. This change reflects behavior of Model Advisor check.
hisl_0017: Usage of blocks that compute relational operators (2)	Add Rational B, "For Relational Operator blocks, ensure that all input signals are of the same data type". This change reflects behavior of the corresponding Model Advisor check.
<p>hisl_0046: Configuration Parameters > Code Generation > Optimization > Block reduction</p> <p>hisl_0051: Configuration Parameters > Code Generation > Optimization > Loop unrolling threshold</p> <p>hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization</p> <p>hisl_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values</p> <p>hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions</p>	<p>In R2018a, the Optimization pane in the Configuration Parameters dialog box moved to Code Generation > Optimization. The title of and support documentation for the affected guidelines were updated to reflect this change. Where applicable, the Model Advisor checks were updated to reflect this change.</p>

High-Integrity Modeling Guideline	Rationale
hisl_0045: Configuration Parameters > Code Generation > Optimization > Implement logic signals as Boolean data (vs. double)	In R2018a, configuration parameter Implement logic signals as Boolean data (vs. double) moved to the new Math and Data Types pane in the Configuration Parameters dialog box. The title of and support documentation for the guideline were updated to reflect this change.
hisl_0019: Usage of Bitwise Operator block	Remove Criteria B, "Choose an output data type that represents zero exactly" from the guideline. Criteria B does not apply because the Bitwise Operator block, by design, accepts only signed or unsigned integer to produce the output data type that represents zero exactly. It does not accept other data types.
hisl_0102: Data type of loop control variables to improve MISRA C:2012 compliance	Remove While loops and While Iterator blocks. These are not recommended for safety critical systems because an infinite loop can occur. Guideline does not have corresponding checks.
hisl_0031: File and folder names	Remove folder naming recommendations because there is no known safety issue with folder names. Guideline title was updated accordingly.

MISRA C: 2012 Modeling Checks: Improve compliance of generated code by using MISRA C:2012 standards checks

Use this new check to verify compliance of your generated code with MISRA C:2012 standards. To execute this check, open Model Advisor (Simulink) and select **By Task > Modeling Standards for MISRA C:2012**

Model Advisor Check	Description	Addresses Standards
Check bus object names that are used as element names	Check now identifies Simulink.Bus object names that are used as Simulink.Bus element names.	<ul style="list-style-type: none"> MISRA C:2012 Rule 5.6 MISRA AC AGC Rule 5.3

Modifications to existing compliance checks are outlined in this table.

Model Advisor Check	Description of Change
Check for bitwise operations on signed integers	The check assumes that code is generated for the whole model. When code is generated by a subsystem build or export functions, the check can produce incorrect results.
Check for blocks not recommended for MISRA C:2012	Checks analysis now includes the content in library linked blocks and masked subsystems.
Check for blocks not recommended for C/C++ production code deployment	

For information about MISRA C® versions and updates, see MISRA C Guidelines.

Secure Coding Modeling Checks: Update to Secure Coding compliance checks

Modifications to existing secure coding checks are outlined in this table.

Model Advisor Check	Description of Change
Check for bitwise operations on signed integers	Check now identifies blocks in the top model only.
Check for blocks not recommended for C/C++ production code deployment	Checks analysis now includes the content in library linked blocks and masked subsystems.

For information about MISRA C versions and updates, see MISRA C Guidelines.

Enhanced Edit-Time Checking Support: Edit-time checking for blocks not recommended for C/C++ production code deployment

In R2017b, you could run the Model Advisor check `mathworks.codegen.PCGSupport` to identify blocks that were not supported by code generation or were not recommended for C/C++ production code deployment. In R2018a, you can use edit-time checking to identify these blocks earlier on in the design process. For more information, see [Check for Compliance Using the Model Advisor and Edit-Time Checking](#).

Model Advisor Support for Inactive Variants: Run Model Advisor checks on active and inactive variants and generate report

In R2017b, you could run Model Advisor checks only on the active variant of a model. You had to manually activate the various variant choices to run the Model Advisor on different variants.

In R2018a, you can run Model Advisor checks on valid variant configurations. Use the Variant Manager to define these configurations. Set the new `Advisor.Application` class property `AnalyzeVariants` to true. The Model Advisor generates a separate HTML report of check results for each variant configuration.

Metric Engine Improvement: Collect and analyze metric data faster

In R2018a, for a given analysis, the metric engine collects and analyzes metric data faster than in R2017b. Also, when you open the Metric Dashboard for a model in which you previously generated results, the results are now loaded more quickly. For more information, see [Collect Model Metrics](#).

Model Metric APIs: Removed Model block architectural component

In R2018a, for a specified metric engine object, you can no longer collect metric data for Model blocks. However, you can still collect metric data for these Simulink objects:

- Model
- Subsystem block

- Chart
- MATLAB Function block
- Protected model

For custom metrics, in your `algorithm` method, you can no longer specify a `ComponentScope` that is a `Model` block.

Eliminating the `Model` block component does not mean that you are missing valuable data. The parent model `AggregatedValue` includes the data for the `Model` block `AggregatedValue`. The `model` block `Value` did not contain data.

For example, in R2017b, for the `sldemo_mdref_basic` model, these are the results for the `mathworks.metrics.SimulinkBlockCount`.

```
ComponentPath: sldemo_mdref_basic
  Value: 12
  AggregatedValue: 66
ComponentPath: sldemo_mdref_basic/CounterA
  Value: NaN
  AggregatedValue: 18
ComponentPath: sldemo_mdref_basic/CounterB
  Value: NaN
  AggregatedValue: 18
ComponentPath: sldemo_mdref_basic/CounterC
  Value: NaN
  AggregatedValue: 18
ComponentPath: sldemo_mdref_basic/More Info
  Value: 0
  AggregatedValue: 0
ComponentPath: sldemo_mdref_counter
  Value: 18
  AggregatedValue: 18
```

The three instances of the referenced model `sldemo_mdref_counter` (that is `Counter A`, `Counter B`, and `Counter C`) have results. They have a `Value` of `NaN` and the `sldemo_mdref_basic` results include their aggregated values.

In R2018a, for the `sldemo_mdref_basic` model, these are the results for the `mathworks.metrics.SimulinkBlockCount` metric:

```
ComponentPath: sldemo_mdref_basic
  Value: 12
  AggregatedValue: 66
ComponentPath: sldemo_mdref_basic/More Info
  Value: 0
  AggregatedValue: 0
ComponentPath: sldemo_mdref_counter
  Value: 18
  AggregatedValue: 18
```

The results do not contain the individual instances of `sldemo_mdref_counter`. The aggregated value of `sldemo_mdref_basic` results still includes their aggregated values.

R2017b

Version: 4.0

New Features

Compatibility Considerations

Simulink Verification and Validation Packaging: Moved compliance checking, model metrics, clone detection and refactoring, edit-time checking and model transformer to Simulink Check

As of R2017b, Simulink Verification and Validation™ transitions to three new products, Simulink Requirements, Simulink Coverage, and Simulink Check.

- Requirements traceability and Requirements Management Interface (RMI) functionality have moved to the Simulink Requirements product.
- Model and generated code coverage functionality, and component verification functions such as `slvnvmakeharness`, have moved to the Simulink Coverage product.
- Compliance checking, model metrics, clone detection and refactoring, and model transformer functionality have moved to the Simulink Check product.

Metrics Dashboard: Collect and view metric data for quality assessment

The Metrics Dashboard collects and integrates quality metric data from multiple Model-Based Design tools to provide you with an assessment of your project quality status. In R2017b, by using the dashboard, you can collect and explore metric data for:

- Model size
- Modeling guidelines compliance
- Model componentization and clone detection

To explore the data in more detail, click an individual metric. For your selected metric, a table displays the value, aggregated value, and measures (if applicable) at the model component level. From the table, the dashboard provides traceability and hyperlinks to the data source so that you can get detailed results and recommended actions for troubleshooting issues.

Open the Metrics Dashboard from the model editor window by selecting **Analysis > Metrics Dashboard**. Or, at the command line, enter `metricsdashboard(system)`.

For more information, see [Collect and Explore Metric Data by Using the Metrics Dashboard](#).

MathWorks High-Integrity Guidelines and Checks: Verify compliance with safety standards by using high-integrity checks and guidelines

Categorization of the Model Advisor Checks for High-Integrity Systems

You can use the Model Advisor to check compliance with safety standards by using the high-integrity checks. To execute these checks, Open the Model Advisor (Simulink) and select the safety standard:

- **By Task > Modeling Standards for DO-178/DO-331 > High-Integrity Systems**
- **By Task > Modeling Standards for EN 50128 > High-Integrity Systems**
- **By Task > Modeling Standards for IEC 61508 > High-Integrity Systems**
- **By Task > Modeling Standards for IEC 62304 > High-Integrity Systems**
- **By Task > Modeling Standards for ISO 26262 > High-Integrity Systems**

The high-integrity checks are categorized into these subgroups:

- Simulink
- Stateflow
- MATLAB
- Configuration
- Requirements
- Code

High-Integrity Model Advisor Checks for DO-178C/DO-331 Standards

The following table identifies the Model Advisor checks that have been introduced in R2017b to check compliance with safety standards DO-178C/DO-331.

These checks are available at **By Task > Modeling Standards for DO-178/DO-331 > High-Integrity Systems**. The high-integrity subgroup in which the check resides is defined in the table.

High-Integrity Systems Subgroup	Check Name
Simulink	Check for root Inports with missing properties
Simulink	Check for root Inports with missing range definitions
Stateflow	Check Stateflow charts for transition paths that cross parallel state boundaries
Stateflow	Check Stateflow charts for strong data typing
Stateflow	Check usage of shift operations for Stateflow data
Stateflow	Check assignment operations in Stateflow Charts
Stateflow	Check Stateflow charts for unary operators
Stateflow	Check usage of Stateflow constructs
Configuration	Check safety-related solver settings for simulation time
Configuration	Check safety-related solver settings for solver options
Configuration	Check safety-related solver settings for tasking and sample-time
Configuration	Check safety-related diagnostic settings for Merge blocks
Configuration	Check safety-related diagnostic settings for Stateflow
Configuration	Check safety-related optimization settings for Loop unrolling threshold
Code	Check for blocks not recommended for MISRA C:2012
Code	Check configuration parameters for MISRA C:2012

The following table identifies modifications to existing Model Advisor checks for DO-178C/DO-331 safety standards.

Model Advisor Check	Description of Change
Check for model elements that do not link to requirements	Check title has been updated. In previous releases, the title of this check was Check for blocks that do not link to requirements . The check ID did not change.
Check model for block upgrade issues	No longer available as a Modeling Standards for DO-178C/DO-331 check. For more information, see "DO-178C/DO-331 Modeling Checks: Removed Model Advisor check "Check model for block upgrade issues"" on page 12-14.

High-Integrity Model Advisor Checks for EN 50128, IEC 61508, IEC 62304, and ISO 26262 Standards

The following table identifies the Model Advisor checks that have been introduced in R2017b to check compliance with safety standards EN 50128, IEC 61508, IEC 62304, and ISO 26262.

These checks are available at:

- **By Task > Modeling Standards for EN 50128 > High-Integrity Systems**
- **By Task > Modeling Standards for IEC 61508 > High-Integrity Systems**
- **By Task > Modeling Standards for IEC 62304 > High-Integrity Systems**
- **By Task > Modeling Standards for ISO 26262 > High-Integrity Systems**

The high-integrity subgroup in which the check resides is defined in the table.

High-Integrity Systems Subgroup	Check Name
Simulink	Check usage of lookup table blocks
Simulink	Check for blocks not recommended for C/C++ production code deployment
Simulink	Check for variant blocks with 'Generate preprocessor conditionals' active
Simulink	Check usage of Signal Routing blocks
Stateflow	Check Stateflow charts for transition paths that cross parallel state boundaries
Stateflow	Check Stateflow charts for ordering of states and transitions
Stateflow	Check Stateflow debugging options
Stateflow	Check Stateflow charts for uniquely defined data objects
Stateflow	Check Stateflow charts for strong data typing
Stateflow	Check usage of shift operations for Stateflow data
Stateflow	Check assignment operations in Stateflow Charts
Stateflow	Check Stateflow charts for unary operators
Stateflow	Check usage of Stateflow constructs
Configuration	Check safety-related optimization settings

High-Integrity Systems Subgroup	Check Name
Configuration	Check safety-related model referencing settings
Configuration	Check safety-related code generation settings
Configuration	Check safety-related diagnostic settings for solvers
Configuration	Check safety-related solver settings for simulation time
Configuration	Check safety-related solver settings for solver options
Configuration	Check safety-related solver settings for tasking and sample-time
Configuration	Check safety-related diagnostic settings for sample time
Configuration	Check safety-related diagnostic settings for signal data
Configuration	Check safety-related diagnostic settings for parameters
Configuration	Check safety-related diagnostic settings for data used for debugging
Configuration	Check safety-related diagnostic settings for data store memory
Configuration	Check safety-related diagnostic settings for type conversions
Configuration	Check safety-related diagnostic settings for signal connectivity
Configuration	Check safety-related diagnostic settings for bus connectivity
Configuration	Check safety-related diagnostic settings that apply to function-call connectivity
Configuration	Check safety-related diagnostic settings for compatibility
Configuration	Check safety-related diagnostic settings for model initialization
Configuration	Check safety-related diagnostic settings for model referencing
Configuration	Check safety-related diagnostic settings for saving
Configuration	Check safety-related diagnostic settings for Merge blocks
Configuration	Check safety-related diagnostic settings for Stateflow
Configuration	Check safety-related optimization settings for Loop unrolling threshold
Requirements	Check for model elements that do not link to requirements
Code	Check configuration parameters for MISRA C:2012
Code	Check for blocks not recommended for MISRA C:2012

High-Integrity Modeling Guidelines

High-integrity system modeling guideline hisl_0070: Placement of requirement links in a model was introduced in R2017b.

These high-integrity system modeling guidelines were removed in R2017b:

- hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)
- hisf_0012: Chart comments

The high-integrity system modeling guidelines in this table were updated to include new Model Advisor checks for DO-178C/DO-331, EN 50128, IEC 61508, IEC 62304, and ISO 26262 safety standards. Where applicable, the table also identifies additional modifications.

For a complete list of high-integrity system modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines (Simulink).

High-Integrity Modeling Guideline	Description of Change
hisl_0002: Usage of Math Function blocks (rem and reciprocal)	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of Math Operations blocks
hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of Math Operations blocks
hisl_0005: Usage of Product blocks	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for signal data
hisl_0013: Usage of data store blocks	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for data store memory
hisl_0018: Usage of Logical Operator block	Removed Model Advisor check Check safety-related optimization settings as it is covered via the prerequisite guideline.
hisl_0020: Blocks not recommended for MISRA C:2012 compliance	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check for blocks not recommended for MISRA C:2012 • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check for blocks not recommended for MISRA C:2012 • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check for blocks not recommended for C/C++ production code deployment <p>Added:</p> <ul style="list-style-type: none"> • Added From Workspace and S-Function Builder blocks to the list of blocks not recommended for MISRA compliance • Identified the deprecated Lookup Table blocks (Lookup and Lookup2D).
hisl_0022: Data type selection for index signals	Removed n-D Lookup Table (internal type index selection) from the list of blocks that use a signal index.
hisl_0023: Verification of model and subsystem variants	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check for variant blocks with 'Generate preprocessor conditionals' active

High-Integrity Modeling Guideline	Description of Change
hisl_0024: Inport interface definition	<p>New Model Advisor check for DO-178C/DO-331: Check for root Inports with missing properties</p> <p>Updated guideline description to include Simulink signal object that explicitly resolves to the connected signal line.</p>
hisl_0025: Design min/max specification of input interfaces	New Model Advisor check for DO-178C/DO-331: Check for root Inports with missing range definitions
hisl_0026: Design min/max specification of output interfaces	New Model Advisor check for DO-178C/DO-331: Check for root Outports with missing range definitions
hisl_0033: Usage of Lookup Table blocks	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of lookup table blocks
hisl_0034: Usage of Signal Routing blocks	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of Signal Routing blocks
hisl_0036: Configuration Parameters > Diagnostics > Saving	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for saving
hisl_0037: Configuration Parameters > Model Referencing	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related model referencing settings
hisl_0038: Configuration Parameters > Code Generation > Comments	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related code generation settings
hisl_0039: Configuration Parameters > Code Generation > Interface	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related code generation settings
hisl_0040: Configuration Parameters > Solver > Simulation time	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related solver settings for simulation time • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related solver settings for simulation time
hisl_0041: Configuration Parameters > Solver > Solver options	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related solver settings for solver options • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related solver settings for solver options
hisl_0042: Configuration Parameters > Solver > Tasking and sample time options	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related solver settings for tasking and sample-time • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related solver settings for tasking and sample-time

High-Integrity Modeling Guideline	Description of Change
hisl_0043: Configuration Parameters > Diagnostics > Solver	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for solvers
hisl_0044: Configuration Parameters > Diagnostics > Sample Time	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for sample time
hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0046: Configuration Parameters > Optimization > Block reduction	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0047: Configuration Parameters > Code Generation > Code Style	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related code generation settings
hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0049: Configuration Parameters > Code Generation > Symbols	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related code generation settings
hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold	New Model Advisor checks: <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related optimization settings for Loop unrolling threshold • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings for Loop unrolling threshold
hisl_0052: Configuration Parameters > Optimization > Data initialization	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related optimization settings
hisl_0060: Configuration parameters that improve MISRA C:2012 compliance	New Model Advisor checks: <ul style="list-style-type: none"> • DO-178C/DO-331: Check configuration parameters for MISRA C:2012 • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check configuration parameters for MISRA C:2012

High-Integrity Modeling Guideline	Description of Change
hisl_0061: Unique identifiers for clarity	New Model Advisor checks: <ul style="list-style-type: none"> • DO-178C/DO-331: Check usage of Stateflow constructs • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow charts for uniquely defined data objects
hisl_0301: Configuration Parameters > Diagnostics > Compatibility	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for compatibility
hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for parameters
hisl_0303: Configuration Parameters > Diagnostics > Merge block	New Model Advisor checks: <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related diagnostic settings for Merge blocks • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for Merge blocks
hisl_0304: Configuration Parameters > Diagnostics > Model initialization	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for model initialization
hisl_0305: Configuration Parameters > Diagnostics > Debugging	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for data used for debugging
hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for signal connectivity
hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for bus connectivity
hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings that apply to function-call connectivity
hisl_0309: Configuration Parameters > Diagnostics > Type Conversion	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for type conversions Added Type Conversion parameters: <ul style="list-style-type: none"> • Unnecessary type conversion • 32-bit integer to single precision float conversion
hisl_0310: Configuration Parameters > Diagnostics > Model Referencing	New Model Advisor check for IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for model referencing

High-Integrity Modeling Guideline	Description of Change
hisl_0311: Configuration Parameters > Diagnostics > Stateflow	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check safety-related diagnostic settings for Stateflow • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check safety-related diagnostic settings for Stateflow <p>The rationale from hisf_0010: Usage of transition paths (looping out of parent of source and destination objects) was incorporated into this guideline.</p>
hisf_0002: User-specified state/transition execution order	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check usage of Stateflow constructs • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow charts for ordering of states and transitions • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of Stateflow constructs
hisf_0009: Strong data typing (Simulink and Stateflow boundary)	<p>New Model Advisor check for DO-178C/DO-331: Check usage of Stateflow constructs</p>
hisf_0011: Stateflow debugging settings	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check usage of Stateflow constructs • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow debugging options • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of Stateflow constructs
hisf_0013: Usage of transition paths (crossing parallel state boundaries)	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check Stateflow charts for transition paths that cross parallel state boundaries • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow charts for transition paths that cross parallel state boundaries
hisf_0015: Strong data typing (casting variables and parameters in expressions)	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check Stateflow charts for strong data typing • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow charts for strong data typing
hisf_0064: Shift operations for Stateflow data to improve code compliance	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> • DO-178C/DO-331: Check usage of shift operations for Stateflow data • IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check usage of shift operations for Stateflow data <p>Title update. No change to guideline content.</p>

High-Integrity Modeling Guideline	Description of Change
hisf_0065: Type cast operations in Stateflow to improve code compliance	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> DO-178C/DO-331: Check assignment operations in Stateflow charts IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check assignment operations in Stateflow charts <p>Title update. No change to guideline content.</p>
hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance	<p>New Model Advisor checks:</p> <ul style="list-style-type: none"> DO-178C/DO-331: Check Stateflow charts for unary operators IEC 61508, IEC 62304, EN 50128, and ISO 26262: Check Stateflow charts for unary operators <p>Title update. No change to guideline content.</p>

Modeling Support for Secure Coding Standards: Check model for compliance with secure coding requirements in CERT C, CWE, ISO/IEC TS 17961 standards to improve security of generated code

You can use Model Advisor to check the model or subsystem for compliance with secure coding requirements in CERT C, CWE, and ISO/IEC TS 17961 standards. To execute these checks, Select and Run Model Advisor Checks (Simulink) and select **By Task > Modeling Guidelines for Secure Coding (CERT C, CWE, ISO/IEC TS 17961)**.

This table summarizes the Modeling Standards for Secure Coding checks.

Check	Description	Addresses Secure Coding Standards
Check configuration parameters for secure coding standards	Identifies configuration parameters that might impact code security.	
Check for blocks not recommended for C/C++ production code deployment	Identifies blocks not supported by code generation or not recommended for C/C++ production code deployment.	
Check for blocks not recommended for secure coding standards	Identifies blocks not supported by secure coding standards.	
Check usage of Assignment blocks	Identifies Assignment blocks that do not have block parameter Action if any output element is not assigned set to Error or Warning	<ul style="list-style-type: none"> ISO/IEC TS 17961: 2013, uninitref CERT C, EXP33-C CWE, CWE-908
Check for switch case expressions without a default case	Identifies switch case expressions that do not have a default case.	<ul style="list-style-type: none"> ISO/IEC TS 17961: 2013, swtchdflt CERT C, MSC01-C CWE, CWE-478

Check	Description	Addresses Secure Coding Standards
Check for bitwise operations on signed integers	Identifies Simulink blocks that contain bitwise operations on signed integers. The check does not flag MATLAB Function or Stateflow blocks that use signed operands for bitwise operators.	<ul style="list-style-type: none"> CERT C, INT13-C CWE, CWE-682
Check for equality and inequality operations on floating-point values	Identifies equality and inequality operations on floating-point values.	<ul style="list-style-type: none"> CERT C, FLP00-C CWE, CWE-697
Check integer word length	Identifies integer word lengths that do not comply with hardware implementation settings.	<ul style="list-style-type: none"> CERT C, INT13-C CWE, CWE-682

If you have Simulink Design Verifier, the following design error detection checks are also available as part of the Modeling Standards for Secure Coding checks.

Check	Description	Addresses Secure Coding Standards
Detect Dead Logic	Identifies logic that stays inactive during simulation.	<ul style="list-style-type: none"> CERT C, MSC07-C CWE, CWE-561
Detect Integer Overflow	Identifies operations that exceed the data type range for integer or fixed-point operations.	<ul style="list-style-type: none"> ISO/IEC TS 17961: 2013, intoflow CERT C, INT30-C and INT32-C CWE, CWE-190
Detect Division by Zero	Identifies operations in the model that cause division-by-zero errors.	<ul style="list-style-type: none"> ISO/IEC TS 17961: 2013, diverr CERT C, INT33-C and FLP03-C CWE, CWE-369
Detect Out Of Bound Array Access	Detects operations that access outside the bounds of an array index	<ul style="list-style-type: none"> ISO/IEC TS 17961: 2013, invptr CERT C, ARR30-C CWE, CWE-118
Detect Violation of Specified Minimum and Maximum Values	Checks the specified minimum and maximum values (the design ranges) on intermediate signals throughout the model and on the output ports. If the analysis detects that a signal exceeds the design range, the results identify where in the model the errors occurred.	<ul style="list-style-type: none"> CERT C, API00-C CWE, CWE-628

For information about the secure coding standards organizations, see Secure Coding Standards (Embedded Coder).

MISRA C: 2012 Modeling Checks: Improve compliance of generated code by using new MISRA C:2012 standards checks

To improve MISRA C:2012 compliance, these new checks are available through the Model Advisor. To execute these checks, Select and Run Model Advisor Checks (Simulink) and select **By Task > Modeling Guidelines for MISRA C:2012**.

Check	Description	Addresses MISRA C:2012
Check for missing error ports for AUTOSAR receiver interfaces	Identifies AUTOSAR receiver interface inports that do not have matching error ports.	Directive 4.7
Check for missing const qualifiers in model functions	Identifies input data pointers that do not have a const qualifier.	Rule 8.13
Check integer word length	Identifies integer word lengths that do not comply with hardware implementation settings.	Rule 10.1

Modifications to existing MISRA C:2012 compliance checks are outlined in this table.

Check	Description of Modification to the Check
Check for blocks not recommended for MISRA C:2012	Flags the inclusion of From Workspace blocks
Check configuration parameters for MISRA C:2012	<p>Flags the following parameter settings:</p> <ul style="list-style-type: none"> • Configuration parameter Wrap on overflow is set to none. • Configuration parameter Inf or NaN block output is set to none • Configuration parameter Dynamic memory allocation in MATLAB Function blocks is selected. • Parameter <code>ERTFilePackagingFormat</code> is set to <code>Modular</code>. • Parameter <code>PreserveStaticInFcnDecls</code> is set to <code>off</code>. <p>hisl_0060: Configuration parameters that improve MISRA C:2012 compliance reflects these parameter settings.</p>
Check for switch case expressions without a default case	<p>Check can be executed on library models.</p> <p>Check can exclude blocks when you have Simulink Check.</p>
Check for bitwise operations on signed integers	Check can exclude blocks when you have Simulink Check.

Check	Description of Modification to the Check
Check for equality and inequality operations on floating-point values	Check can exclude blocks when you have Simulink Check.

For information about MISRA C versions and updates, see MISRA C Guidelines (Embedded Coder).

DO-178C/DO-331 Modeling Checks: Removed Model Advisor check "Check model for block upgrade issues"

In R2017b, Model Advisor check Check model for block upgrade issues (check ID `mathworks.design.Update`) is no longer available under **Analysis > Model Advisor > Modeling Standards for DO-178C/DO-331 > Simulink**.

You can still execute this check through the Upgrade Advisor (Simulink) at **Analysis > Model Advisor > Upgrade Advisor**.

Model Metrics: Evaluate model quality by using new metric algorithms

Evaluate model quality by using these new model metrics:

- Simulink diagnostic warning count: Measures the number of Simulink diagnostic warnings reported during model compilation for simulation.
- Parameter count: Measures the number of parameters in a model.
- Simulink clone count: Measures the number of clones in a model.
- Clone component content: Quantifies cloned content in the model.
- Library linked component content: Quantifies library-linked content in the model.
- Stateflow chart count: Measures the number of Stateflow charts at the model level.
- MatlabFunction count: Measures the number of MATLAB Function blocks at the model level.
- Explicit IO count: Measures the number of inports and outports to and from the model.
- File Count: Measures the number of model and library files.
- Model file count: Measures the number of model files.

For more information on these, and other available metric algorithms, see Model Metrics.

Model Metric APIs: Create custom metrics with more detailed results and determine passed or failed compliance checks

In R2017b, the `slmetric.metric.Result` class contains the new property `Details`. `Details` is an array of objects of the new class `slmetric.metric.ResultDetail`. You can write custom metrics that use this new class to store details about what the `Value` property of the `slmetric.metric.Result` object counts. You can also use this class to determine which MAAB and DO-178C/DO-331 metrics passed or failed.

For existing classes, there are these new properties:

Class	New Property
slmetric.metric.ResultDetail	Details
slmetric.metric.Result	ID
slmetric.metric.ResultCollection	Outdated
slmetric.metric.Metric	SupportsResultDetails

For more information see, `slmetric.metric.ResultDetail`.

Compatibility Considerations

In R2017b, you cannot collect metric data for MISRA C:2012 and ISO 26262 metrics. Specifically, these metrics are not available:

- `mathworks.metrics.ModelAdvisorCheckCompliance.misra_c`
- `mathworks.metrics.ModelAdvisorCheckCompliance.ISO26262`
- `mathworks.metrics.ModelAdvisorCheckIssues.misra_c`
- `mathworks.metrics.ModelAdvisorCheckIssues.ISO26262`

For the DO-178C/DO-331 compliance metrics, the metric IDs `mathworks.metrics.ModelAdvisorCheckCompliance.do178` and `mathworks.metrics.ModelAdvisorCheckIssues.do178` are now named `mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178` and `mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178`.

Model Advisor Configuration Editor: Select edit-time checks from folders

In the Model Advisor Configuration Editor, the tool now lists edit-time checks in folders instead of in a flat list. The folder structure is the same folder structure as for the Model Advisor. The Model Advisor Configuration Editor includes only folders that contain edit-time checks.

For more information, see [Organize Checks and Folders Using the Model Advisor Configuration Editor](#).

