Simulink® Check™

Reference

MATLAB® & SIMULINK®

R2017b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

Simulink® Check™ Reference

Revision History

September 2017	Online only	New for Version 4.0 (Release 2017b)

Contents

1 Functions — Alphabetical List

2 Model Advisor Checks

3 Model Transformer Tasks

4 Clone Detection Tasks

Functions — Alphabetical List

actionCallback

Class: Advisor.authoring.CustomCheck
Package: Advisor.authoring

Register action callback for model configuration check

Syntax

`Advisor.authoring.CustomCheck.actionCallback(task)`

Description

`Advisor.authoring.CustomCheck.actionCallback(task)` is used as the action callback function when registering custom checks that use an XML data file to specify check behavior.

Examples

This `sl_customization.m` file registers the action callback for configuration parameter checks with fix actions.

```
function defineModelAdvisorChecks

    rec = ModelAdvisor.Check('com.mathworks.Check1');
    rec.Title = 'Test: Check1';
    rec.setCallbackFcn(@(system)(Advisor.authoring.CustomCheck.checkCallback(system)), …
            'None', 'StyleOne');
    rec.TitleTips = 'Example check for check authoring infrastructure.';

    % --- data file input parameters
    rec.setInputParametersLayoutGrid([1 1]);
    inputParam1 = ModelAdvisor.InputParameter;
    inputParam1.Name = 'Data File';
    inputParam1.Value = 'Check1.xml';
    inputParam1.Type = 'String';
    inputParam1.Description = 'Name or full path of XML data file.';
    inputParam1.setRowSpan([1 1]);
    inputParam1.setColSpan([1 1]);
    rec.setInputParameters({inputParam1});
```

```
    % -- set fix operation
    act = ModelAdvisor.Action;
    act.setCallbackFcn(@(task)(Advisor.authoring.CustomCheck.actionCallback(task)));
    act.Name = 'Modify Settings';
    act.Description = 'Modify model configuration settings.';
    rec.setAction(act);

    mdladvRoot = ModelAdvisor.Root;
    mdladvRoot.register(rec);
end
```

See Also

`Advisor.authoring.CustomCheck.checkCallback` | `Advisor.authoring.DataFile` | `Advisor.authoring.generateConfigurationParameterDataFile`

Topics

"Create Check for Model Configuration Parameters"

addCheck

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Add check to folder

Syntax

`addCheck(fg_obj, check_ID)`

Description

`addCheck(fg_obj, check_ID)` adds checks, identified by `check_ID`, to the folder specified by `fg_obj`, which is an instantiation of the `ModelAdvisor.FactoryGroup` class.

Examples

Add three checks to `rec`:

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
.
.
.
addCheck(rec, 'com.mathworks.sample.Check1');
addCheck(rec, 'com.mathworks.sample.Check2');
addCheck(rec, 'com.mathworks.sample.Check3');
```

addGroup

Class: ModelAdvisor.Group
Package: ModelAdvisor

Add subfolder to folder

Syntax

```
addGroup(group_obj, child_obj)
```

Description

`addGroup(group_obj, child_obj)` adds a new subfolder, identified by `child_obj`, to the folder specified by `group_obj`, which is an instantiation of the `ModelAdvisor.Group` class.

Examples

Add three checks to `rec`:

```
group_obj = ModelAdvisor.Group('com.mathworks.sample.group');
.
.
.
addGroup(group_obj, 'com.mathworks.sample.subgroup1');
addGroup(group_obj, 'com.mathworks.sample.subgroup2');
addGroup(group_obj, 'com.mathworks.sample.subgroup3');
```

To add `ModelAdvisor.Task` objects to a group using `addGroup`:

```
mdladvRoot = ModelAdvisor.Root();

% MAT1, MAT2, and MAT3 are registered ModelAdvisor.Task objects
% Create the group 'My Group'
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
```

```
MAG.DisplayName='My Group';

% Add the first task to the 'My Group' folder
MAG.addTask(MAT1);

% Create a subfolder 'Folder1'
MAGSUB1 = ModelAdvisor.Group('com.mathworks.sample.Folder1');
MAGSUB1.DisplayName='Folder1';

% Add the second task to Folder1
MAGSUB1.addTask(MAT2);

% Create a subfolder 'Folder2'
MAGSUB2 = ModelAdvisor.Group('com.mathworks.sample.Folder2');
MAGSUB2.DisplayName='Folder2';

% Add the third task to Folder2
MAGSUB2.addTask(MAT3);

% Register the two subfolders. This must be done before calling addGroup
mdladvRoot.register(MAGSUB1);
mdladvRoot.register(MAGSUB2);

% Invoke addGroup to place the subfolders under 'My Group'
MAG.addGroup(MAGSUB1);
MAG.addGroup(MAGSUB2);

mdladvRoot.publish(MAG); % publish under Root
```

addItem

Class: ModelAdvisor.List
Package: ModelAdvisor

Add item to list

Syntax

`addItem(element)`

Description

`addItem(element)` adds items to the list created by the `ModelAdvisor.List` constructor.

Input Arguments

element	Specifies an element to be added to a list in one of the following: • Element • Cell array of elements. When you add a cell array to a list, they form different rows in the list. • Character vector

Examples

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”

addItem

Class: ModelAdvisor.Paragraph
Package: ModelAdvisor

Add item to paragraph

Syntax

`addItem(text, element)`

Description

`addItem(text, element)` adds an element to `text`. `element` is one of the following:

- Character vector
- Element
- Cell array of elements

Examples

Add two lines of text:

```
result = ModelAdvisor.Paragraph;
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

addProcedure

Class: ModelAdvisor.Group
Package: ModelAdvisor

Add procedure to folder

Syntax

`addProcedure(group_obj, procedure_obj)`

Description

`addProcedure(group_obj, procedure_obj)` adds a procedure, specified by `procedure_obj`, to the folder `group_obj`. `group_obj` is an instantiation of the `ModelAdvisor.Group` class.

Examples

Add three procedures to `MAG`.

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');

MAP1=ModelAdvisor.Procedure('com.mathworks.sample.procedure1');
MAP2=ModelAdvisor.Procedure('com.mathworks.sample.procedure2');
MAP3=ModelAdvisor.Procedure('com.mathworks.sample.procedure3');

addProcedure(MAG, MAP1);
addProcedure(MAG, MAP2);
addProcedure(MAG, MAP3);
```

addProcedure

Class: ModelAdvisor.Procedure
Package: ModelAdvisor

Add subprocedure to procedure

Syntax

`addProcedure(procedure1_obj, procedure2_obj)`

Description

`addProcedure(procedure1_obj, procedure2_obj)` adds a procedure, specified by `procedure2_obj`, to the procedure `procedure1_obj`. `procedure2_obj` and `procedure1_obj` are instantiations of the `ModelAdvisor.Procedure` class.

Examples

Add three procedures to `MAP`.

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');

MAP1=ModelAdvisor.Procedure('com.mathworks.sample.procedure1');
MAP2=ModelAdvisor.Procedure('com.mathworks.sample.procedure2');
MAP3=ModelAdvisor.Procedure('com.mathworks.sample.procedure3');

addProcedure(MAP, MAP1);
addProcedure(MAP, MAP2);
addProcedure(MAP, MAP3);
```

addRow

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add row to table

Syntax

`addRow(ft_obj, {item1, item2, ..., itemn})`

Description

`addRow(ft_obj, {item1, item2, ..., itemn})` is an optional method that adds a row to the end of a table in the result. *`ft_obj`* is a handle to the template object previously created. `{item1, item2, ..., itemn}` is a cell array of character vectors and objects to add to the table. The order of the items in the array determines which column the item is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

Note Before adding rows to a table, you must specify column titles using the `setColTitle` method.

Examples

Find all of the blocks in the model and create a table of the blocks:

```
% Create FormatTemplate object, specify table format
ft = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information to the table
setTableTitle(ft, {'Blocks in Model'});
setColTitles(ft, {'Index', 'Block Name'});
% Find all the blocks in the system and add them to a table.
allBlocks = find_system(system);
for inx = 2 : length(allBlocks)
    % Add information to the table
```

```
    addRow(ft, {inx-1,allBlocks(inx)});
end
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

addTask

Class: ModelAdvisor.Group
Package: ModelAdvisor

Add task to folder

Syntax

```
addTask(group_obj, task_obj)
```

Description

`addTask(group_obj, task_obj)` adds a task, specified by `task_obj`, to the folder `group_obj`.`group_obj` is an instantiation of the `ModelAdvisor.Group` class.

Examples

Add three tasks to `MAG`.

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
addTask(MAG, MAT1);
addTask(MAG, MAT2);
addTask(MAG, MAT3);
```

addTask

Class: ModelAdvisor.Procedure
Package: ModelAdvisor

Add task to procedure

Syntax

`addTask(procedure_obj, task_obj)`

Description

`addTask(procedure_obj, task_obj)` adds a task, specified by `task_obj`, to `procedure_obj`.`procedure_obj` is an instantiation of the `ModelAdvisor.Procedure` class.

Examples

Add three tasks to `MAP`.

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');

MAT1=ModelAdvisor.Task('com.mathworks.sample.task1');
MAT2=ModelAdvisor.Task('com.mathworks.sample.task2');
MAT3=ModelAdvisor.Task('com.mathworks.sample.task3');

addTask(MAP, MAT1);
addTask(MAP, MAT2);
addTask(MAP, MAT3);
```

Advisor.Application class

Package: Advisor

Run Model Advisor across model hierarchy

Description

Use instances of `Advisor.Application` to run Model Advisor checks across a model hierarchy. You can use `Advisor.Application` to:

- Run checks on referenced models.
- Select model components for Model Advisor analysis.
- Select checks to run during Model Advisor analysis.

Consider using `Advisor.Application` if you have a large model with subsystems and model references. `Advisor.Application` does not run checks on library models. If you want to run checks on multiple independent models that are not in a model reference hierarchy or you want to leverage parallel processing, use `ModelAdvisor.run` to run Model Advisor checks on your model.

The `Advisor.Application` methods use the following definitions:

- Model component — Model in the system hierarchy. Models that the root model references and that `setAnalysisroot` specifies are model components.
- Check instance — Instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. Each check instance has an instance ID. When you change the Model Advisor configuration, the instance ID can change.

Construction

To create an `Advisor.Application` object, use `Advisor.Manager.createApplication`.

Properties

`AnalysisRoot` — Name of root model in the model hierarchy to analyze
character vector

Name of root model in the model hierarchy to analyze, as specified by the `Advisor.Application.setAnalysisRoot` method. This property is read only.

`ID` — Unique identifier
character vector

Unique identifier for the `Advisor.Application` object. This property is read only.

`UseTempDir` — Run analysis in a temporary working folder
false (default) | true

Run analysis in a temporary working folder. Specified by the `Advisor.Manager.createApplication` method. This property is read only.

Data Types: `logical`

Methods

delete	Delete Advisor.Application object
deselectCheckInstances	Clear check instances from Model Advisor analysis
deselectComponents	Clear model components from Model Advisor analysis
generateReport	Generate report for Model Advisor analysis
getCheckInstanceIDs	Obtain check instance IDs
getResults	Access Model Advisor analysis results
loadConfiguration	Load Model Advisor configuration
run	Run Model Advisor analysis on model components
selectCheckInstances	Select check instances to use in Model Advisor analysis
selectComponents	Select model components for Model Advisor analysis
setAnalysisRoot	Specify model hierarchy for Model Advisor analysis

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects (MATLAB).

Examples

Run Model Advisor Checks on Referenced Model

This example shows how to run a check on model `sldemo_mdlref_counter` referenced from `sldemo_mdlref_basic`.

1. In the Command Window, open model `sldemo_mdlref_basic` and referenced model `sldemo_mdlref_counter`.

   ```
   open_system('sldemo_mdlref_basic');
   open_system('sldemo_mdlref_counter');
   ```
2. Save a copy of the models to a work folder, renaming them to `mdlref_basic` and `mdlref_counter`.

   ```
   save_system('sldemo_mdlref_basic','mdlref_basic');
   save_system('sldemo_mdlref_counter','mdlref_counter');
   ```
3. In `mdlref_basic`, change model reference from `sldemo_mdlref_counter` to `mdlref_counter`. Save `mdlref_basic`.

   ```
   set_param('mdlref_basic/CounterA','ModelName','mdlref_counter');
   set_param('mdlref_basic/CounterB','ModelName','mdlref_counter');
   set_param('mdlref_basic/CounterC','ModelName','mdlref_counter');
   save_system('mdlref_basic');
   ```
4. Set root model to `mdlref_basic`.

   ```
   RootModel='mdlref_basic';
   ```
5. Create an `Application` object.

   ```
   app = Advisor.Manager.createApplication();
   ```
6. Set root analysis.

   ```
   setAnalysisRoot(app,'Root',RootModel);
   ```
7. Clear all check instances from Model Advisor analysis.

   ```
   deselectCheckInstances(app);
   ```

8 Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
selectCheckInstances(app,'IDs',checkinstanceID);
```

9 Run Model Advisor analysis.

```
run(app);
```

10 Get analysis results.

```
getResults(app);
```

11 Generate and view the Model Advisor report. The Model Advisor runs the check on both `mdlref_basic` and `mdlref_counter`.

```
report = generateReport(app);
web(report)
```

12 Close the models.

```
close_system('mdlref_basic');
close_system('mdlref_counter');
```

Run Model Advisor Checks on a Subsystem

This example shows how to run a check on subsystem `CounterA` referenced from `sldemo_mdlref_basic`.

1 In the Command Window, open model `sldemo_mdlref_basic`.

```
open_system('sldemo_mdlref_basic');
```

2 Set root model to `sldemo_mdlref_basic`.

```
RootModel='sldemo_mdlref_basic';
```

3 Create an `Application` object.

```
app = Advisor.Manager.createApplication();
```

4 Set root analysis to subsystem `sldemo_mdlref_basic/CounterA`.

```
setAnalysisRoot(app,'Root','sldemo_mdlref_basic/CounterA','RootType','Subsystem');
```

5 Clear all check instances from Model Advisor analysis.

```
deselectCheckInstances(app);
```

6 Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
selectCheckInstances(app,'IDs',checkinstanceID);
```

7 Run Model Advisor analysis.

```
run(app);
```

8 Get analysis results.

```
getResults(app);
```

9 Generate and view the Model Advisor report. The Model Advisor runs the check on subsystem `sldemo_mdlref_basic/CounterA`.

```
report = generateReport(app);
web(report)
```

10 Close the model.

```
close_system('sldemo_mdlref_basic');
```

See Also

Topics

Class Attributes (MATLAB)
Property Attributes (MATLAB)

Introduced in R2015b

Advisor.authoring.generateConfigurationParameter DataFile

Package: Advisor.authoring

Generate XML data file for custom configuration parameter check

Syntax

```
Advisor.authoring.generateConfigurationParameterDataFile(dataFile,
source)
Advisor.authoring.generateConfigurationParameterDataFile(dataFile,
source,Name,Value)
```

Description

`Advisor.authoring.generateConfigurationParameterDataFile(dataFile, source)` generates an XML data file named `dataFile` specifying the configuration parameters for `source`. The data file uses tagging to specify the configuration parameter settings you want. When you create a check for configuration parameters, you use the data file. Each model configuration parameter specified in the data file is a subcheck.

`Advisor.authoring.generateConfigurationParameterDataFile(dataFile, source,Name,Value)` generates an XML data file named `dataFile` specifying the configuration parameters for `source`. It also specifies additional options by one or more optional `Name,Value` arguments. The data file uses tagging to specify the configuration parameter settings you want. When you create a check for configuration parameters, you use the data file. Each model configuration parameter specified in the data file is a subcheck.

Examples

Create data file for configuration parameter check

Create a data file with all the configuration parameters. You use the data file to create a configuration parameter.

```
model = 'vdp';
dataFile = 'myDataFile.xml';
Advisor.authoring.generateConfigurationParameterDataFile( ...
               dataFile, model);
```

Data file `myDataFile.xml` has tagging specifying subcheck information for each configuration parameter. `myDataFile.xml` specifies the configuration parameters settings you want. The following specifies XML tagging for configuration parameter `AbsTol`. If the configuration parameter is set to `1e-6`, the configuration parameter subcheck specified in `myDataFile.xml` passes.

```
<!-- Absolute tolerance: (AbsTol)-->
  <PositiveModelParameterConstraint>
     <parameter>AbsTol</parameter>
     <value>1e-6</value>
     </PositiveModelParameterConstraint>
```

Create data file for Solver pane configuration parameter check with fix action

Create a data file with configuration parameters for the **Solver** pane. You use the data file to create a **Solver** pane configuration parameter check with fix actions.

```
model = 'vdp';
dataFile = 'myDataFile.xml';
Advisor.authoring.generateConfigurationParameterDataFile( ...
               dataFile, model, 'Pane', 'Solver', 'FixValues', true);
```

Data file `myDataFile.xml` has tagging specifying subcheck information for each configuration parameter. `myDataFile.xml` specifies the configuration parameters settings that you want. The following specifies XML tagging for configuration parameter `AbsTol`. If the configuration parameter is set to `1e-6`, the configuration parameter subcheck specified in `myDataFile.xml` passes. If the subcheck does not pass, the check fix action modifies the configuration parameter to `1e-6`.

```
<!-- Absolute tolerance: (AbsTol)-->
  <PositiveModelParameterConstraint>
```

```
<parameter>AbsTol</parameter>
<value>1e-6</value>
<fixvalue>1e-6</fixvalue>
</PositiveModelParameterConstraint>
```

- “Create Check for Model Configuration Parameters”

Input Arguments

`dataFile` — Name of data file to create
character vector

Name of XML data file to create, specified as a character vector.

Example: `'myDataFile.xml'`

`source` — Name of model or configuration set
character vector | `Simulink.ConfigSet`

Name of model or `Simulink.ConfigSet` object used to specify configuration parameters

Example: 'vdp'

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Pane', 'Solver', 'FixValues', true` specifies a `dataFile` with Solver pane configuration parameters and fix tagging.

`Pane` — Limit the configuration parameters in the `dataFile`
`Solver` | `Data Import/Export` | `Optimization` | `Diagnostics` | `Hardware Implementation` | `Model Referencing` | `Code Generation`

Option to limit the configuration parameters in the data file to the pane specified as the comma-separated pair of `'Pane'` and one of the following:

- `Solver`
- `Data Import/Export`
- `Optimization`
- `Diagnostics`
- `Hardware Implementation`
- `Model Referencing`
- `Code Generation`

Example: `'Pane','Solver'` limits the `dataFile` to configuration parameters on the Solver pane.

Data Types: `char`

`FixValues` — Create fix tagging in the `dataFile`
false | true

Setting `FixValues` to `true` provides the `dataFile` with fix tagging. When you generate a custom configuration parameter check using a `dataFile` with fix tagging, each configuration parameter subcheck has a fix action. Specified as the comma-separated pair of `'FixValues'` and either `true` or `false`.

Example: `'FixValues,true` specifies fix tagging in the `dataFile`.

Data Types: `logical`

See Also

Topics
"Create Check for Model Configuration Parameters"
"Data File for Configuration Parameter Check"

Introduced in R2014a

Advisor.authoring.CustomCheck class

Package: Advisor.authoring

Define custom check

Description

Instances of the `Advisor.authoring.CustomCheck` class provide a container for static methods used as callback functions when defining a configuration parameter check. The configuration parameter check is defined in an XML data file.

Methods

actionCallback	Register action callback for model configuration check
checkCallback	Register check callback for model configuration check

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB® Programming Fundamentals documentation.

See Also

`Advisor.authoring.DataFile` | `Advisor.authoring.generateConfigurationParameterDataFile`

Topics

"Create Check for Model Configuration Parameters"

Advisor.authoring.DataFile class

Package: Advisor.authoring

Interact with data file for model configuration checks

Description

The `Advisor.authoring.DataFile` class provides a container for a static method used when interacting with the data file for configuration parameter checks.

Methods

validate	Validate XML data file used for model configuration check

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

`Advisor.authoring.CustomCheck` | `Advisor.authoring.generateConfigurationParameterDataFile`

Topics

"Create Check for Model Configuration Parameters"

Advisor.Manager class

Package: Advisor

Manage applications

Description

The `Advisor.Manager` class defines application objects.

Methods

createApplication	Create `Advisor.Application` object
getApplication	Return handle to `Advisor.Application` object
refresh_customizations	Refresh Model Advisor check information cache

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects (MATLAB).

See Also

Topics

Class Attributes (MATLAB)
Property Attributes (MATLAB)

Introduced in R2015b

checkCallback

Class: Advisor.authoring.CustomCheck
Package: Advisor.authoring

Register check callback for model configuration check

Syntax

`Advisor.authoring.CustomCheck.checkCallback(system)`

Description

`Advisor.authoring.CustomCheck.checkCallback(system)` is used as the check callback function when registering custom checks that use an XML data file to specify check behavior.

Examples

This `sl_customization.m` file registers a configuration parameter check using `Advisor.authoring.CustomCheck.checkCallback(system)`.

```
function defineModelAdvisorChecks

    rec = ModelAdvisor.Check('com.mathworks.Check1');
    rec.Title = 'Test: Check1';
    rec.setCallbackFcn(@(system)(Advisor.authoring.CustomCheck.checkCallback(system)), …
            'None', 'StyleOne');
    rec.TitleTips = 'Example check for check authoring infrastructure.';

    % --- data file input parameters
    rec.setInputParametersLayoutGrid([1 1]);
    inputParam1 = ModelAdvisor.InputParameter;
    inputParam1.Name = 'Data File';
    inputParam1.Value = 'Check1.xml';
    inputParam1.Type = 'String';
    inputParam1.Description = 'Name or full path of XML data file.';
    inputParam1.setRowSpan([1 1]);
    inputParam1.setColSpan([1 1]);
    rec.setInputParameters({inputParam1});
```

```
    % -- set fix operation
    act = ModelAdvisor.Action;
    act.setCallbackFcn(@(task)(Advisor.authoring.CustomCheck.actionCallback(task)));
    act.Name = 'Modify Settings';
    act.Description = 'Modify model configuration settings.';
    rec.setAction(act);

    mdladvRoot = ModelAdvisor.Root;
    mdladvRoot.register(rec);
end
```

See Also

`Advisor.authoring.CustomCheck.actionCallback` | `Advisor.authoring.DataFile` | `Advisor.authoring.generateConfigurationParameterDataFile`

Topics

"Create Check for Model Configuration Parameters"

createApplication

Class: Advisor.Manager
Package: Advisor

Create `Advisor.Application` object

Syntax

```
app = Advisor.Manager.createApplication()
app = Advisor.Manager.createApplication(Name,Value)
```

Description

`app = Advisor.Manager.createApplication()` constructs an `Advisor.Application` object.

`app = Advisor.Manager.createApplication(Name,Value)` constructs an `Advisor.Application` object that operates in a temporary working folder.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'UseTempDir',true` specifies that `Advisor.Application` object operates in a temporary working folder.

`UseTempDir` — Create `Advisor.Application` object that operates in a temporary working folder
false (default) | true

Data Types: `logical`

Output Arguments

`app` — Application
`Advisor.Application` object

Constructed `Advisor.Application` object.

See Also

`Advisor.Application` | `Advisor.Manager.getApplication`

Introduced in R2015b

delete

Class: Advisor.Application
Package: Advisor

Delete Advisor.Application object

Syntax

`delete(app)`

Description

`delete(app)` deletes the `Application` object when you close the root model specified using `Advisor.Application.setAnalysisRoot`, `Application` objects are implicitly closed.

Examples

```
app = Advisor.Manager.createApplication();
delete(app)
```

Input Arguments

`app` — `Advisor.Application` object to destroy
handle

`Advisor.Application` object to destroy, as specified by `Advisor.Manager.createApplication`.

See Also

`Advisor.Application.setAnalysisRoot` | `Advisor.Manager.createApplication`

Introduced in R2015b

deselectCheckInstances

Class: Advisor.Application
Package: Advisor

Clear check instances from Model Advisor analysis

Syntax

```
deselectCheckInstances(app)
deselectCheckInstances(app,Name,Value)
```

Description

You can clear check instances from Model Advisor analysis. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. To obtain the check instance ID, use the `getCheckInstanceIDs` method.

`deselectCheckInstances(app)` clears all check instances from Model Advisor analysis.

`deselectCheckInstances(app,Name,Value)` clears check instances specified by `Name,Value` pair arguments from Model Advisor analysis.

Input Arguments

app — Application

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`IDs` — Checks instance IDs
cell array

Check instances to clear from Model Advisor analysis, as specified by a cell array of IDs

Data Types: `cell`

Examples

Clear All Check Instances from Model Advisor Analysis

This example shows how to set the root model, create an `Application` object, set root analysis, and clear checks instances from Model Advisor analysis.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Deselect all checks
deselectCheckInstances(app);
```

Clear Check Instance from Model Advisor Analysis Using Instance ID

This example shows how to set the root model, create an `Application` object, set root analysis, and deselect checks instances using instance IDs.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();
```

```
% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Deselect "Identify unconnected lines, input ports, and output
% ports" check using instance ID
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
deselectCheckInstances(app,'IDs',checkinstanceID);
```

See Also

Advisor.Application.getCheckInstanceIDs | Advisor.Application.selectCheckInstances | Advisor.Application.setAnalysisRoot | Advisor.Manager.createApplication

Introduced in R2015b

deselectComponents

Class: Advisor.Application
Package: Advisor

Clear model components from Model Advisor analysis

Syntax

```
deselectComponents(app)
deselectComponents(app,Name,Value)
```

Description

You can clear model components from Model Advisor analysis. A model component is a model in the system hierarchy. Models that the root model references and that `Advisor.Application.setAnalysisRoot` specifies are model components.

`deselectComponents(app)` clears all components from Model Advisor analysis.

`deselectComponents(app,Name,Value)` clears model components specified by `Name,Value` pair arguments from Model Advisor analysis.

Input Arguments

app — Application
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`IDs` — Component IDs
cell array

Components to clear from Model Advisor analysis, as specified by a cell array of IDs

Data Types: `cell`

`HierarchicalSelection` — Clear component and component children
false (default) | true

Clear components specified by IDs and component children from Model Advisor analysis

Data Types: `logical`

Examples

Clear All Components from Model Advisor Analysis

This example shows how to set the root model, create an `Application` object, set root analysis, and clear all components from Model Advisor analysis.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Deselect all components
deselectComponents(app);
```

Clear Components from Model Advisor Analysis Using IDs

This example shows how to set the root model, create an `Application` object, set root analysis, and clear model components using IDs.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Deselect component using IDs
deselectComponents(app,'IDs',RootModel);
```

See Also

`Advisor.Application.selectComponents` | `Advisor.Application.setAnalysisRoot` | `Advisor.Manager.createApplication`

Introduced in R2015b

generateReport

Class: Advisor.Application
Package: Advisor

Generate report for Model Advisor analysis

Syntax

```
generateReport(app)
generateReport(app,Name,Value)
```

Description

Generate a Model Advisor report for an `Application` object analysis.

`generateReport(app)` generates a Model Advisor report for each component specified by the `Application` object. By default, a report with the name of the analysis root is generated in the current folder.

`generateReport(app,Name,Value)` generates a Model Advisor report for each component specified by the `Application` object. Use the `Name,Value` pairs to specify the location and name of the report.

Input Arguments

app — Application
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`Location` — Path to report location
character vector

`Name` — Report name
character vector

Examples

Generate Report

This example shows how to generate a report with the analysis root name in the current folder.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Run Model Advisor analysis
run(app);

% Generate report
report = generateReport(app);

% Open the report in web browser
web(report);
```

Generate Report with Specified Name and Location

This example shows how to generate a report with a specified name and location.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';
```

```
% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Run Model Advisor analysis
run(app);

% Generate report in my_work directory
mkdir my_work
report = generateReport(app,'Location','my_work','Name','RootModelReport');

%Open the report in web browser
web(report);
```

See Also

Advisor.Application.run | Advisor.Application.setAnalysisRoot | Advisor.Manager.createApplication

Introduced in R2015b

getApplication

Class: Advisor.Manager
Package: Advisor

Return handle to `Advisor.Application` object

Syntax

```
app = getApplication(Name,Value)
```

Description

`app = getApplication(Name,Value)` returns the handle to an `Advisor.Application` object by using the object properties.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Id',appID` returns handle to an `Advisor.Application` using the object ID.

`Id` — `Advisor.Application` object `ID`
`Advisor.Application` object

Data Types: `function_handle`

`Root` — Root model name
character vector

Data Types: `char`

`RootType` — Type of root analysis
`'Model'` (default) | `'Subsystem'`

Data Types: `char`

Output Arguments

`app` — Handle to `Advisor.Application` object
`Advisor.Application` object

Data Types: `function_handle`

See Also

`Advisor.Application` | `Advisor.Manager.createApplication`

Introduced in R2015b

getCheckInstanceIDs

Class: Advisor.Application
Package: Advisor

Obtain check instance IDs

Syntax

```
CheckInstanceIDs = getCheckInstanceIDs(app)
CheckInstanceIDs = getCheckInstanceIDs(app,CheckID)
```

Description

Obtain the check instance ID for a check using the check ID. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. The check ID is a static identifier that does not change.

`CheckInstanceIDs = getCheckInstanceIDs(app)` returns a cell array of IDs.

`CheckInstanceIDs = getCheckInstanceIDs(app,CheckID)` returns a instance ID for a check.

Input Arguments

`app` — Application

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

`CheckID` — Check ID associated with Model Advisor check

character vector

Check ID associated with Model Advisor check.

Example: `'mathworks.design.UnconnectedLinesPorts'`

Output Arguments

`CheckInstanceIDs` — Cell array of check instance IDs
cell array

Check instance IDs, returned as a cell array of IDs

Examples

Obtain Check Instance IDs

This example shows how to set the root model, create an `Application` object, set root analysis, and obtain the check instance ID.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select all check instances
selectCheckInstances(app);

% Obtain check instance IDs
CheckInstanceIDs = getCheckInstanceIDs(app);
```

Obtain Check Instance ID for a Check

This example shows how to set the root model, create an `Application` object, set root analysis, and obtain the check instance ID for check **Identify unconnected lines, input ports**.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select all check instances
selectCheckInstances(app);

%  Obtain check instance ID for Model Advisor check "Identify unconnected lines,
%   input ports"
CheckInstanceIDs = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
```

Alternatives

In the left-hand pane of the Model Advisor window, right-click the check and select **Send Check Instance ID to Workspace**.

See Also

`Advisor.Application.selectCheckInstances` | `Advisor.Application.setAnalysisRoot` | `Advisor.Manager.createApplication`

Introduced in R2015b

getEntry

Class: ModelAdvisor.Table
Package: ModelAdvisor

Get table cell contents

Syntax

```
content = getEntry(table, row, column)
```

Description

`content = getEntry(table, row, column)` gets the contents of the specified cell.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying the row
`column`	An integer specifying the column

Output Arguments

`content`	An element object or object array specifying the content of the table entry

Examples

Get the content of the table cell in the third column, third row:

```
table1 = ModelAdvisor.Table(4, 4);
.
.
```

```
.
content = getEntry(table1, 3, 3);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

getID

Class: ModelAdvisor.Check
Package: ModelAdvisor

Return check identifier

Syntax

```
id = getID(check_obj)
```

Description

`id = getID(check_obj)` returns the ID of the check `check_obj`. `id` is a unique identifier for the check.

You create this unique identifier when you create the check. This unique identifier is the equivalent of the `ModelAdvisor.Check ID` property.

See Also

“Model Advisor Customization”

Topics

“Define Custom Checks”
“Create Model Advisor Checks”

execute

Class: slmetric.Engine
Package: slmetric

Collect metric data

Syntax

```
execute(metric_engine)
execute(slmetric_obj,MetricIDs)
```

Description

Collect model metric data for the specified metric engine object.

`execute(metric_engine)` collects metric data for available model metrics, which can include MathWorks metrics and custom metrics.

`execute(slmetric_obj,MetricIDs)` collects metric data for only the specified metrics, which can be MathWorks metrics or custom metrics.

Input Arguments

`metric_engine` — Metric engine object
`slmetric.Engine` object

Create a `slmetric.Engine` object.

```
metric_engine = slmetric.Engine();
```

`MetricIDs` — Metric identifier
character vector | cell array of character vectors

Metric identifier for “Model Metrics” on page 2-309 or custom model metrics that you create. You can specify one or multiple metric identifiers. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

Examples

Collect and Access Metric Data for a Model

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis, these properties are on by d
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data

```
execute(metric_engine);
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
```

```
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis, these properties are on by d
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
```

```
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2016a

getAnalysisRootMetric

Class: slmetric.Engine
Package: slmetric

Get metric data for one metric for analysis root only

Syntax

```
metricResult = getAnalysisRootMetric(metric_engine,MetricID)
```

Description

Get metric data from the metric engine where the root of analysis was set using `slmetric.Engine.setAnalysisRoot`.

`metricResult = getAnalysisRootMetric(metric_engine,MetricID)` get the metric data from `metric_engine`, for a specified metric identifier, `MetricID`, only for the analysis root.

Input Arguments

`metric_engine` — Collects and accesses metric data

`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`MetricID` — Metric identifier

character vector

Metric identifier for "Model Metrics" on page 2-309 or custom model metrics, that you create. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: 'mathworks.metrics.DescriptiveBlockNames'

Output Arguments

metricResult — Result of metric analysis on the analysis root
slmetric.metric.Result object

Outputs the object of the slmetric.metric.Result object containing the result data for the requested analysis root and metric.

Examples

Collect and Access Metric Data for the Analysis Root

This example shows how to set the analysis root, collect, and access the metric data for a metric.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify the model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics for only the analysis root
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);

metricResult = getAnalysisRootMetric(metric_engine, metricID);
```

See Also

slmetric.metric.ResultCollection | slmetric.metric.getAvailableMetrics

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2017a

getErrorLog

Class: slmetric.Engine
Package: slmetric

Get error log

Syntax

```
metricLog = getErrorLog(metric_engine)
```

Description

Get a log of errors and warnings that occurred during metric data collection of a specified metric engine object. The log includes errors that occurred during the execution of metric algorithms, model compilation, and metric data validation.

`metricLog = getErrorLog(metric_engine).`

Input Arguments

`metric_engine` **— Metric engine object**
`slmetric.Engine` object

Constructed `slmetric.Engine` object.

Output Arguments

`metricLog` **— Log of metric errors and warnings**
string array

The `metricLog` string contains the errors and warnings from metric analysis and is formatted in HTML.

Examples

Get Error Log

This example shows how to create a `slmetric.Engine` object, set the analysis root, generate metrics, and create and display the error log for the model `sldemo_fuelsys`.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics for only the analysis root
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);

metricLog = getErrorLog(metricEngine);
disp(metricLog);
```

See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2017a

getMetricDistribution

Class: slmetric.Engine
Package: slmetric

Get metric distribution

Syntax

```
getMetricDistribution(metric_engine,MetricID)
```

Description

`getMetricDistribution(metric_engine,MetricID)` generates distribution for a specific metric, `MetricID`, for the metric data in the `slmetric.Engine` object, `metric_engine`. The distribution is on the metric data from the Value property of a `slmetric.metric.Result` object.

Input Arguments

`metric_engine` — Collects and accesses metric data
`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`MetricID` — Metric identifier
character vector

Metric identifier for a model metric, specified as a character vector.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

Output Arguments

dist — Distribution of the metric data
slmetric.metric.MetricDistribution object

Distribution of the metric data contains the following properties:

- MetricID is a char array that returns the metric ID specified in the getMetricDistribution function call.
- BinCounts is an uint64 array of the number of components corresponding to a bin.
- BinEdges is a double array of equally spaced edges of each bin.

Examples

Generate Metric Distribution

To generate the distribution for a specific metric, create a slmetric.Engine object, set the analysis root for the sldemo_fuelsys model, and create a histogram of the data.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics and get distribution
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);
dist = getMetricDistribution(metric_engine, metricID);

% View the distribution using a histogram to show the number of components correspondin
histogram('BinEdges',dist.BinEdges,'BinCounts',dist.BinCounts);
```

See Also

histcounts | slmetric.Engine | slmetric.metric.Result | slmetric.metric.ResultCollection | slmetric.metric.getAvailableMetrics

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2017a

getMetrics

Class: slmetric.Engine
Package: slmetric

Access model metric data

Syntax

```
Results = getMetrics(metric_engine)
Results = getMetrics(metric_engine,MetricIDs)
Results = getMetrics(metric_engine,MetricIDs,'AggregationDepth',ad)
```

Description

Access model metric data from the specified model metric engine. When you call `slmetric.Engine.execute`, the metric engine collects the metric data. The returned metric data is based on defined architectural components. The components are these Simulink objects:

- Model
- Model block
- Subsystem block
- Chart
- MATLAB Function block
- Protected model

`Results = getMetrics(metric_engine)` returns metric data for all metrics that the metric engine executed.

`Results = getMetrics(metric_engine,MetricIDs)` returns metric data for the specified metric identifiers.

`Results = getMetrics(metric_engine,MetricIDs,'AggregationDepth',ad)` returns metric data for the specified metric identifiers and specifying how to aggregate data.

Input Arguments

`metric_engine` — Collects and accesses metric data

`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for all available MathWorks metrics or for the specified `MetricIDs`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`MetricIDs` — Metric identifier

character vector | cell array of character vectors

Metric identifier for “Model Metrics” on page 2-309 or custom model metrics that you create. You can specify one or multiple metric identifiers. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

`AggregationDepth` — Depth or level in the component hierarchy to which `getMetrics` aggregates the metric data

`All` (default) | `None`

Depth or level in the component for which `getMetrics` aggregates the metric data, specified as a name-value pair argument. Values are one of the following:

- `All` — `getMetrics` aggregates the detailed results to the component level. Then, the component level results are used to calculate the aggregated values by traversing the component hierarchy. `getMetrics` returns only the component-level results.
- `None` — Do not aggregate measures and values. If you specify this option, `getMetrics` returns metric values as collected by the metric algorithm. For example, if the metric algorithm returns detailed results, the detailed results are returned without aggregation. `AggregatedValue` and `AggregatedMeasures` properties of the returned `slmetric.metric.Result` objects are empty.

Example: `'AggregationDepth','None'`

Data Types: `char`

Output Arguments

`Results` — Metric data from the metric engine
array of `slmetric.metric.Result` objects

Metric data from the metric engine.

Examples

Collect and Access Metric Data for a Model

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis, these properties are on by d
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
```

```
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

“Collect Model Metrics Programmatically”
“Model Metrics” on page 2-309

Introduced in R2016a

getResults

Class: Advisor.Application
Package: Advisor

Access Model Advisor analysis results

Syntax

```
Results = getResults(app)
Results = getResults(app,Name,Value)
```

Description

Access `Application` object analysis results.

`Results = getResults(app)` provides access to Model Advisor analysis results.

`Results = getResults(app,Name,Value)`

Input Arguments

`app` — Application
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`IDs` — Component IDs
cell array

Component IDs, as specified as a cell array of IDs

Data Types: `cell`

Output Arguments

`Result` — Analysis results
cell array of `ModelAdvisor.SystemResult` objects

Analysis results, returned as a cell array of `ModelAdvisor.SystemResult` objects.

See Also

`Advisor.Application.deselectCheckInstances` | `Advisor.Application.run` | `Advisor.Application.selectCheckInstances` | `Advisor.Application.setAnalysisRoot` | `Advisor.Manager.createApplication` | `ModelAdvisor.run`

Introduced in R2015b

getStatistics

Class: slmetric.Engine
Package: slmetric

Get statistics on metric data

Syntax

```
stats = getStatistics(metric_engine,MetricID)
```

Description

Generate statistics on the `Value` properties of the `slmetric.metric.Result` objects for the specified metric engine object, `metric_engine`.

`stats = getStatistics(metric_engine,MetricID)` generate statistics for the specified metric identifier.

Input Arguments

`metric_engine` — **Collects and accesses metric data**
`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`MetricID` — **Metric identifier**
character vector

Metric identifier for “Model Metrics” on page 2-309 or custom model metrics that you create. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

Output Arguments

stats — Metric statistics
slmetric.metric.Statistics object

The Statistics object contains the following properties:

- MinValue is a double that returns the minimum of the Value of the slmetric.metric.Result object.
- MaxValue is a double that returns the maximum of the Value of the slmetric.metric.Result object.
- MeanValue is a double that returns the mean of the Value of the slmetric.metric.Result object.
- StandardDeviation is a double that returns the standard deviation of the Value of the slmetric.metric.Result object.

Examples

Collect Statistics

This example shows how to create a slmetric.Engine object, set the analysis root, collect the block count metric, and collect statistics for the model sldemo_fuelsys.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root','sldemo_fuelsys');

% Generate and collect model metrics
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);
stats = getStatistics(metric_engine, metricID);
```

See Also

slmetric.metric.ResultCollection | slmetric.metric.getAvailableMetrics

Topics

“Collect Model Metrics Programmatically”
“Model Metrics” on page 2-309

Introduced in R2017a

loadConfiguration

Class: Advisor.Application
Package: Advisor

Load Model Advisor configuration

Syntax

`loadConfiguration(app,filename)`

Description

`loadConfiguration(app,filename)` loads a Model Advisor configuration MAT-file.

Input Arguments

`app` **— Application**
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

`filename` **— Name of Model Advisor configuration MAT-file**
character vector

Name of Model Advisor configuration MAT-file, specified as a character vector.

Example: `'MyConfiguration.mat'`

Data Types: `char`

See Also

`Advisor.Application.setAnalysisRoot` |
`Advisor.Manager.createApplication`

Introduced in R2015b

mdltransformer

Open Model Transformer

Syntax

```
mdltransformer(model)
```

Description

`mdltransformer(model)` opens the Model Transformer for a model specified by `model`. If the specified model is not open, this command opens it.

Examples

Open Model Transformer for model

Open the Model Transformer for `rtwdemo_reusable_sys_outputs` example model:

```
mdltransformer('rtwdemo_reusable_sys_outputs')
```

- “Transform Model to Variant System”

Input Arguments

`model` — Model name
character vector

Model name or handle, specified as a character vector.

Data Types: `char`

See Also

Topics

"Transform Model to Variant System"

Introduced in R2016b

metricsdashboard

Open Metrics Dashboard

Syntax

`metricsdashboard(system)`

Description

`metricsdashboard(system)` opens the Metrics Dashboard for a system specified by `system`. The *`system`* can be either a model name or a block path to a subsystem. The system cannot be a Configurable Subsystem block.

Examples

Open Metrics Dashboard for system

Open the Metrics Dashboard for `vdp` example model:

```
metricsdashboard('vdp')
```

Input Arguments

`system` — System name
character vector

System name, specified as a character vector.

Data Types: `char`

See Also

Topics

"Collect and Explore Metric Data by Using the Metrics Dashboard"

Introduced in R2017b

slmetric.metric.Metric class

Package: slmetric.metric

Abstract class for creating model metrics

Description

Abstract base class for creating model metrics. To create a model metric, create a MATLAB class that derives from the `slmetric.metric.Metric` class.

Properties

`CompileContext` — Compile mode
character vector

Compile mode for metric calculation. If your model metric requires model compilation, specify `PostCompile`. If your model metric does not require model compilation, specify `None`.

Example: `'PostCompile'`

Data Types: `char`

`ComponentScope` — Component scope
array of `Advisor.component.Types` enum values

Model components for which metric is calculated. The metric is calculated for all components that match the type.

`Description` — Metric description
character vector

Metric description.

Data Types: `char`

`ID` — Metric ID
character vector

Unique metric identifier.

Data Types: `char`

`Version` — Metric version number
integer

Use this property to communicate changes in your metric algorithm to the metric engine.

Data Types: `uint32`

`Name` — Name of the metric algorithm
character vector

Specify a name for the custom metric algorithm.

Data Types: `char`

`ResultChecksumCoverage` — Reuse metric data
logical

If `true`, results produced by the metric algorithm change only if the model or library source files change. If the source file and the metric `Version` have not changed, metric data is not regenerated. If `false`, each call to `slmetric.Engine.execute` collects new data for this metric and stores it in the metric repository.

Data Types: `logical`

`AggregationMode` — How the metric algorithm aggregates the metric data
character array

Specify the operation to aggregate the `slmetric.metric.Result` object properties `Value` and `Measure` across the component hierarchy. The metric algorithm outputs the aggregated values in the `slmetric.metric.Result` object properties `AggregatedValues` and `AggregatedMeasures`. Options are:

- `Sum`: Returns the sum of the `Value` property and the `Value` properties of all its children components across the component hierarchy.
- `Max`: Returns the maximum of the `Value` property and the `Value` properties of all its children components across the component hierarchy.
- `None`: No aggregation of metric values.

Data Types: `char`

`AggregateComponentDetails` — **Aggregate detailed results to the component level**
logical

If `true`, metric results that do not cover the full component scope are aggregated to the component level. Values and measures of the detailed results belonging to a component are summed and a new result is created that spans the complete component.

If `false`, returns the detailed results of the component. Detailed results are not aggregated.

Data Types: `logical`

`SupportsResultDetails` — **Specify whether `Details` property contains data**
logical

Specify whether the `slmetric.metric.Result` object property `Details` contains data. The default value is `false`.

Data Types: `logical`

Methods

algorithm	Specify logic for metric data analysis

See Also

`slmetric.Engine` | `slmetric.metric.Result` | `slmetric.metric.createNewMetricClass` | `slmetric.metric.getAvailableMetrics`

Topics

"Create a Custom Model Metric"
"Model Metrics" on page 2-309

Introduced in R2016a

algorithm

Class: slmetric.metric.Metric
Package: slmetric.metric

Specify logic for metric data analysis

Syntax

`Result = algorithm(Metric,Component)`

Description

Specify logic for metric algorithm analysis. Custom-authored metric algorithms are not called for library links and external MATLAB file components.

`Result = algorithm(Metric,Component)` specifies logic for metric algorithm analysis.

Input Arguments

`Metric` **— New model metric class**
`slmetric.metric.Metric` object

Model metric class you are defining for a new metric.

`Component` **— Component for metric analysis**
`Advisor.component.Component` object

Instance of `Advisor.component.Component` for metric analysis.

Output Arguments

`Result` — Algorithm result data
array of `slmetric.metric.Result` objects

Algorithm data, returned as an array of `slmetric.metric.Result` objects.

Examples

Create Metric Algorithm for Nonvirtual Block Count

This example shows how to use the `algorithm` method to create a nonvirtual block count metric.

Using the `createNewMetricClass` function, create a metric class with the name `nonvirtualblockcount`. The function creates the `nonvirtualblockcount.m` file in the current working folder.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Open and edit the metric algorithm file `nonvirtualblockcount.m`. The file contains an empty metric algorithm method.

```
edit(className);
```

Copy and paste the following code into the `nonvirtualblockcount.m` file. Save `nonvirtualblockcount.m`. The code provides a metric algorithm for counting the nonvirtual blocks.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    % nonvirtualblockcount calculate number of non-virtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as non-virtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end
```

```
methods
function this = nonvirtualblockcount()
    this.ID = 'nonvirtualblockcount';
    this.Version = 1;
    this.CompileContext = 'None';
    this.Description = 'Algorithm that counts nonvirtual blocks per level.';
    this.ComponentScope = [Advisor.component.Types.Model, ...
        Advisor.component.Types.SubSystem];
end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % use find_system to get all blocks inside this component
    blocks = find_system(getComponentSource(component), ...
        'FollowLinks','on', 'SearchDepth', 1, ...
        'Type', 'Block', ...
        'FollowLinks', 'On');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                            'on')
                        isNonVirtual(n) = false;
                    end
                case 'Outport'
                    % Outport: Virtual when the block resides within
                    % any SubSystem block (conditional or not), and
                    % does not reside in the root (top-level) Simulink window.
```

```
                    if component.Type ~= Advisor.component.Types.Model
                        isNonVirtual(n) = false;
                    end
                case 'Selector'
                    % Virtual only when Number of input dimensions
                    % specifies 1 and Index Option specifies Select
                    % all, Index vector (dialog), or Starting index (dialog).
                    nod = get_param(blocks{n}, 'NumberOfDimensions');
                    ios = get_param(blocks{n}, 'IndexOptionArray');

                    ios_settings = {'Assign all', 'Index vector (dialog)', ...
                        'Starting index (dialog)'};

                    if nod == 1 && any(strcmp(ios_settings, ios))
                        isNonVirtual(n) = false;
                    end
                case 'Trigger'
                    % Virtual when the output port is not present.
                    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                        isNonVirtual(n) = false;
                    end
                case 'Enable'
                    % Virtual unless connected directly to an Outport block.
                    isNonVirtual(n) = false;

                    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
                        pc = get_param(blocks{n}, 'PortConnectivity');

                        if ~isempty(pc.DstBlock) && ...
                                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                                'Outport')
                            isNonVirtual(n) = true;
                        end
                    end
            end
        end
    end

    blocks = blocks(isNonVirtual);

    res.Value = length(blocks);
end
```

```
    end
end
```

See Also
slmetric.metric.Result | slmetric.metric.createNewMetricClass

Topics
"Create a Custom Model Metric"
"Model Metrics" on page 2-309

Introduced in R2016a

slmetric.metric.ResultDetail class

Package: slmetric.metric

Details about instances of `slmetric.metric.Result` objects

Description

Details about what the metric engine counts for the `slmetric.metric.Result` object property `Value`.

Construction

Calling the `slmetric.Engine.execute` method creates the `slmetric.metric.Result` objects, which optionally includes the `slmetric.metric.ResultDetail` objects.

Properties

`ID` — Unique identifier
character vector

Unique identifier for the entity that the result detail instance counts. This property is read/write.

Data Types: `char`

`Name` — Name of model entity
character vector

Name of model entity that result detail instance counts. This property is read/write.

Data Types: `char`

`Value` — Value of `ID` property
double

Scalar value generated by metric algorithm for ID. This property is read/write.

Data Types: double

Methods

setGroup	Set the name and identifier for a group of slmetric.metric.ResultDetail objects
getGroupIdentifier	Obtain the identifier for a group of slmetric.metric.ResultDetail objects
getGroupName	Obtain the name for a group of slmetric.metric.ResultDetail objects

Examples

Obtain Clone Group Names and Identifiers

Use the getGroupName and getGroupIdentfier methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the slmetric.Engine.execute method. Apply the getMetrics method for themathworks.metric.CloneDetection metric.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root','ex_clone_detection','RootType','Model');
execute(metric_engine);
rc = getMetrics(metric_engine,'mathworks.metrics.CloneDetection');
```

For each slmetric.metric.Result object, display the ComponentPath. For each slmetric.metric.ResultDetail object, display the clone group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
```

```
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

The results show that the model contains one clone group, `CloneGroup1`, which contains two clones.

Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file, `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                  'blocks and groups them by the corresponding Data Store Memory block.';
```

```
        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');

            % Create a ResultDetail object for each data store read and write block.
            % Group ResultDetails by the data store name.
            details1 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dswBlocks)
                details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
                            get_param(dswBlocks{i}, 'Name'));
           groupID = get_param(dswBlocks{i},'DataStoreName');
           groupName = get_param(dswBlocks{i},'DataStoreName');
                details1(i).setGroup(groupID, groupName);
                details1(i).Value = 1;
            end

            details2 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dsrBlocks)
                details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
                   get_param(dsrBlocks{i}, 'Name'));
                groupID = get_param(dsrBlocks{i},'DataStoreName');
                groupName = get_param(dsrBlocks{i},'DataStoreName');
                details2(i).setGroup(groupID, groupName);
                details2(i).Value = 1;
            end

            res = slmetric.metric.Result();
            res.ComponentID = component.ID;
            res.MetricID = this.ID;
            res.Value = length(dswBlocks)+ length(dsrBlocks);
            res.Details = [details1 details2];
        end
    end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_mdlref_dsm` model.

Load the `sldemo_mdlref_dsm` model.

```
model = 'sldemo_mdlref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root',model,'RootType','Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdlref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond

No results for ComponentPath: sldemo_mdlref_dsm/A

No results for ComponentPath: sldemo_mdlref_dsm/A1

No results for ComponentPath: sldemo_mdlref_dsm/More Info1

ComponentPath: sldemo_mdlref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdlref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond

ComponentPath: sldemo_mdlref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot/NegativeSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;
bdclose('all');
```

See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` | `slmetric.metric.ResultDetail` | `slmetric.metric.getAvailableMetrics`

Introduced in R2017b

setGroup

Class: slmetric.metric.ResultDetail
Package: slmetric.metric

Set the name and identifier for a group of `slmetric.metric.ResultDetail` objects

Syntax

`setGroup(groupIdentifier,groupName)`

Description

For a custom-authored metric, set the identifier and name for a group of `slmetric.metric.ResultDetail` objects. Apply this method from within the part of the metric algorithm that specifies the details for `slmetric.Engine.getMetrics` objects.

`setGroup(groupIdentifier,groupName)` sets the values of the group name and identifier for an `slmetric.metric.ResultDetail` object.

Input Arguments

`groupIdentifier` — Group identifier
character vector

Specify a value for the identifier for a group of `slmetric.metric.ResultDetail` objects.

`groupName` — Group name
character vector

Specify a value for the name of a group of `slmetric.metric.ResultDetail` objects.

Examples

Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file.For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                  'blocks and groups them by the corresponding Data Store Memory block.';

        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');
```

```
                % Create a ResultDetail object for each data store read and write block.
                % Group ResultDetails by the data store name.
                details1 = slmetric.metric.ResultDetail.empty();
                for i=1:length(dswBlocks)
                    details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
                                get_param(dswBlocks{i}, 'Name'));
               groupID = get_param(dswBlocks{i},'DataStoreName');
               groupName = get_param(dswBlocks{i},'DataStoreName');
                    details1(i).setGroup(groupID, groupName);
                    details1(i).Value = 1;
                end

                details2 = slmetric.metric.ResultDetail.empty();
                for i=1:length(dsrBlocks)
                    details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
                       get_param(dsrBlocks{i}, 'Name'));
                    groupID = get_param(dsrBlocks{i},'DataStoreName');
                    groupName = get_param(dsrBlocks{i},'DataStoreName');
                    details2(i).setGroup(groupID, groupName);
                    details2(i).Value = 1;
                end

                res = slmetric.metric.Result();
                res.ComponentID = component.ID;
                res.MetricID = this.ID;
                res.Value = length(dswBlocks)+ length(dsrBlocks);
                res.Details = [details1 details2];
            end
        end
    end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_mdlref_dsm` model.

Load the `sldemo_mdlref_dsm` model.

```
model = 'sldemo_mdlref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root..

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root',model,'RootType','Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdlref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond

No results for ComponentPath: sldemo_mdlref_dsm/A

No results for ComponentPath: sldemo_mdlref_dsm/A1

No results for ComponentPath: sldemo_mdlref_dsm/More Info1

ComponentPath: sldemo_mdlref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond

ComponentPath: sldemo_mdlref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot/NegativeSS
```

```
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;
bdclose('all');
```

See Also

slmetric.metric.Result | slmetric.metric.ResultCollection | slmetric.metric.ResultDetail | slmetric.metric.getAvailableMetrics

Introduced in R2017b

getGroupIdentifier

Class: slmetric.metric.ResultDetail
Package: slmetric.metric

Obtain the identifier for a group of `slmetric.metric.ResultDetail` objects

Syntax

```
groupIdentifier = getGroupIdentifier(mrd)
```

Description

Obtain the identifier for a group of `slmetric.metric.ResultDetail` objects. Calling the `slmetric.Engine.execute` method collects metric data. Calling `slmetric.Engine.getMetrics` accesses the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects. Apply the `getGroupIdentifier` method to the `slmetric.metric.ResultDetail` object.

`groupIdentifier = getGroupIdentifier(mrd)` obtains the group identifier for the `slmetric.metric.ResultDetail` object `mrd`.

Input Arguments

`mrd` — `slmetric.metric.ResultDetail` object
character vector

Calling the `slmetric.Engine.execute` method creates the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects.

Output Arguments

`groupIdentifier` — Group identifier
character vector

Identifier for a group of `slmetric.metric.ResultDetail` objects.

Examples

Obtain Clone Group Names and Identifiers

Use the `getGroupName` and `getGroupIdentfier` methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the `slmetric.Engine.execute` method. Apply the `getMetrics` method for the`mathworks.metric.CloneDetection` metric.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root','ex_clone_detection','RootType','Model');
execute(metric_engine);
rc = getMetrics(metric_engine,'mathworks.metrics.CloneDetection');
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetail` object, display the clone group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

The results show that the model contains one clone group, `CloneGroup1`, which contains two clones.

Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a new metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file, `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                  'blocks and groups them by the corresponding Data Store Memory block.';

        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');
```

```
            % Create a ResultDetail object for each data store read and write block.
            % Group ResultDetails by the data store name.
            details1 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dswBlocks)
                details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
                            get_param(dswBlocks{i}, 'Name'));
           groupID = get_param(dswBlocks{i},'DataStoreName');
           groupName = get_param(dswBlocks{i},'DataStoreName');
                details1(i).setGroup(groupID, groupName);
                details1(i).Value = 1;
            end

            details2 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dsrBlocks)
                details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
                   get_param(dsrBlocks{i}, 'Name'));
                groupID = get_param(dsrBlocks{i},'DataStoreName');
                groupName = get_param(dsrBlocks{i},'DataStoreName');
                details2(i).setGroup(groupID, groupName);
                details2(i).Value = 1;
            end

            res = slmetric.metric.Result();
            res.ComponentID = component.ID;
            res.MetricID = this.ID;
            res.Value = length(dswBlocks)+ length(dsrBlocks);
            res.Details = [details1 details2];
        end
    end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the`sldemo_mdlref_dsm` model.

Load the `sldemo_mdlref_dsm` model.

```
model = 'sldemo_mdlref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root..

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root',model,'RootType','Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdlref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond

No results for ComponentPath: sldemo_mdlref_dsm/A

No results for ComponentPath: sldemo_mdlref_dsm/A1

No results for ComponentPath: sldemo_mdlref_dsm/More Info1

ComponentPath: sldemo_mdlref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond

ComponentPath: sldemo_mdlref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot/NegativeSS
```

```
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;
bdclose('all');
```

See Also

slmetric.metric.Result | slmetric.metric.ResultCollection | slmetric.metric.ResultDetail | slmetric.metric.getAvailableMetrics

Introduced in R2017b

getGroupName

Class: slmetric.metric.ResultDetail
Package: slmetric.metric

Obtain the name for a group of `slmetric.metric.ResultDetail` objects

Syntax

`groupName = getGroupName(mrd)`

Description

Obtain the name of a group of `slmetric.metric.ResultDetail` objects. Calling the `slmetric.Engine.execute` method collects metric data. Calling `slmetric.Engine.getMetrics` accesses the `slmetric.metric.Result` objects which include the `slmetric.metric.ResultDetail` objects. Apply the `getGroupName` method to the `slmetric.metric.ResultDetail` object.

`groupName = getGroupName(mrd)` obtains the name for the `slmetric.metric.ResultDetail` object `mrd`.

Input Arguments

`mrd` — `slmetric.metric.ResultDetail` object
character vector

Calling the `slmetric.Engine.execute` method creates the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects.

Output Arguments

groupName — Group name
character vector

Name for a group of slmetric.metric.ResultDetail objects

Examples

Obtain Clone Group Names and Identifiers

Use the getGroupName and getGroupIdentifier methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the slmetric.Engine.execute method. Apply the getMetrics method for the mathworks.metric.CloneDetection metrics.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root','ex_clone_detection','RootType','Model');
execute(metric_engine);
rc = getMetrics(metric_engine,'mathworks.metrics.CloneDetection');
```

For each slmetric.metric.Result object, display the ComponentPath. For each slmetric.metric.ResultDetail object, display the clone group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

The results show that the model contains one clone group, CloneGroup1, which contains two clones.

Set Group Names and Group Identifiers for a Custom Model Metric

Use the setGroup method to group detailed results. When you create a custom model metric, you apply this method as part of the algorithm method.

Using the createNewMetricClass function, create a metric class named DataStoreCount. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The createNewMetricClass function creates a file, DataStoreCount.m, in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the DataStoreCount.m file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the DataStoreCount.m file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                  'blocks and groups them by the corresponding Data Store Memory block.';

        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');
```

```
            % Create a ResultDetail object for each data store read and write block.
            % Group ResultDetails by the data store name.
            details1 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dswBlocks)
                details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
                            get_param(dswBlocks{i}, 'Name'));
           groupID = get_param(dswBlocks{i},'DataStoreName');
           groupName = get_param(dswBlocks{i},'DataStoreName');
                details1(i).setGroup(groupID, groupName);
                details1(i).Value = 1;
            end

            details2 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dsrBlocks)
                details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
                   get_param(dsrBlocks{i}, 'Name'));
                groupID = get_param(dsrBlocks{i},'DataStoreName');
                groupName = get_param(dsrBlocks{i},'DataStoreName');
                details2(i).setGroup(groupID, groupName);
                details2(i).Value = 1;
            end

            res = slmetric.metric.Result();
            res.ComponentID = component.ID;
            res.MetricID = this.ID;
            res.Value = length(dswBlocks)+ length(dsrBlocks);
            res.Details = [details1 details2];
        end
    end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the`sldemo_mdlref_dsm` model.

Load the `sldemo_mdlref_dsm` model.

```
model = 'sldemo_mdlref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine,'Root',model,'RootType','Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
    for m=1:length(rc.Results(n).Details)
      disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
          disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
          disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
    disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdlref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond

No results for ComponentPath: sldemo_mdlref_dsm/A

No results for ComponentPath: sldemo_mdlref_dsm/A1

No results for ComponentPath: sldemo_mdlref_dsm/More Info1

ComponentPath: sldemo_mdlref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond

ComponentPath: sldemo_mdlref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal

ComponentPath: sldemo_mdlref_dsm_bot/NegativeSS
```

```
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;
bdclose('all');
```

See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` | `slmetric.metric.ResultDetail` | `slmetric.metric.getAvailableMetrics`

Introduced in R2017b

Advisor.component.Component class

Package: Advisor.component

Create component for metric analysis

Description

Model component used for metric analysis. When you define a custom model metric, the component object defines the component for metric analysis.

Construction

`component_obj = Advisor.component.Component` creates a model component object.

Properties

`ID` — Component ID

character vector

Component identifier. This property is read/write.

`Type` — Component type

enum

Component type, as specified by `Advisor.component.Types`. This property is read/write.

`Name` — Component name

character vector

Model component name. This property is read/write.

`IsLinked` — Specifies if the component is linked to a library

logical

`IsLinked` is `true` if the component is linked to a library. Components of type `Model`, `ModelBlock`, `ProtectedModel` cannot be linked. For these properties, the `IsLinked` is always `true`.

Methods

getPath	Retrieve component path

See Also

`Advisor.component.Types` | `slmetric.metric.Metric`

Topics

"Create a Custom Model Metric"
"Model Metrics" on page 2-309

Introduced in R2016a

getPath

Class: Advisor.component.Component
Package: Advisor.component

Retrieve component path

Syntax

`path = getPath(component)`

Description

`path = getPath(component)` retrieves the path to the component.

Input Arguments

`component` — Component
`Advisor.component.Component` model object

Constructed `Advisor.component.Component` model object.

Output Arguments

`path` — Model component path
character vector

Model component path, specified as a character vector.

See Also

`Advisor.component.Types`

Introduced in R2016a

Advisor.component.Types class

Package: Advisor.component

Create `enum` class specifying component type

Description

Create an enumeration `Advisor.component.Types` class to specify the model component type.

Construction

`enum_comp_type = Advisor.component.Type.Model` creates an enumeration of component type `Model`. The following table lists the component types.

Type	Description
`Model`	Simulink block diagram.
`LibraryBlock`	Library linked block.
`MFile`	MATLAB code file.
`ProtectedModel`	Protect Simulink block diagram.
`SubSystem`	Simulink subsystem block.
`ModelBlock`	Simulink model block.
`Chart`	Stateflow® chart or Stateflow block.
`MATLABFunction`	MATLAB function block.

See Also

`Advisor.component.Component` | `slmetric.metric.Metric`

Topics

"Create a Custom Model Metric"

"Model Metrics" on page 2-309

Introduced in R2016a

ModelAdvisor.Action class

Package: ModelAdvisor

Add actions to custom checks

Description

Instances of this class define actions you take when the Model Advisor checks do not pass. Users access actions by clicking the **Action** button that you define in the Model Advisor window.

Construction

ModelAdvisor.Action	Add actions to custom checks

Methods

setCallbackFcn	Specify action callback function

Properties

Description	Message in **Action** box
Name	Action button label

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
myAction.Name='Fix block fonts';
myAction.Description=...
   'Click the button to update all blocks with specified font';
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.Action

Class: ModelAdvisor.Action
Package: ModelAdvisor

Add actions to custom checks

Syntax

```
action_obj = ModelAdvisor.Action
```

Description

`action_obj = ModelAdvisor.Action` creates a handle to an action object.

Note

- Include an action definition in a check definition.
- Each check can contain only one action.

Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.Check class

Package: ModelAdvisor

Create custom checks

Description

The `ModelAdvisor.Check` class creates a Model Advisor check object. Checks must have an associated `ModelAdvisor.Task` object to be displayed in the Model Advisor tree.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product** > **Simulink** folder and in the **By Task** > **Model Referencing** folder in the Model Advisor tree.

When you use checks in task definitions, the following rules apply:

- If you define the properties of the check in the check definition and the task definition, the task definition takes precedence. The Model Advisor displays the information contained in the task definition. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property and in the check definition using the `ModelAdvisor.Check.Title` property, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.
- If you define the properties of the check in the check definition but not the task definition, the task uses the properties from the check. For example, if you define the name of the check in the check definition using the `ModelAdvisor.Check.Title` property, and you register the check using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Check.Title`.
- If you define the properties of the check in the task definition but not the check definition, the Model Advisor displays the information as long as you register the task with the Model Advisor instead of the check. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property instead of the `ModelAdvisor.Check.Title` property, and you register the

check using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.

Construction

ModelAdvisor.Check	Create custom checks

Methods

getID	Return check identifier
setAction	Specify action for check
setCallbackFcn	Specify callback function for check
setInputParameters	Specify input parameters for check
setInputParametersLayoutGrid	Specify layout grid for input parameters

Properties

CallbackContext	Specify when to run check
CallbackHandle	Callback function handle for check
CallbackStyle	Callback function type
EmitInputParametersToReport	Display check input parameters in the Model Advisor report
Enable	Indicate whether user can enable or disable check
ID	Identifier for check
LicenseName	Product license names required to display and run check
ListViewVisible	Status of **Explore Result** button
Result	Results cell array
supportExclusion	Set to support exclusions
SupportLibrary	Set to support library models
Title	Name of check
TitleTips	Description of check
Value	Status of check
Visible	Indicate to display or hide check

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
```

See Also

“Model Advisor Customization”

Topics

"Create Model Advisor Checks"

ModelAdvisor.Check

Class: ModelAdvisor.Check
Package: ModelAdvisor

Create custom checks

Syntax

```
check_obj = ModelAdvisor.Check(check_ID)
```

Description

`check_obj = ModelAdvisor.Check(check_ID)` creates a check object, `check_obj`, and assigns it a unique identifier, `check_ID`. `check_ID` must remain constant. To display checks in the Model Advisor tree, checks must have an associated `ModelAdvisor.Task` or `ModelAdvisor.Root` object.

Note You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product** > **Simulink folder** and in the **By Task** > **Model Referencing** folder in the Model Advisor tree.

Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.FactoryGroup class

Package: ModelAdvisor

Define subfolder in **By Task** folder

Description

The `ModelAdvisor.FactoryGroup` class defines a new subfolder to add to the **By Task** folder.

Construction

ModelAdvisor.FactoryGroup	Define subfolder in **By Task** folder

Methods

addCheck	Add check to folder

Properties

Description	Description of folder
DisplayName	Name of folder
ID	Identifier for folder
MAObj	Model Advisor object

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

ModelAdvisor.FactoryGroup

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Define subfolder in **By Task** folder

Syntax

`fg_obj = ModelAdvisor.FactoryGroup(fg_ID)`

Description

`fg_obj = ModelAdvisor.FactoryGroup(fg_ID)` creates a handle to a factory group object, `fg_obj`, and assigns it a unique identifier, `fg_ID`. `fg_ID` must remain constant.

Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.FormatTemplate class

Package: ModelAdvisor

Template for formatting Model Advisor analysis results

Description

Use the `ModelAdvisor.FormatTemplate` class to format the result of a check in the analysis result pane of the Model Advisor for a uniform look and feel among the checks you create. There are two formats for the analysis result:

- Table
- List

Construction

ModelAdvisor.FormatTemplate	Construct template object for formatting Model Advisor analysis results

Methods

addRow	Add row to table
setCheckText	Add description of check to result
setColTitles	Add column titles to table
setInformation	Add description of subcheck to result
setListObj	Add list of hyperlinks to model objects
setRecAction	Add Recommended Action section and text
setRefLink	Add See Also section and links
setSubBar	Add line between subcheck results
setSubResultStatus	Add status to check or subcheck result
setSubResultStatusText	Add text below status in result
setSubTitle	Add title for subcheck in result
setTableInfo	Add data to table
setTableTitle	Add title to table

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

The following code creates two template objects, `ft1` and `ft2`, and uses them to format the result of running the check in a table and a list. The result identifies the blocks in the model. The graphics following the code display the output as it appears in the Model Advisor when the check passes and fails.

```
function sl_customization(cm)

% register custom checks
cm.addModelAdvisorCheckFcn(@defineModelAdvisorChecks);

% register custom factory group
cm.addModelAdvisorTaskFcn(@defineModelAdvisorTasks);
```

```
% -----------------------------
% defines Model Advisor Checks
% -----------------------------
function defineModelAdvisorChecks

% Define and register a sample check
rec = ModelAdvisor.Check('mathworks.example.SampleStyleOne');
rec.Title = 'Sample check for Model Advisor using the ModelAdvisor.FormatTemplate';
setCallbackFcn(rec, @SampleStyleOneCallback,'None','StyleOne');

mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);


% -----------------------------
% defines Model Advisor Tasks
% -----------------------------
function defineModelAdvisorTasks
mdladvRoot = ModelAdvisor.Root;

% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='My Group 1';
rec.Description='Demo Factory Group';
rec.addCheck('mathworks.example.SampleStyleOne');
mdladvRoot.publish(rec); % publish inside By Group list

% -----------------------------
% Sample Check With Subchecks Callback Function
% -----------------------------
function ResultDescription = SampleStyleOneCallback(system)
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system); % get object

% Initialize variables
ResultDescription={};
ResultStatus = false; % Default check status is 'Warning'
mdladvObj.setCheckResultStatus(ResultStatus);

% Create FormatTemplate object for first subcheck, specify table format
ft1 = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information describing the overall check
setCheckText(ft1, ['Find and report all blocks in the model. '...
    '(setCheckText method - Description of what the check reviews)']);

% Add information describing the subcheck
setSubTitle(ft1, 'Table of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft1, ['Find and report all blocks in a table. '...
    '(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
setRefLink(ft1, {{'Standard 1 reference (setRefLink method)'},
    {'Standard 2 reference (setRefLink method)'}});
```

```
% Add information to the table
setTableTitle(ft1, {'Blocks in the Model (setTableTitle method)'});
setColTitles(ft1, {'Index (setColTitles method)',
    'Block Name (setColTitles method)'});

% Perform the check actions
allBlocks = find_system(system);
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft1, 'Warn');
    setSubResultStatusText(ft1, ['The model does not contain blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft1, {'Add blocks to the model. '...
        '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
else
    % Add status for subcheck
    setSubResultStatus(ft1, 'Pass');
    setSubResultStatusText(ft1, ['The model contains blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    for inx = 2 : length(allBlocks)
        % Add information to the table
        addRow(ft1, {inx-1,allBlocks(inx)});
    end
    ResultStatus = true;
end

% Pass table template object for subcheck to Model Advisor
ResultDescription{end+1} = ft1;

% Create FormatTemplate object for second subcheck, specify list format
ft2 = ModelAdvisor.FormatTemplate('ListTemplate');

% Add information describing the subcheck
setSubTitle(ft2, 'List of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft2, ['Find and report all blocks in a list. '...
    '(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
setRefLink(ft2, {{'Standard 1 reference (setRefLink method)'},
    {'Standard 2 reference (setRefLink method)'}});

% Last subcheck, supress line
setSubBar(ft2, false);

% Perform the subcheck actions
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft2, 'Warn');
    setSubResultStatusText(ft2, ['The model does not contain blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft2, {'Add blocks to the model. '...
        '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
```

```
else
    % Add status for subcheck
    setSubResultStatus(ft2, 'Pass');
    setSubResultStatusText(ft2, ['The model contains blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    % Add information to the list
    setListObj(ft2, allBlocks);
end

% Pass list template object for the subcheck to Model Advisor
ResultDescription{end+1} = ft2;
% Set overall check status
mdladvObj.setCheckResultStatus(ResultStatus);
```

The following graphic displays the output as it appears in the Model Advisor when the check passes.

Result: Passed

Table of Blocks (setSubTitle method - Title of the subcheck)
Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

Passed
The model contains blocks. (setSubResultStatusText method - Description of result status)

Blocks in the Model (setTableTitle method)

Index (setColTitles method)	Block Name (setColTitles method)
1	model/Constant
2	model/Constant1
3	model/Gain
4	model/Product
5	model/Out1

List of Blocks (setSubTitle method - Title of the subcheck)
Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

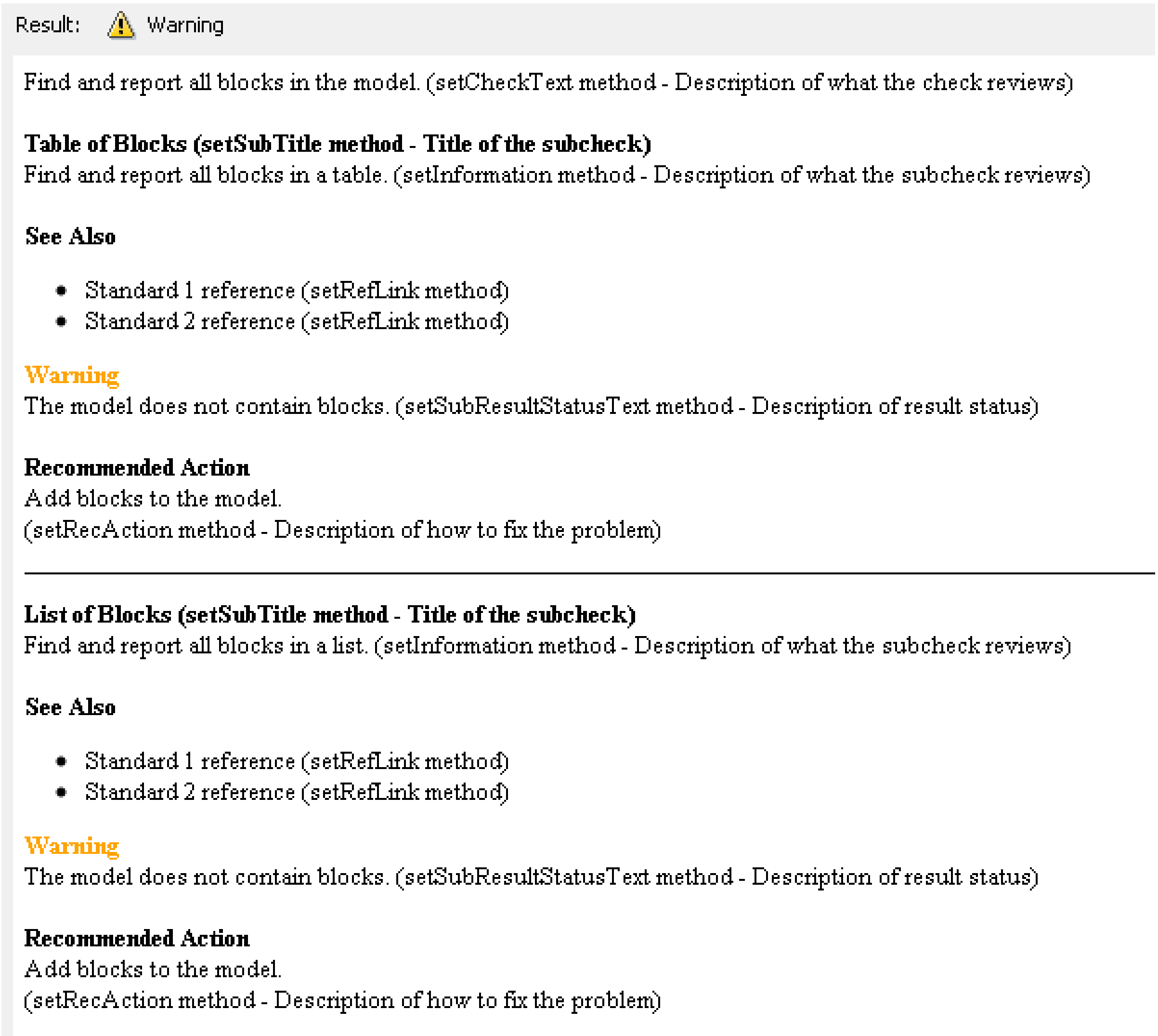
See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

Passed
The model contains blocks. (setSubResultStatusText method - Description of result status)

- model
- model/Constant
- model/Constant1
- model/Gain
- model/Product
- model/Out1

The following graphic displays the output as it appears in the Model Advisor when the check fails.



Result: Warning

Find and report all blocks in the model. (setCheckText method - Description of what the check reviews)

Table of Blocks (setSubTitle method - Title of the subcheck)
Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

Warning
The model does not contain blocks. (setSubResultStatusText method - Description of result status)

Recommended Action
Add blocks to the model.
(setRecAction method - Description of how to fix the problem)

List of Blocks (setSubTitle method - Title of the subcheck)
Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

Warning
The model does not contain blocks. (setSubResultStatusText method - Description of result status)

Recommended Action
Add blocks to the model.
(setRecAction method - Description of how to fix the problem)

Alternatives

Use the Model Advisor Formatting API to format check analysis results. However, use the `ModelAdvisor.FormatTemplate` class for a uniform look and feel among the checks you create.

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

ModelAdvisor.FormatTemplate

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Construct template object for formatting Model Advisor analysis results

Syntax

`obj = ModelAdvisor.FormatTemplate('type')`

Description

`obj = ModelAdvisor.FormatTemplate('type')` creates a handle, *obj*, to an object of the `ModelAdvisor.FormatTemplate` class. *type* is a character vector identifying the format type of the template, either list or table. Valid values are `ListTemplate` and `TableTemplate`.

You must return the result object to the Model Advisor to display the formatted result in the analysis result pane.

Note Use the `ModelAdvisor.FormatTemplate` class in check callbacks.

Examples

Create a template object, `ft`, and use it to create a list template:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”
“Format Check Results”

ModelAdvisor.Group class

Package: ModelAdvisor

Define custom folder

Description

The `ModelAdvisor.Group` class defines a folder that is displayed in the Model Advisor tree. Use folders to consolidate checks by functionality or usage.

Construction

ModelAdvisor.Group	Define custom folder

Methods

addGroup	Add subfolder to folder
addProcedure	Add procedure to folder
addTask	Add task to folder

Properties

Description	Description of folder
DisplayName	Name of folder
ID	Identifier for folder
MAObj	Model Advisor object

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

ModelAdvisor.Group

Class: ModelAdvisor.Group
Package: ModelAdvisor

Define custom folder

Syntax

```
group_obj = ModelAdvisor.Group(group_ID)
```

Description

`group_obj = ModelAdvisor.Group(group_ID)` creates a handle to a group object, `group_obj`, and assigns it a unique identifier, `group_ID`. `group_ID` must remain constant.

Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.Image class

Package: ModelAdvisor

Include image in Model Advisor output

Description

The `ModelAdvisor.Image` class adds an image to the Model Advisor output.

Construction

ModelAdvisor.Image	Include image in Model Advisor output

Methods

setHyperlink	Specify hyperlink location
setImageSource	Specify image location

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.Image

Class: ModelAdvisor.Image
Package: ModelAdvisor

Include image in Model Advisor output

Syntax

```
object = ModelAdvisor.Image
```

Description

`object = ModelAdvisor.Image` creates a handle to an image object, `object`, that the Model Advisor displays in the output. The Model Advisor supports many image formats, including, but not limited to, JPEG, BMP, and GIF.

Examples

```
image_obj = ModelAdvisor.Image;
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.InputParameter class

Package: ModelAdvisor

Add input parameters to custom checks

Description

Instances of the `ModelAdvisor.InputParameter` class specify the input parameters a custom check uses in analyzing the model. Access input parameters in the Model Advisor window.

Construction

ModelAdvisor.InputParameter	Add input parameters to custom checks

Methods

setColSpan	Specify number of columns for input parameter
setRowSpan	Specify rows for input parameter

Properties

Description	Description of input parameter
Entries	Drop-down list entries
Name	Input parameter name
Type	Input parameter type
Value	Value of input parameter

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.InputParameter

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Add input parameters to custom checks

Syntax

`input_param = ModelAdvisor.InputParameter`

Description

`input_param = ModelAdvisor.InputParameter` creates a handle to an input parameter object, `input_param`.

Note You must include input parameter definitions in a check definition.

Examples

Note The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvnvdemo_mdladv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.setInputParametersLayoutGrid([3 2]);
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
inputParam1.Description = 'sample tooltip';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
inputParam2 = ModelAdvisor.InputParameter;
```

```
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
inputParam3 = ModelAdvisor.InputParameter;
inputParam3.Name='Valid font';
inputParam3.Type='Combobox';
inputParam3.Description='sample tooltip';
inputParam3.Entries={'Arial', 'Arial Black'};
inputParam3.setRowSpan([2 2]);
inputParam3.setColSpan([2 2]);
rec.setInputParameters({inputParam1,inputParam2,inputParam3});
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.LineBreak class

Package: ModelAdvisor

Insert line break

Description

Use instances of the `ModelAdvisor.LineBreak` class to insert line breaks in the Model Advisor outputs.

Construction

ModelAdvisor.LineBreak	Insert line break

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.LineBreak

Class: ModelAdvisor.LineBreak
Package: ModelAdvisor

Insert line break

Syntax

`ModelAdvisor.LineBreak`

Description

`ModelAdvisor.LineBreak` inserts a line break into the Model Advisor output.

Examples

Add a line break between two lines of text:

```
result = ModelAdvisor.Paragraph;
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.List class

Package: ModelAdvisor

Create list class

Description

Use instances of the `ModelAdvisor.List` class to create list-formatted outputs.

Construction

ModelAdvisor.List	Create list class

Methods

addItem	Add item to list
setType	Specify list type

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.List

Class: ModelAdvisor.List
Package: ModelAdvisor

Create list class

Syntax

`list = ModelAdvisor.List`

Description

`list = ModelAdvisor.List` creates a list object, `list`.

Examples

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.ListViewParameter class

Package: ModelAdvisor

Add list view parameters to custom checks

Description

The Model Advisor uses list view parameters to populate the Model Advisor Result Explorer. Access the information in list views by clicking **Explore Result** in the Model Advisor window.

Construction

ModelAdvisor.ListViewParameter	Add list view parameters to custom checks

Properties

Attributes	Attributes to display in Model Advisor Report Explorer
Data	Objects in Model Advisor Result Explorer
Name	Drop-down list entry

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

Note The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvnvdemo_mdladv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

```
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
mdladvObj.setCheckResultStatus(true);

% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object')';
myLVParam.Attributes = {'FontName'}; % name is default property
mdladvObj.setListViewParameters({myLVParam});
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.ListViewParameter

Class: ModelAdvisor.ListViewParameter
Package: ModelAdvisor

Add list view parameters to custom checks

Syntax

```
lv_param = ModelAdvisor.ListViewParameter
```

Description

`lv_param = ModelAdvisor.ListViewParameter` defines a list view, `lv_param`.

Note Include list view parameter definitions in a check definition.

See Also

"Model Advisor Customization"

Topics

"Define Model Advisor Result Explorer Views"
"Create Model Advisor Checks"
"Batch-Fix Warnings or Failures" (Simulink)
"Customization Example"
"getListViewParameters" (Simulink)
"setListViewParameters" (Simulink)

ModelAdvisor.lookupCheckID

Package: ModelAdvisor

Look up Model Advisor check ID

Syntax

```
NewID = ModelAdvisor.lookupCheckID('OldCheckID')
```

Description

`NewID = ModelAdvisor.lookupCheckID('OldCheckID')` returns the check ID of the check specified by `OldCheckID`. `OldCheckID` is the ID of a check prior to R2010b.

Input Arguments

`OldCheckID`

`OldCheckID` is the ID of a check prior to R2010b.

Output Arguments

`NewID`

Check ID that corresponds to the previous check ID identified by `OldCheckID`.

Examples

Look up the check ID for **By Product** > **Simulink Check** > **Modeling Standards** > **DO-178C/DO-331 Checks** > **Check safety-related optimization settings** using the previous ID `DO178B:OptionSet`:

```
NewID = ModelAdvisor.lookupCheckID('DO178B:OptionSet');
```

Alternatives

"Archive and View Results"

See Also

`ModelAdvisor.run`

Topics

"Archive and View Results"

Introduced in R2010b

ModelAdvisor.Paragraph class

Package: ModelAdvisor

Create and format paragraph

Description

The `ModelAdvisor.Paragraph` class creates and formats a paragraph object.

Construction

ModelAdvisor.Paragraph	Create and format paragraph

Methods

addItem	Add item to paragraph
setAlign	Specify paragraph alignment

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
% Check Simulation optimization setting
ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check Simulation '...
'optimization settings:']);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

ModelAdvisor.Paragraph

Class: ModelAdvisor.Paragraph
Package: ModelAdvisor

Create and format paragraph

Syntax

`para_obj = ModelAdvisor.Paragraph`

Description

`para_obj = ModelAdvisor.Paragraph` defines a paragraph object `para_obj`.

Examples

```
% Check Simulation optimization setting
ResultDescription{end+1} = ModelAdvisor.Paragraph(['Check Simulation '...
'optimization settings:']);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.Procedure class

Package: ModelAdvisor

Define custom procedures

Description

The `ModelAdvisor.Procedure` class defines a procedure that is displayed in the Model Advisor tree. Use procedures to organize additional procedures or checks by functionality or usage.

Construction

ModelAdvisor.Procedure	Define custom procedures

Properties

`Description`

Provides information about the procedure. Details about the procedure are displayed in the right pane of the Model Advisor.

Default: `''` (empty character vector)

`Name`

Specifies the name of the procedure that is displayed in the Model Advisor.

Default: `''` (empty character vector)

`ID`

Specifies a permanent, unique identifier for the procedure.

Note

- You must specify this field.
- The value of `ID` must remain constant.
- The Model Advisor generates an error if `ID` is not unique.
- Procedure definitions must refer to other procedures by `ID`.

`MAObj`

Specifies a handle to the current Model Advisor object.

Methods

addProcedure	Add subprocedure to procedure
addTask	Add task to procedure

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Procedures"
"Create Procedural-Based Configurations"
"Create Model Advisor Checks"

ModelAdvisor.Procedure

Class: ModelAdvisor.Procedure
Package: ModelAdvisor

Define custom procedures

Syntax

```
procedure_obj = ModelAdvisor.Procedure(procedure_ID)
```

Description

`procedure_obj = ModelAdvisor.Procedure(procedure_ID)` creates a handle to a procedure object, `procedure_obj`, and assigns it a unique identifier, `procedure_ID`. `procedure_ID` must remain constant.

Examples

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');
```

See Also

"Model Advisor Customization"

Topics

"Create Procedures"
"Create Procedural-Based Configurations"
"Create Model Advisor Checks"

ModelAdvisor.Root class

Package: ModelAdvisor

Identify root node

Description

The `ModelAdvisor.Root` class returns the root object.

Construction

ModelAdvisor.Root	Identify root node

Methods

publish	Publish object in Model Advisor root
register	Register object in Model Advisor root

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

ModelAdvisor.Root

Class: ModelAdvisor.Root
Package: ModelAdvisor

Identify root node

Syntax

```
root_obj = ModelAdvisor.Root
```

Description

`root_obj = ModelAdvisor.Root` creates a handle to the root object, `root_obj`.

Examples

```
mdladvRoot = ModelAdvisor.Root;
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

ModelAdvisor.run

Package: ModelAdvisor

Run Model Advisor checks on systems

Syntax

```
SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList,Name,Value)
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',
FileName,Name,Value)
```

Description

`SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList,Name,Value)` runs the Model Advisor on the systems provided by `SysList` with additional options specified by one or more optional `Name,Value` pair arguments. `CheckIDList` contains cell array of check IDs to run.

`SysResultObjArray = ModelAdvisor.run(SysList,'Configuration', FileName,Name,Value)` runs the Model Advisor on the systems provided by `SysList`. The list of checks to run is specified using a Model Advisor configuration file, specified by `FileName`.

Input Arguments

`SysList`

Cell array of systems to run.

`CheckIDList`

Cell array of check IDs to run. For details on how to find check IDs, see “Find Check IDs”.

CheckIDList optionally can include input parameters for specific checks using the following syntax; {'CheckID','InputParam',{'IP','IPV'}}, where IP is the input parameter name and IPV is the corresponding input parameter value. You can specify several input parameter name and value pair arguments in any order as IP1,IPV1, …,IPN,IPVN.

FileName

Name of the Model Advisor configuration file. For details on creating a configuration file, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

DisplayResults

Setting DisplayResults to 'Summary' displays a summary of the system results in the Command Window. Setting DisplayResults to 'Details' displays the following in the Command Window:

- Which system the Model Advisor is checking while the run is in progress.
- For each system, the pass and fail results of each check.
- A summary of the system results.

Setting DisplayResults to 'None' displays no information in the Command Window.

Default: 'Summary'

Force

Setting Force to 'On' removes existing modeladvisor/*system* folders. Setting Force to 'Off' prompts you before removing existing modeladvisor/*system* folders.

Default: 'Off'

ParallelMode

Setting `ParallelMode` to `'On'` runs the Model Advisor in parallel mode if you have a Parallel Computing Toolbox license and a multicore machine. The Parallel Computing Toolbox does not support 32-bit Windows® machines. Each parallel process runs checks on one model at a time. In parallel mode, load the model data from the model workspace or data dictionary. The Model Advisor in parallel mode does not support model data in the base workspace. For an example, see “Create a Function for Checking Multiple Systems in Parallel”.

Default: `'Off'`

TempDir

Setting `TempDir` to `'On'` runs the Model Advisor from a temporary working folder, to avoid concurrency issues when running using a parallel pool. For more information, see “Resolving Data Concurrency Issues” (Simulink). Setting `TempDir` to `'Off'` runs the Model Advisor in the current working folder.

Default: `'Off'`

ShowExclusions

Setting `ShowExclusions` to `'On'` lists Model Advisor check exclusions in the report. Setting `ShowExclusions` to `'Off'` does not list Model Advisor check exclusion in the report.

Default: `'On'`

Output Arguments

SysResultObjArray

Cell array of `ModelAdvisor.SystemResult` objects, one for each model specified in `SysList`. Each `ModelAdvisor.SystemResult` object contains an array of `CheckResultObj` objects. Save `SysResultObjArray` to review results at a later time without having to rerun the Model Advisor (see “Save and Load Process for Objects” (MATLAB)).

CheckResultObj

Array of `ModelAdvisor.CheckResult` objects, one for each check that runs.

Examples

Runs the Model Advisor checks **Check model diagnostic parameters** and **Check for fully defined interface** on the `sldemo_auto_climatecontrol/Heater Control` and `sldemo_auto_climatecontrol/AC Control` subsystems:

```
% Create list of checks and models to run.
CheckIDList ={'mathworks.maab.jc_0021',...
    'mathworks.iec61508.RootLevelInports'};
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList);
```

Runs the Model Advisor configuration file `slvnvdemo_mdladv_config.mat` on the `sldemo_auto_climatecontrol/Heater Control` and `sldemo_auto_climatecontrol/AC Control` subsystems:

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);
```

Tips

- If you have a Parallel Computing Toolbox™ license and a multicore machine, Model Advisor can run on multiple systems in parallel. You can run the Model Advisor in parallel mode by using `ModelAdvisor.run` with `'ParallelMode'` set to `'On'`. By default, `'ParallelMode'` is set to `'Off'`. When you use `ModelAdvisor.run` with `'ParallelMode'` set to `'On'`, MATLAB automatically creates a parallel pool.

Alternatives

- Use the Model Advisor GUI to run each system, one at a time.
- Create a script or function using the `Simulink.ModelAdvisor` class to run each system, one at a time.

See Also

`ModelAdvisor.lookupCheckID` | `ModelAdvisor.summaryReport` | `view` | `viewReport`

Topics

"Checking Systems Programmatically"
"Check Multiple Systems in Parallel"
"Create a Function for Checking Multiple Systems in Parallel"
"Automate Model Advisor Check Execution"
"Find Check IDs"
"Organize Checks and Folders Using the Model Advisor Configuration Editor"
"Save and Load Process for Objects" (MATLAB)

Introduced in R2010b

ModelAdvisor.summaryReport

Package: ModelAdvisor

Open Model Advisor Command-Line Summary report

Syntax

`ModelAdvisor.summaryReport(SysResultObjArray)`

Description

`ModelAdvisor.summaryReport(SysResultObjArray)` opens the Model Advisor Command-Line Summary report in a web browser. `SysResultObjArray` is a cell array of `ModelAdvisor.SystemResult` objects returned by `ModelAdvisor.run`.

Input Arguments

`SysResultObjArray`

Cell array of `ModelAdvisor.SystemResult` objects returned by `ModelAdvisor.run`.

Examples

Opens the Model Advisor Command-Line Summary report after running the Model Advisor:

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);
```

```
% Open the Model Advisor Command-Line Summary report.
ModelAdvisor.summaryReport(SysResultObjArray)
```

Alternatives

"View Results in Model Advisor Command-Line Summary Report"

See Also

`ModelAdvisor.run` | `view` | `viewReport`

Topics

"Checking Systems Programmatically"
"Check Multiple Systems in Parallel"
"Create a Function for Checking Multiple Systems in Parallel"
"Automate Model Advisor Check Execution"
"Archive and View Model Advisor Run Results"

Introduced in R2010b

ModelAdvisor.Table class

Package: ModelAdvisor

Create table

Description

Instances of the `ModelAdvisor.Table` class create and format a table. Specify the number of rows and columns in a table, excluding the table title and table heading row.

Construction

ModelAdvisor.Table	Create table

Methods

getEntry	Get table cell contents
setColHeading	Specify table column title
setColHeadingAlign	Specify column title alignment
setColHeadingValign	Specify column title vertical alignment
setColWidth	Specify column widths
setEntries	Set contents of table
setEntry	Add cell to table
setEntryAlign	Specify table cell alignment
setEntryValign	Specify table cell vertical alignment
setHeading	Specify table title
setHeadingAlign	Specify table title alignment
setRowHeading	Specify table row title
setRowHeadingAlign	Specify table row title alignment
setRowHeadingValign	Specify table row title vertical alignment

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.Table

Class: ModelAdvisor.Table
Package: ModelAdvisor

Create table

Syntax

```
table = ModelAdvisor.Table(row, column)
```

Description

`table = ModelAdvisor.Table(row, column)` creates a table object (`table`). The Model Advisor displays the table object containing the number of rows (`row`) and columns (`column`) that you specify.

Examples

Create two table objects

Create two table objects, `table1` and `table2`. The Model Advisor displays `table1` in the results as a table with one row and one column. The Model Advisor display `table2` in the results as a table with two rows and three columns.

```
table1 = ModelAdvisor.Table(1,1);
table2 = ModelAdvisor.Table(2,3);
```

Create table with five rows and five columns

Create a table with five rows and five columns containing randomly generated numbers.

Use the following MATLAB code in a callback function. The Model Advisor displays `table1` in the results.

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	81472.3686	9754.0405	15761.3082	14188.6339	65574.0699
Row 2	90579.1937	27849.8219	97059.2782	42176.1283	3571.1679
Row 3	12698.6816	54688.1519	Example Text	91573.5525	84912.9306
Row 4	91337.5856	95750.6835	48537.5649	79220.733	93399.3248
Row 5	63235.9246	96488.8535	80028.0469	95949.2426	67873.5155

See Also

`ModelAdvisor.Table.setColHeading` | `ModelAdvisor.Table.setColHeadingAlign` | `ModelAdvisor.Table.setColWidth` | `ModelAdvisor.Table.setEntry` | `ModelAdvisor.Table.setEntryAlign` | `ModelAdvisor.Table.setRowHeading` | `ModelAdvisor.Text`

Topics

"Create Callback Functions and Results"

ModelAdvisor.Task class

Package: ModelAdvisor

Define custom tasks

Description

The `ModelAdvisor.Task` class is a wrapper for a check so that you can access the check with the Model Advisor.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product** > **Simulink** folder and in the **By Task** > **Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

Construction

ModelAdvisor.Task	Define custom tasks

Methods

setCheck	Specify check used in task

Properties

Description	Description of task
DisplayName	Name of task
Enable	Indicate if user can enable and disable task
ID	Identifier for task
LicenseName	Product license names required to display and run task
MAObj	Model Advisor object
Value	Status of task
Visible	Indicate to display or hide task

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

ModelAdvisor.Task

Class: ModelAdvisor.Task
Package: ModelAdvisor

Define custom tasks

Syntax

```
task_obj = ModelAdvisor.Task(task_ID)
```

Description

`task_obj = ModelAdvisor.Task(task_ID)` creates a task object, `task_obj`, with a unique identifier, `task_ID`. `task_ID` must remain constant. If you do not specify `task_ID`, the Model Advisor assigns a random `task_ID` to the task object.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product** > **Simulink folder** and in the **By Task** > **Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

Examples

In the following example, you create three task objects, `MAT1`, `MAT2`, and `MAT3`.

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

See Also
“Model Advisor Customization”

Topics
“Create Model Advisor Checks”

ModelAdvisor.Text class

Package: ModelAdvisor

Create Model Advisor text output

Description

Instances of `ModelAdvisor.Text` class create formatted text for the Model Advisor output.

Construction

ModelAdvisor.Text	Create Model Advisor text output

Methods

setBold	Specify bold text
setColor	Specify text color
setHyperlink	Specify hyperlinked text
setItalic	Italicize text
setRetainSpaceReturn	Retain spacing and returns in text
setSubscript	Specify subscripted text
setSuperscript	Specify superscripted text
setUnderlined	Underline text

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

ModelAdvisor.Text

Class: ModelAdvisor.Text
Package: ModelAdvisor

Create Model Advisor text output

Syntax

`text = ModelAdvisor.Text(`*`content`*`, {`*`attribute`*`})`

Description

`text = ModelAdvisor.Text(`*`content`*`, {`*`attribute`*`})` creates a text object for the Model Advisor output.

Input Arguments

`content`	Optional character vector specifying the content of the text object. If *`content`* is empty, empty text is output.

attribute	Optional cell array of character vectors specifying the formatting of the content. If no attribute is specified, the output text has default coloring with no formatting. Possible formatting options include: • `'normal'` (default) — Text is default color and style. • `'bold'` — Text is bold. • `'italic'` — Text is italicized. • `'underline'` — Text is underlined. • `'pass'` — Text is green. • `'warn'` — Text is yellow. • `'fail'` — Text is red. • `'keyword'` — Text is blue. • `'subscript'` — Text is subscripted. • `'superscript'` — Text is superscripted.

Output Arguments

`text`	The text object you create

Examples

```
text = ModelAdvisor.Text('Sub entry 1', {'pass','bold'})
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

publish

Class: ModelAdvisor.Root
Package: ModelAdvisor

Publish object in Model Advisor root

Syntax

```
publish(root_obj, check_obj, location)
publish(root_obj, group_obj)
publish(root_obj, procedure_obj)
publish(root_obj, fg_obj)
```

Description

`publish(root_obj, check_obj, location)` specifies where the Model Advisor places the check in the Model Advisor tree. `location` is either one of the subfolders in the **By Product** folder, or the name of a new subfolder to put in the **By Product** folder. Use a pipe-delimited character vector to indicate multiple subfolders. For example, to add a check to the **Simulink Check** > **Modeling Standards** folder, use the following: `'Simulink Check|Modeling Standards'`.

If the **By Product** is not displayed in the Model Advisor window, select **Show By Product Folder** from the **Settings** > **Preferences** dialog box.

`publish(root_obj, group_obj)` specifies the `ModelAdvisor.Group` object to publish as a folder in the **Model Advisor Task Manager** folder.

`publish(root_obj, procedure_obj)` specifies the `ModelAdvisor.Procedure` object to publish.

`publish(root_obj, fg_obj)` specifies the `ModelAdvisor.FactoryGroup` object to publish as a subfolder in the **By Task** folder.

Examples

```
% publish check into By Product > Demo group.
mdladvRoot.publish(rec, 'Demo');
```

See Also

Topics

"Define Where Custom Checks Appear"
"Define Where Tasks Appear"
"Define Where Custom Folders Appear"

refresh_customizations

Class: Advisor.Manager
Package: Advisor

Refresh Model Advisor check information cache

Syntax

```
Advisor.Manager.refresh_customizations()
```

Description

`Advisor.Manager.refresh_customizations()` refreshes the Model Advisor check information cache.

Alternatives

To refresh the cache from Model Advisor, select **Settings** > **Preferences**. Click **Update check information cache**, then click **OK**. To see updates, close and reopen model, then start Model Advisor.

See Also

Topics

"Create and Add Custom Checks - Basic Examples"
"Create Check for Model Configuration Parameters"

Introduced in R2016b

register

Class: ModelAdvisor.Root
Package: ModelAdvisor

Register object in Model Advisor root

Syntax

`register(MAobj, obj)`

Description

`register(MAobj, obj)` registers the object, *`obj`*, in the root object `MAobj`.

In the Model Advisor memory, the `register` method registers the following types of objects:

- `ModelAdvisor.Check`
- `ModelAdvisor.FactoryGroup`
- `ModelAdvisor.Group`
- `ModelAdvisor.Procedure`
- `ModelAdvisor.Task`

The `register` method places objects in the Model Advisor memory that you use in other functions. The `register` method does not place objects in the Model Advisor tree.

Examples

```
mdladvRoot = ModelAdvisor.Root;

MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.DisplayName='Example task with input parameter and auto-fix ability';
MAT1.setCheck('com.mathworks.sample.Check1');
mdladvRoot.register(MAT1);
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT2.DisplayName='Example task 2';
MAT2.setCheck('com.mathworks.sample.Check2');
mdladvRoot.register(MAT2);

MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
MAT3.DisplayName='Example task 3';
MAT3.setCheck('com.mathworks.sample.Check3');
mdladvRoot.register(MAT3)
```

run

Class: Advisor.Application
Package: Advisor

Run Model Advisor analysis on model components

Syntax

`run(app)`

Description

`run(app)` runs a Model Advisor analysis, as specified by the `Application` object.

Examples

This example shows how to create an `Application` object, set root analysis to `RootModel`, and run a Model Advisor analysis.

```
% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Run Model Advisor analysis
run(app);
```

Input Arguments

app — Application

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

See Also

`Advisor.Application.setAnalysisRoot` | `Advisor.Manager.createApplication`

Introduced in R2015b

selectCheckInstances

Class: Advisor.Application
Package: Advisor

Select check instances to use in Model Advisor analysis

Syntax

```
selectCheckInstances(app)
selectCheckInstances(app,Name,Value)
```

Description

You can select check instances to use in a Model Advisor analysis. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. To obtain the check instance ID, use the `getCheckInstanceIDs` method.

`selectCheckInstances(app)` selects all check instances to use for Model Advisor analysis.

`selectCheckInstances(app,Name,Value)` selects check instances specified by `Name,Value` pair arguments to use for Model Advisor analysis.

Input Arguments

app — Application
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`IDs` — Check instance IDs
cell array

Select check instances to use in Model Advisor analysis, as specified as a cell array of IDs

Data Types: `cell`

Examples

Select All Check Instances to Use in Model Advisor Analysis

This example shows how to set the root model, create an `Application` object, set root analysis, and select all check instances for Model Advisor analysis.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select all checks
selectCheckInstances(app);
```

Select Check Instance for Model Advisor Analysis Using Instance ID

This example shows how to set the root model, create an `Application` object, set root analysis, and select a check using instance ID.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();
```

```
% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select "Identify unconnected lines, input ports, and output
% ports" check using check instance ID
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
selectCheckInstances(app,'IDs',checkinstanceID);
```

See Also

`Advisor.Application.deselectCheckInstances` |
`Advisor.Application.getCheckInstanceIDs` |
`Advisor.Application.setAnalysisRoot` |
`Advisor.Manager.createApplication`

Introduced in R2015b

selectComponents

Class: Advisor.Application
Package: Advisor

Select model components for Model Advisor analysis

Syntax

```
selectComponents(app)
selectComponents(app,Name,Value)
```

Description

You can select model components for Model Advisor analysis. A model component is a model in the system hierarchy. Models that the root model references and that `Advisor.Application.setAnalysisRoot` specifies are model components. By default, all components are selected.

`selectComponents(app)` includes all components for Model Advisor analysis.

`selectComponents(app,Name,Value)` includes model components specified by `Name,Value` pair arguments for Model Advisor analysis.

Input Arguments

app — Application
`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`IDs` — Component IDs
cell array

Components to select for Model Advisor analysis, as specified by a cell array of IDs

Data Types: `cell`

`HierarchicalSelection` — Select component and component children
false (default) | true

Select components specified by IDs and component children from Model Advisor analysis.

Data Types: `logical`

Examples

Include All Components in Model Advisor Analysis

This example shows how to set the root model, create an `Application` object, set root analysis, and include model components in Model Advisor analysis.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select all components
selectComponents(app);
```

Select Components for Model Advisor Analysis Using IDs

This example shows how to set the root model, create an `Application` object, set root analysis, and include model components using IDs.

```
% Set root model to  sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select component using IDs
selectComponents(app,'IDs',RootModel);
```

See Also

Advisor.Application.deselectComponents | Advisor.Application.setAnalysisRoot | Advisor.Manager.createApplication

Introduced in R2015b

setAction

Class: ModelAdvisor.Check
Package: ModelAdvisor

Specify action for check

Syntax

```
setAction(check_obj, action_obj)
```

Description

`setAction(check_obj, action_obj)` returns the action object `action.obj` to use in the check `check_obj`. The `setAction` method identifies the action you want to use in a check.

See Also

"Model Advisor Customization" | `ModelAdvisor.Action`

Topics

"Create Model Advisor Checks"

setAlign

Class: ModelAdvisor.Paragraph
Package: ModelAdvisor

Specify paragraph alignment

Syntax

`setAlign(paragraph, alignment)`

Description

`setAlign(paragraph, alignment)` specifies the alignment of text. Possible values are:

- `'left'` (default)
- `'right'`
- `'center'`

Examples

```
report_paragraph = ModelAdvisor.Paragraph;
setAlign(report_paragraph, 'center');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setAnalysisRoot

Class: Advisor.Application
Package: Advisor

Specify model hierarchy for Model Advisor analysis

Syntax

```
setAnalysisRoot(app,'Root',root)
setAnalysisRoot(app,'Root',root,Name,Value)
```

Description

Specify the model hierarchy for an `Application` object analysis.

`setAnalysisRoot(app,'Root',root)` specifies the analysis root.

`setAnalysisRoot(app,'Root',root,Name,Value)` specifies the analysis root using `Name,Value` options.

Input Arguments

`app` — Application

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

`'Root',root` — `Name,Value` argument specifying model or subsystem path

character vector

Comma-separated `Name,Value` argument specifying model or subsystem path

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`RootType` — Analysis root

`Model` (default) | `Subsystem`

Examples

Specify Root Model as Analysis Root

This example shows how to set the root model, create an `Application` object, and set the root analysis.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);
```

Specify Subsystem as Analysis Root

This example shows how to set the root model, create an `Application` object, and specify a subsystem as the analysis root.

```
% Set root model to sldemo_mdlref_basic model
RootModel='sldemo_mdlref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();
```

```
% Set the Application object root analysis
setAnalysisRoot(app,'Root','sldemo_mdlref_basic/CounterA','RootType','Subsystem');
```

See Also

`Advisor.Manager.createApplication`

Introduced in R2015b

setAnalysisRoot

Class: slmetric.Engine
Package: slmetric

Specify model or subsystem for metric analysis

Syntax

```
setAnalysisRoot(metric_engine,'Root',root)
setAnalysisRoot(metric_engine,'Root',root,Name,Value)
```

Description

Specify the model or subsystem for `slmetric.Engine` metric object analysis.

`setAnalysisRoot(metric_engine,'Root',root)` specifies the metric analysis root.

`setAnalysisRoot(metric_engine,'Root',root,Name,Value)` specifies the metric analysis root by using `Name,Value` pairs.

Input Arguments

`metric_engine` — Collects and accesses metric data
`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for all MathWorks metrics or for the specified `MetricIDs`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`'Root'` — `Name,Value` argument specifying model or subsystem path
character vector

Comma-separated `Name,Value` argument specifying model or subsystem path.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`RootType` — Type of model component for metric analysis
`Model` (default) | `Subsystem`

Examples

Specify Model for Metric Analysis

This example shows how to set the root model, create an `slmetric.Engine` object, and specify the model for metric analysis.

```
% Set root model to vdp model
RootModel='vdp';

% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine,'Root',RootModel);
```

Specify Subsystem for Metric Analysis

This example shows how to set the root model, create an `slmetric.Engine` object, and specify a subsystem for metric analysis.

```
% Set subsystem to CounterA
Subsys ='sf_car/Engine';

% Create an slmetric.Engine object
metric_engine = slmetric.Engine();
```

```
% Set a subsystem for metric analysis
setAnalysisRoot(metric_engine,'Root',Subsys,'RootType','Subsystem');
```

See Also

`slmetric.metric.Metric` | `slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2016a

setBold

Class: ModelAdvisor.Text
Package: ModelAdvisor

Specify bold text

Syntax

`setBold(text, *mode*)`

Description

`setBold(text, mode)` specifies whether `text` should be formatted in bold font.

Input Arguments

`text`	Instantiation of the ModelAdvisor.Text class
`mode`	A Boolean value indicating bold formatting of text: • `true` — Format the text in bold font. • `false` — Do not format the text in bold font.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
setBold(t1, 'true');
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”

setCallbackFcn

Class: ModelAdvisor.Action
Package: ModelAdvisor

Specify action callback function

Syntax

`setCallbackFcn(action_obj, @handle)`

Description

`setCallbackFcn(action_obj, @handle)` specifies the handle to the callback function, `handle`, to use with the action object, `action_obj`.

Examples

Note The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvnvdemo_mdladv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

```
rec = ModelAdvisor.Check('mathworks.example.optimizationSettings');
% Define an automatic fix action for this check
modifyAction = ModelAdvisor.Action;
modifyAction.setCallbackFcn(@modifyOptmizationSetting);
modifyAction.Name = 'Modify Settings';
modifyAction.Description = ['Modify model configuration optimization' ...
                            ' settings that can impact safety'];
modifyAction.Enable = true;
rec.setAction(modifyAction);
```

See Also

"Model Advisor Customization"

Topics

“Define Check Actions”
“Create Model Advisor Checks”
“setActionEnable” (Simulink)

setCallbackFcn

Class: ModelAdvisor.Check
Package: ModelAdvisor

Specify callback function for check

Syntax

`setCallbackFcn(check_obj, @handle, context, style)`

Description

`setCallbackFcn(check_obj, @handle, context, style)` specifies the callback function to use with the check, `check_obj`.

Input Arguments

`check_obj` — Instantiation of the `ModelAdvisor.Check` class

`handle` — Handle to a check callback function

`context` — Context for checking the model or subsystem:

- `'None'` — No special requirements.
- `'PostCompile'` — The model must be compiled.

`style` — Type of callback function:

- `'StyleOne'` — Simple check callback function, for formatting results using template
- `'StyleTwo'` — Detailed check callback function
- `'StyleThree'` — Check callback functions with hyperlinked results

Examples

```
% --- sample check 1
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
rec.setCallbackFcn(@SampleStyleThreeCallback,'None','StyleThree');
```

See Also

"Model Advisor Customization"

Topics

"Create Callback Functions and Results"
"Create Model Advisor Checks"

setCheck

Class: ModelAdvisor.Task
Package: ModelAdvisor

Specify check used in task

Syntax

```
setCheck(task, check_ID)
```

Description

`setCheck(task, check_ID)` specifies the check to use in the task.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** appears in the **By Product** > **Simulink folder** and in the **By Task** > **Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

Input Arguments

`task`	Instantiation of the `ModelAdvisor.Task` class
`check_ID`	A unique identifier for the check to use in the task

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
setCheck(MAT1, 'com.mathworks.sample.Check1');
```

setCheckText

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add description of check to result

Syntax

`setCheckText(ft_obj, text)`

Description

`setCheckText(ft_obj, text)` is an optional method that adds text or a model advisor template object as the first item in the report. Use this method to add information describing the overall check.

Input Arguments

`ft_obj`

A handle to a template object.

`text`

A character vector or a handle to a formatting object.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

`text` appears as the first line in the analysis result.

Examples

Create a list object, `ft`, and add a line of text to the result:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setCheckText(ft, ['Identify unconnected lines, input ports,'...
    'and output ports in the model']);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

setColHeading

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table column title

Syntax

```
setColHeading(table, column, heading)
```

Description

`setColHeading(table, column, heading)` specifies that the column header of `column` is set to `heading`.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`column`	An integer specifying the column number
`heading`	A character vector, element object, or object array specifying the table column title

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setColHeading(table1, 1, 'Header 1');
setColHeading(table1, 2, 'Header 2');
setColHeading(table1, 3, 'Header 3');
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”

setColHeadingAlign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify column title alignment

Syntax

`setColHeadingAlign(table, column, alignment)`

Description

`setColHeadingAlign(table, column, alignment)` specifies the alignment of the column heading.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`column`	An integer specifying the column number
`alignment`	Alignment of the column heading. *`alignment`* can have one of the following values: • `left` (default) • `right` • `center`

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setColHeading(table1, 1, 'Header 1');
setColHeadingAlign(table1, 1, 'center');
setColHeading(table1, 2, 'Header 2');
```

```
setColHeadingAlign(table1, 2, 'center');
setColHeading(table1, 3, 'Header 3');
setColHeadingAlign(table1, 3, 'center');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setColHeadingValign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify column title vertical alignment

Syntax

`setColHeadingValign(table, column, alignment)`

Description

`setColHeadingValign(table, column, alignment)` specifies the vertical alignment of the column heading.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`column`	An integer specifying the column number
`alignment`	Vertical alignment of the column heading. *`alignment`* can have one of the following values: • `top` (default) • `middle` • `bottom`

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setColHeading(table1, 1, 'Header 1');
setColHeadingValign(table1, 1, 'middle');
setColHeading(table1, 2, 'Header 2');
```

```
setColHeadingValign(table1, 2, 'middle');
setColHeading(table1, 3, 'Header 3');
setColHeadingValign(table1, 3, 'middle');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setColor

Class: ModelAdvisor.Text
Package: ModelAdvisor

Specify text color

Syntax

`setColor(text, *color*)`

Description

`setColor(text, *color*)` sets the `text` color to *`color`*.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`color`	Color of the text, specified as one of the following formatting options: • `'normal'` (default) — Text is default color. • `'pass'` — Text is green. • `'warn'` — Text is yellow. • `'fail'` — Text is red. • `'keyword'` — Text is blue.

Examples

```
t1 = ModelAdvisor.Text('This is a warning');
setColor(t1, 'warn');
```

setColSpan

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Specify number of columns for input parameter

Syntax

```
setColSpan(input_param, [start_col end_col])
```

Description

`setColSpan(input_param, [start_col end_col])` specifies the number of columns that the parameter occupies. Use the `setColSpan` method to specify where you want an input parameter located in the layout grid when there are multiple input parameters.

Input Arguments

`input_param`	Instantiation of the `ModelAdvisor.InputParameter` class
`start_col`	A positive integer representing the first column that the input parameter occupies in the layout grid
`end_col`	A positive integer representing the last column that the input parameter occupies in the layout grid

Examples

```
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
```

```
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
```

setColTitles

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add column titles to table

Syntax

```
setColTitles(ft_obj, {col_title_1, col_title_2, ...})
```

Description

`setColTitles(`*`ft_obj`*`, {`*`col_title_1`*`, `*`col_title_2`*`, ...})` is method you must use when you create a template object that is a table type. Use it to specify the titles of the columns in the table.

Note Before adding data to a table, you must specify column titles.

Input Arguments

`ft_obj`

A handle to a template object.

`col_title_N`

A cell of character vectors or handles to formatting objects, specifying the column titles.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The order of the *`col_title_N`* inputs determines which column the title is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

Examples

Create a table object, `ft`, and specify two column titles:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');
setColTitles(ft, {'Index', 'Block Name'});
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

setColWidth

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify column widths

Syntax

```
setColWidth(table, column, width)
```

Description

`setColWidth(table, column, width)` specifies the `column`.

The `setColWidth` method specifies the table column widths relative to the entire table width. If column widths are [1 2 3], the second column is twice the width of the first column, and the third column is three times the width of the first column. Unspecified columns have a default width of 1. For example:

```
setColWidth(1, 1);
setColWidth(3, 2);
```

specifies [1 1 2] column widths.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`column`	An integer specifying column number
`width`	An integer or array of integers specifying the column widths, relative to the entire table width

Examples

```
table1 = ModelAdvisor.Table(2, 3)
setColWidth(table1, 1, 1);
setColWidth(table1, 3, 2);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setEntries

Class: ModelAdvisor.Table
Package: ModelAdvisor

Set contents of table

Syntax

`setEntries(content)`

Description

`setEntries(content)` sets `content` of the table.

Input Arguments

`content`	A 2–D cell array containing the contents of the table. Each item of the cell array must be either a character vector or an instance of ModelAdvisor.Element. The size of the cell array must be equal to the size of the table specified in the ModelAdvisor.Table constructor.

Examples

```
table = ModelAdvisor.Table(4,3);
contents = cell(4,3); % 4 by 3 table
for k=1:4
    for m=1:3
      contents{k,m} = ['Contents for row-' num2str(k) ' column-' num2str(m)];
    end
end
table.setEntries(contents);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setEntry

Class: ModelAdvisor.Table
Package: ModelAdvisor

Add cell to table

Syntax

```
setEntry(table, row, column, string)
setEntry(table, row, column, content)
```

Description

`setEntry(table, row, column, string)` adds a character vector to a cell in a table.

`setEntry(table, row, column, content)` adds an object specified by `content` to a cell in a table.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying the row
`column`	An integer specifying the column
`string`	A character vector representing the contents of the entry
`content`	An element object or object array specifying the content of the table entries

Examples

Create two tables and insert `table2` into the first cell of `table1`:

```
table1 = ModelAdvisor.Table(1, 1);
table2 = ModelAdvisor.Table(2, 3);
.
.
.
setEntry(table1, 1, 1, table2);
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setEntryAlign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table cell alignment

Syntax

`setEntryAlign(table, row, column, *alignment*)`

Description

`setEntryAlign(table, row, column, *alignment*)` specifies the cell alignment of the designated cell.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying row number
`column`	An integer specifying column number
`alignment`	Cell alignment, specified as one of the following: • `'left'` (default) • `'right'` • `'center'`

Examples

```
table1 = ModelAdvisor.Table(2,3);
setHeading(table1, 'New Table');
.
.
```

```
.
setEntry(table1, 1, 1, 'First Entry');
setEntryAlign(table1, 1, 1, 'center');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setEntryValign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table cell vertical alignment

Syntax

`setEntryValign(table, row, column, alignment)`

Description

`setEntryValign(table, row, column, alignment)` specifies the cell alignment of the designated cell.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying row number
`column`	An integer specifying column number
`alignment`	Cell vertical alignment, specified as one of the following: • `'top'` (default) • `'middle'` • `'bottom'`

Examples

```
table1 = ModelAdvisor.Table(2,3);
setHeading(table1, 'New Table');
.
.
```

```
.
setEntry(table1, 1, 1, 'First Entry');
setEntryValign(table1, 1, 1, 'middle');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setHeading

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table title

Syntax

```
setHeading(table, title)
```

Description

`setHeading(table, title)` specifies the table title.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`title`	A character vector, element object, or object array that specifies the table title

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setHeading(table1, 'New Table');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setHeadingAlign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table title alignment

Syntax

`setHeadingAlign(table, alignment)`

Description

`setHeadingAlign(table, alignment)` specifies the alignment for the table title.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`alignment`	Table title alignment, specified as one of the following: • `'left'` (default) • `'right'` • `'center'`

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setHeading(table1, 'New Table');
setHeadingAlign(table1, 'center');
```

See Also

"Model Advisor Customization"

Topics
"Create Model Advisor Checks"

setHyperlink

Class: ModelAdvisor.Image
Package: ModelAdvisor

Specify hyperlink location

Syntax

```
setHyperlink(image, url)
```

Description

`setHyperlink(image, url)` specifies the target location of the hyperlink associated with `image`.

Input Arguments

`image`	Instantiation of the `ModelAdvisor.Image` class
`url`	The target URL

Examples

```
matlab_logo=ModelAdvisor.Image;
setHyperlink(matlab_logo, 'http://www.mathworks.com');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setHyperlink

Class: ModelAdvisor.Text
Package: ModelAdvisor

Specify hyperlinked text

Syntax

```
setHyperlink(text, url)
```

Description

`setHyperlink(text, url)` creates a hyperlink from the text to the specified URL.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`url`	The target location of the URL

Examples

```
t1 = ModelAdvisor.Text('MathWorks home page');
setHyperlink(t1, 'http://www.mathworks.com');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setImageSource

Class: ModelAdvisor.Image
Package: ModelAdvisor

Specify image location

Syntax

`setImageSource(image_obj, source)`

Description

`setImageSource(image_obj, source)` specifies the location of the image.

Input Arguments

`image_obj`	Instantiation of the `ModelAdvisor.Image` class
source	The location of the image file

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setInformation

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add description of subcheck to result

Syntax

`setInformation(ft_obj, text)`

Description

`setInformation(ft_obj, text)` is an optional method that adds text as the first item after the subcheck title. Use this method to add information describing the subcheck.

Input Arguments

`ft_obj`

A handle to a template object.

`text`

A character vector or a handle to a formatting object, that describes the subcheck.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The Model Advisor displays *`text`* after the title of the subcheck.

Examples

Create a list object, `ft`, and specify a subcheck title and description:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubTitle(ft, ['Check for constructs in the model '...
    'that are not supported when generating code']);
setInformation(ft, ['Identify blocks that should not '...
    'be used for code generation.']);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

setInputParameters

Class: ModelAdvisor.Check
Package: ModelAdvisor

Specify input parameters for check

Syntax

```
setInputParameters(check_obj, params)
```

Description

`setInputParameters(check_obj, params)` specifies `ModelAdvisor.InputParameter` objects (`params`) to be used as input parameters to a check (`check_obj`).

Input Arguments

`check_obj`	Instantiation of the `ModelAdvisor.Check` class
`params`	A cell array of `ModelAdvisor.InputParameters` objects

Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
inputParam1 = ModelAdvisor.InputParameter;
inputParam2 = ModelAdvisor.InputParameter;
inputParam3 = ModelAdvisor.InputParameter;
setInputParameters(rec, {inputParam1,inputParam2,inputParam3});
```

See Also

"Model Advisor Customization" | `ModelAdvisor.InputParameter`

Topics

“Create Model Advisor Checks”

setInputParametersLayoutGrid

Class: ModelAdvisor.Check
Package: ModelAdvisor

Specify layout grid for input parameters

Syntax

`setInputParametersLayoutGrid(check_obj, [row col])`

Description

`setInputParametersLayoutGrid(check_obj, [row col])` specifies the layout grid for input parameters in the Model Advisor. Use the `setInputParametersLayoutGrid` method when there are multiple input parameters.

Input Arguments

`check_obj`	Instantiation of the `ModelAdvisor.Check` class
`row`	Number of rows in the layout grid
`col`	Number of columns in the layout grid

Examples

```
% --- sample check 1
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
rec.setCallbackFcn(@SampleStyleThreeCallback,'None','StyleThree');
rec.setInputParametersLayoutGrid([3 2]);
```

See Also

"Model Advisor Customization" | `ModelAdvisor.InputParameter`

Topics

"Create Model Advisor Checks"

setItalic

Class: ModelAdvisor.Text
Package: ModelAdvisor

Italicize text

Syntax

`setItalic(text, mode)`

Description

`setItalic(text, mode)` specifies whether `text` should be italicized.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`mode`	A Boolean value indicating italic formatting of text: • `true` — Italicize the text. • `false` — Do not italicize the text.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
setItalic(t1, 'true');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setListObj

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add list of hyperlinks to model objects

Syntax

`setListObj(ft_obj, {model_obj})`

Description

`setListObj(ft_obj, {model_obj})` is an optional method that generates a bulleted list of hyperlinks to model objects. *`ft_obj`* is a handle to a list template object. *`model_obj`* is a cell array of handles or full paths to blocks, or model objects that the Model Advisor displays as a bulleted list of hyperlinks in the report.

Examples

Create a list object, `ft`, and add a list of the blocks found in the model:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');

% Find all the blocks in the system
allBlocks = find_system(system);

% Add the blocks to a list
setListObj(ft, allBlocks);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

“Format Check Results”

setRecAction

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add Recommended Action section and text

Syntax

`setRecAction(ft_obj, {text})`

Description

`setRecAction(ft_obj, {text})` is an optional method that adds a Recommended Action section to the report. Use this method to describe how to fix the check.

Input Arguments

`ft_obj`

A handle to a template object.

`text`

A cell array of character vectors or handles to formatting objects, that describes the recommended action to fix the issues reported by the check.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The Model Advisor displays the recommended action as a separate section below the list or table in the report.

Examples

Create a list object, `ft`, find Gain blocks in the model, and recommend changing them:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
% Find all Gain blocks
gainBlocks = find_system(gcs, 'BlockType','Gain');

% Find Gain blocks
for idx = 1:length(gainBlocks)
    gainObj = get_param(gainBlocks(idx), 'Object');

    setRecAction(ft, {'If you are using these blocks '...
    'as buffers, you should replace them with '...
    'Signal Conversion blocks'});
end
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

setRefLink

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add See Also section and links

Syntax

```
setRefLink(ft_obj, {{'standard'}})
setRefLink(ft_obj, {{'url', 'standard'}})
```

Description

`setRefLink(ft_obj, {{'standard'}})` is an optional method that adds a See Also section above the table or list in the result. Use this method to add references to standards. *`ft_obj`* is a handle to a template object. *`standard`* is a cell array of character vectors that you want to display in the result. If you include more than one cell, the Model Advisor displays the character vectors in a bulleted list.

`setRefLink(ft_obj, {{'url', 'standard'}})` generates a list of links in the See Also section. *`url`* indicates the location to link to. You must provide the full link including the protocol. For example, `http:\\www.mathworks.com` is a valid link, while `www.mathworks.com` is not a valid link. You can create a link to a protocol that is valid URL, such as a web site address, a full path to a file, or a relative path to a file.

Note `setRefLink` expects a cell array of cell arrays for the second input.

Examples

Create a list object, `ft`, and add a related standard:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setRefLink(ft, {{'IEC 61508-3, Table A.3 (3) ''Language subset'''}});
```

Create a list object, `ft`, and add a list of related standards:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setRefLink(ft, {
    {'IEC 61508-3, Table A.3 (2) ''Strongly typed programming language'''},...
    {'IEC 61508-3, Table A.3 (3) ''Language subset'''}});
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

setRetainSpaceReturn

Class: ModelAdvisor.Text
Package: ModelAdvisor

Retain spacing and returns in text

Syntax

`setRetainSpaceReturn(text, mode)`

Description

`setRetainSpaceReturn(text, mode)` specifies whether the text must retain the spaces and carriage returns.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`mode`	A Boolean value indicating whether to preserve spaces and carriage returns in the text: • `true` (default) — Preserve spaces and carriage returns. • `false` — Do not preserve spaces and carriage returns.

Examples

```
t1 = ModelAdvisor.Text('MathWorks home page');
setRetainSpaceReturn(t1, 'true');
```

See Also

“Model Advisor Customization”

Topics

"Create Model Advisor Checks"

setRowHeading

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table row title

Syntax

```
setRowHeading(table, row, heading)
```

Description

`setRowHeading(table, row, heading)` specifies a title for the designated table row.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying row number
`heading`	A character vector, element object, or object array specifying the table row title

Examples

```
table1 = ModelAdvisor.Table(2,3);
setRowHeading(table1, 1, 'Row 1 Title');
setRowHeading(table1, 2, 'Row 2 Title');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setRowHeadingAlign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table row title alignment

Syntax

`setRowHeadingAlign(table, row, alignment)`

Description

`setRowHeadingAlign(table, row, alignment)` specifies the alignment for the designated table row.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying row number.
`alignment`	Cell alignment, specified as one of the following: • `'left'` (default) • `'right'` • `'center'`

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setRowHeading(table1, 1, 'Row 1 Title');
setRowHeadingAlign(table1, 1, 'center');
setRowHeading(table1, 2, 'Row 2 Title');
setRowHeadingAlign(table1, 2, 'center');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setRowHeadingValign

Class: ModelAdvisor.Table
Package: ModelAdvisor

Specify table row title vertical alignment

Syntax

`setRowHeadingValign(table, row, alignment)`

Description

`setRowHeadingValign(table, row, alignment)` specifies the vertical alignment for the designated table row.

Input Arguments

`table`	Instantiation of the `ModelAdvisor.Table` class
`row`	An integer specifying row number.
`alignment`	Cell vertical alignment, specified as one of the following: • `'top'` (default) • `'middle'` • `'bottom'`

Examples

```
table1 = ModelAdvisor.Table(2, 3);
setRowHeading(table1, 1, 'Row 1 Title');
setRowHeadingValign(table1, 1, 'middle');
setRowHeading(table1, 2, 'Row 2 Title');
setRowHeadingValign(table1, 2, 'middle');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”

setRowSpan

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Specify rows for input parameter

Syntax

```
setRowSpan(input_param, [start_row end_row])
```

Description

`setRowSpan(input_param, [start_row end_row])` specifies the number of rows that the parameter occupies. Specify where you want an input parameter located in the layout grid when there are multiple input parameters.

Input Arguments

`input_param`	The input parameter object
`start_row`	A positive integer representing the first row that the input parameter occupies in the layout grid
`end_row`	A positive integer representing the last row that the input parameter occupies in the layout grid

Examples

```
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
```

setSubBar

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add line between subcheck results

Syntax

`setSubBar(ft_obj, value)`

Description

`setSubBar(ft_obj, value)` is an optional method that adds lines between results for subchecks. *`ft_obj`* is a handle to a template object. *`value`* is a boolean value that specifies when the Model Advisor includes a line between subchecks in the check results. By default, the value is `true`, and the Model Advisor displays the bar. The Model Advisor does not display the bar when you set the value to `false`.

Examples

Create a list object, `ft`, turn off the subbar:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubBar(ft, false);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

setSubResultStatus

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add status to check or subcheck result

Syntax

`setSubResultStatus(ft_obj, 'status')`

Description

`setSubResultStatus(ft_obj, 'status')` is an optional method that displays the status in the result. Use this method to display the status of the check or subcheck in the result. *`ft_obj`* is a handle to a template object. *`status`* is a character vector identifying the status of the check:

`Pass`
`Warn`
`Fail`

Examples

Create a list object, `ft`, and add a passing status:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubResutlStatus(ft, 'Pass');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

“Format Check Results”

setSubResultStatusText

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add text below status in result

Syntax

`setSubResultStatusText(ft_obj, message)`

Description

`setSubResultStatusText(ft_obj, message)` is an optional method that displays text below the status in the result. Use this method to describe the status.

Input Arguments

`ft_obj`

A handle to a template object.

`message`

A character vector or a handle to a formatting object that the Model Advisor displays below the status in the report.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

Examples

Create a list object, `ft`, add a passing status and a description of why the check passed:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubResutlStatus(ft, 'Pass');
setSubResultStatusText(ft, ['Constructs that are not supported when '...
    'generating code were not found in the model or subsystem']);
```

See Also

"Model Advisor Customization"

Topics

"Model Advisor Customization"
"Format Check Results"

setSubscript

Class: ModelAdvisor.Text
Package: ModelAdvisor

Specify subscripted text

Syntax

`setSubscript(text, *mode*)`

Description

`setSubscript(text, mode)` indicates whether to make `text` subscript.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`mode`	A Boolean value indicating subscripted formatting of text: • `true` — Make the text subscript. • `false` — Do not make the text subscript.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
setSubscript(t1, 'true');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setSuperscript

Class: ModelAdvisor.Text
Package: ModelAdvisor

Specify superscripted text

Syntax

`setSuperscript(text, *mode*)`

Description

`setSuperscript(text, *mode*)` indicates whether to make `text` superscript.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
mode	A Boolean value indicating superscripted formatting of text: • `true` — Make the text superscript. • `false` — Do not make the text superscript.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
setSuperscript(t1, 'true');
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"

setSubTitle

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add title for subcheck in result

Syntax

`setSubTitle(ft_obj, title)`

Description

`setSubTitle(ft_obj, title)` is an optional method that adds a subcheck result title. Use this method when you create subchecks to distinguish between them in the result.

Input Arguments

`ft_obj`

A handle to a template object.

`title`

A character vector or a handle to a formatting object specifying the title of the subcheck.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

Examples

Create a list object, `ft`, and add a subcheck title:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubTitle(ft, ['Check for constructs in the model '...
    'that are not supported when generating code']);
```

See Also

"Model Advisor Customization"

Topics

"Create Model Advisor Checks"
"Format Check Results"

setTableInfo

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add data to table

Syntax

`setTableInfo(ft_obj, {data})`

Description

`setTableInfo(ft_obj, {data})` is an optional method that creates a table. `ft_obj` is a handle to a table template object. `data` is a cell array of character vectors or objects specifying the information in the body of the table. The Model Advisor creates hyperlinks to objects. If you do not add data to the table, the Model Advisor does not display the table in the result.

Note Before creating a table, you must specify column titles using the `setColTitle` method.

Examples

Create a table object, `ft`, add column titles, and add data to the table:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');
setColTitle(ft, {'Index', 'Block Name'});
setTableInfo(ft, {'1', 'Gain'});
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

setTableTitle

Class: ModelAdvisor.FormatTemplate
Package: ModelAdvisor

Add title to table

Syntax

`setTableTitle(ft_obj, title)`

Description

`setTableTitle(ft_obj, title)` is an optional method that adds a title to a table.

Input Arguments

`ft_obj`

A handle to a template object.

`title`

A character vector or a handle to a formatting object specifying the title of the table.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The title appears above the table. If you do not add data to the table, the Model Advisor does not display the table and title in the result.

Examples

Create a table object, `ft`, and add a table title:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');
setTableTitle(ft, 'Table of fonts and styles used in model');
```

See Also

“Model Advisor Customization”

Topics

“Create Model Advisor Checks”
“Format Check Results”

setType

Class: ModelAdvisor.List
Package: ModelAdvisor

Specify list type

Syntax

`setType(list_obj, listType)`

Description

`setType(list_obj, listType)` specifies the type of list the `ModelAdvisor.List` constructor creates.

Input Arguments

`list_obj`	Instantiation of the `ModelAdvisor.List` class
`listType`	Specifies the list type: • `numbered` • `bulleted`

Examples

```
subList = ModelAdvisor.List();
subList.setType('numbered')
subList.addItem(ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
subList.addItem(ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”

setUnderlined

Class: ModelAdvisor.Text
Package: ModelAdvisor

Underline text

Syntax

`setUnderlined(text, mode)`

Description

`setUnderlined(text, mode)` indicates whether to underline `text`.

Input Arguments

`text`	Instantiation of the `ModelAdvisor.Text` class
`mode`	A Boolean value indicating underlined formatting of text: • `true` — Underline the text. • `false` — Do not underline the text.

Examples

```
t1 = ModelAdvisor.Text('This is some text');
setUnderlined(t1, 'true');
```

See Also

"Model Advisor Customization"

Topics

“Create Model Advisor Checks”

slmetric.Engine class

Package: slmetric

Collect metric data on models or model components

Description

Use a `slmetric.Engine` object to collect metric data on models by calling `slmetric.Engine.execute`. Use `slmetric.Engine.getMetrics` to access the metric data and return an array of `slmetric.metric.ResultCollection` objects. This metric data is persistent in the simulation cache folder. Future instantiations of the `slmetric.Engine` object for the same model can access the cached metric data without regenerating the metric data.

Construction

`metric_engine = slmetric.Engine()` creates a metric engine object.

Properties

`AnalysisRoot` — Name of root model or subsystem on which to collect metric data
character vector

Name of root model or subsystem on which to collect metric data, as specified by the `slmetric.Engine.setAnalysisRoot` method. This property is read-only.

`AnalyzeLibraries` — Collect metric data on library linked subsystems in the model
1 (default)

Specify if the metric engine analyzes library-linked subsystems in the root model, including libraries inside referenced models under the root. Metric analysis does not include linked blocks to Simulink built-in libraries. Set this parameter to `false` or `0` to not include libraries in the metric analysis.

Data Types: `logical`

`AnalyzeModelReferences` — Collect metric data on all referenced models under the root model

1 (default)

Specify if the metric engine analyzes referenced models in your root model. Set this parameter to `false` or `0` to not include referenced models in the metric analysis.

Data Types: `logical`

Methods

execute	Collect metric data
getAnalysisRootMetric	Get metric data for one metric for analysis root only
getErrorLog	Get error log
getMetricDistribution	Get metric distribution
getMetrics	Access model metric data
getStatistics	Get statistics on metric data
setAnalysisRoot	Specify model or subsystem for metric analysis
exportMetrics	Export model metrics

Examples

Collect and Access Metric Data for a Model

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis, these properties are on by d
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2016a

slmetric.metric.getAvailableMetrics

Package: slmetric.metric

Obtain available metrics

Syntax

```
IDs = slmetric.metric.getAvailableMetrics()
[IDs,props] = slmetric.metric.getAvailableMetrics()
```

Description

`IDs = slmetric.metric.getAvailableMetrics()` get metric identifiers for available metrics in the metric engine.

`[IDs,props] = slmetric.metric.getAvailableMetrics()` get metric identifiers and properties.

Examples

Obtain Available Metric IDs for Model

This example shows how to obtain the available model metric IDs.

```
ID = slmetric.metric.getAvailableMetrics()
```

```
ID =

    'mathworks.metrics.CyclomaticComplexity'
    'mathworks.metrics.DescriptiveBlockNames'
    'mathworks.metrics.IOCount'
    'mathworks.metrics.LayerSeparation'
    'mathworks.metrics.LibraryLinkCount'
    'mathworks.metrics.MatlabCodeAnalyzerWarnings'
```

```
    'mathworks.metrics.MatlabLOCCount'
    'mathworks.metrics.ModelAdvisorCheckCompliance.misra_c'
    'mathworks.metrics.ModelAdvisorCheckCompliance.do178'
    'mathworks.metrics.ModelAdvisorCheckCompliance.ISO26262'
    'mathworks.metrics.ModelAdvisorCheckCompliance.maab'
    'mathworks.metrics.ModelAdvisorCheckIssues.misra_c'
    'mathworks.metrics.ModelAdvisorCheckIssues.do178'
    'mathworks.metrics.ModelAdvisorCheckIssues.ISO26262'
    'mathworks.metrics.ModelAdvisorCheckIssues.maab'
    'mathworks.metrics.SimulinkBlockCount'
    'mathworks.metrics.StateflowChartObjectCount'
    'mathworks.metrics.StateflowLOCCount'
    'mathworks.metrics.SubSystemCount'
    'mathworks.metrics.SubSystemDepth'
```

Obtain Available Metric IDs and Metric Properties

This example shows how to obtain the available model metric properties.

```
[ID,PROPS]=slmetric.metric.getAvailableMetrics()
```

```
ID =

 'mathworks.metrics.CyclomaticComplexity'
    'mathworks.metrics.DescriptiveBlockNames'
    'mathworks.metrics.IOCount'
    'mathworks.metrics.LayerSeparation'
    'mathworks.metrics.LibraryLinkCount'
    'mathworks.metrics.MatlabCodeAnalyzerWarnings'
    'mathworks.metrics.MatlabLOCCount'
    'mathworks.metrics.ModelAdvisorCheckCompliance.misra_c'
    'mathworks.metrics.ModelAdvisorCheckCompliance.do178'
    'mathworks.metrics.ModelAdvisorCheckCompliance.ISO26262'
    'mathworks.metrics.ModelAdvisorCheckCompliance.maab'
    'mathworks.metrics.ModelAdvisorCheckIssues.misra_c'
    'mathworks.metrics.ModelAdvisorCheckIssues.do178'
    'mathworks.metrics.ModelAdvisorCheckIssues.ISO26262'
    'mathworks.metrics.ModelAdvisorCheckIssues.maab'
    'mathworks.metrics.SimulinkBlockCount'
    'mathworks.metrics.StateflowChartObjectCount'
    'mathworks.metrics.StateflowLOCCount'
    'mathworks.metrics.SubSystemCount'
    'mathworks.metrics.SubSystemDepth'
```

```
PROPS =

1x20 struct array with fields:

    Name
    Description
    IsBuiltIn
    Version
```

Output Arguments

`IDs` — Metric identifiers

cell array of character vectors

Metric identifiers in the metric engine.

`props` — Metric properties

structure array

Metric properties, returned as a structure array with the following fields:

Name	Name of the metric algorithm.
Description	Description of the metric algorithm.
IsBuiltIn	Boolean indicating if the metric is included with Simulink Check.
Version	Metric algorithm version.

Data Types: `struct`

See Also

`slmetric.Engine` | `slmetric.metric.Result` | `slmetric.metric.ResultCollection`

Introduced in R2016a

slmetric.metric.Result class

Package: slmetric.metric

Metric data for specified model component and metric algorithm

Description

Instances of `slmetric.metric.Result` contain the metric data for a specified model component and metric algorithm.

Construction

`metric_result = slmetric.metric.Result` creates a handle to a metric results object.

Properties

`ID` — Numeric identifier
integer

Unique numeric identifier for the metric result object. This property is read-only.

Data Types: `uint64`

`ComponentID` — Component ID
character vector

Unique identifier of the component object for which the metric is calculated. Use `ComponentID` to trace the generated result object to the analyzed component. Set the `ComponentID` or `ComponentPath` properties by using the `slmetric.metric.Metric.algorithm` method.

This property is read/write.

Data Types: `char`

ComponentPath — Component path
character vector

Component path for which metric is calculated. Use ComponentPath as an alternative to setting the ComponentID property. The metric engine converts the ComponentPath to a ComponentID. Set the ComponentID or ComponentPath properties by using the slmetric.metric.Metric.algorithm method.

This property is read/write.

Data Types: char

MetricID — Metric identifier
character vector

Metric identifier for "Model Metrics" on page 2-309 or custom model metrics that you create. You can get metric identifiers by calling slmetric.metric.getAvailableMetrics.

This property is read/write.

Data Types: char

Value — Metric value
double (default)

Metric scalar value, generated by the algorithm for the metric specified by MetricID and the component specified by ComponentID.

This property is read/write.

Data Types: double

AggregatedValue — Aggregated metric value
double (default)

Metric value aggregated across the model hierarchy. The metric engine implicitly aggregates the metric values. Do not set this property.

This property is read-only.

Data Types: double

`Measures` **— Metric measures**
double array

Metric measures, optionally specified by the metric algorithm. Metric measures contain detailed information about the metric value. For example, for a metric that counts the number of blocks per subsystem, you can specify measures that contain the number of virtual and nonvirtual blocks. The metric value is the sum of the virtual and nonvirtual block count.

Set the property by using the `slmetric.metric.Metric.algorithm` method. This property is read/write.

Data Types: `double`

`AggregatedMeasures` **— Aggregated metric measures**
double array

Metric measures value aggregated across the model hierarchy. The metric engine implicitly aggregates the metric measure values. Do not set this property.

This property is read-only.

Data Types: `double`

`Details` **— Metric result details**
array of `slmetric.metric.ResultDetail` objects

Details about what the metric engine counts for the `Value` property

This property is read/write.

`UserData` **— User data**
character vector

User data optionally provided by the metric algorithm.

This property is read/write.

Data Types: `char`

Examples

Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis,
%     these properties are on by default
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
```

```
    disp(' ');
end
```

See Also

`slmetric.Engine` | `slmetric.metric.Metric` |
`slmetric.metric.ResultCollection`

Topics

“Collect Model Metrics Programmatically”
“Model Metrics” on page 2-309

Introduced in R2016a

slmetric.metric.ResultCollection class

Package: slmetric.metric

Metric data for specified model metric

Description

Instances of `slmetric.metric.ResultCollection` contain the metric data for a specific model metric.

Construction

`metricRC = slmetric.metric.ResultCollection` creates a handle to a metric result collection object.

Properties

`MetricID` — Metric identifier
character vector

Metric identifier for a MathWorks metric or a custom metric. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

`Status` — Unique identifier
integer

Status code of metric execution. This property is read-only.

Integer	Status
1	No result. Metric algorithm is not applicable to the analyzed system. Components analyzed by the metric not found, or metric with compile requirement cannot be executed on library model.
0	Result collected.

Integer	Status
-1	No result. Error executing metric.
-2	No result available from previous run.
-3	No result. Compilation error.
-4	Empty result. Missing prerequisite.

`Outdated` — Determine if metric data is current
logical

If `true`, the metric data is out-of-date because the model or source files have changed. This property is read-only.

`Results` — Metric data collected for executing one or more metrics
array of `slmetric.metric.Result` objects

Metric data collected when you call the `slmetric.Engine.execute` method for one or more metrics. This property is read-only.

Examples

Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdlref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis, these properties are on by d
metric_engine.AnalyzeModelReferences = 1;
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root',  'sldemo_mdlref_basic');
```

Collect model metric data.

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp(['  ComponentPath: ', result(m).ComponentPath]);
            disp(['  Value: ', num2str(result(m).Value)]);
            disp(['  AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

See Also

`slmetric.Engine` | `slmetric.metric.Result` | `slmetric.metric.getAvailableMetrics`

Introduced in R2016a

Attributes property

Class: ModelAdvisor.ListViewParameter
Package: ModelAdvisor

Attributes to display in Model Advisor Report Explorer

Values

Cell array

Default: `{}` (empty cell array)

Description

The `Attributes` property specifies the attributes to display in the center pane of the Model Advisor Results Explorer.

Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object')';
myLVParam.Attributes = {'FontName'}; % name is default property
```

CallbackContext property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Specify when to run check

Values

`'PostCompile'`
`'None'` (default)

Description

The `CallbackContext` property specifies the context for checking the model or subsystem.

`'None'`	No special requirements for the model before checking.
`'Postcompile'`	The model must be compiled.

CallbackHandle property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Callback function handle for check

Values

Function handle.

An empty handle `[ ]` is the default.

Description

The `CallbackHandle` property specifies the handle to the check callback function.

CallbackStyle property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Callback function type

Values

`'StyleOne'` (default)
`'StyleTwo'`
`'StyleThree'`

Description

The `CallbackStyle` property specifies the type of the callback function.

`'StyleOne'`	Simple check callback function
`'StyleTwo'`	Detailed check callback function
`'StyleThree'`	Check callback function with hyperlinked results

EmitInputParametersToReport property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Display check input parameters in the Model Advisor report

Values

`'true'` (default)
`'false'`

Description

The `EmitInputParametersToReport` property specifies the display of check input parameters in the Model Advisor report.

`'true'`	Display check input parameters in the Model Advisor report
`'false'`	Do not display check input parameters in the Model Advisor report

Data property

Class: ModelAdvisor.ListViewParameter
Package: ModelAdvisor

Objects in Model Advisor Result Explorer

Values

Array of Simulink objects

Default: `[]` (empty array)

Description

The `Data` property specifies the objects displayed in the Model Advisor Result Explorer.

Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object')';
```

Description property

Class: ModelAdvisor.Action
Package: ModelAdvisor

Message in **Action** box

Values

Character vector

Default:`''` (empty character vector)

Description

The `Description` property specifies the message displayed in the Action box.

Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
%Specify a callback function for the action
myAction.setCallbackFcn(@sampleActionCB);
myAction.Name='Fix block fonts';
myAction.Description=...
   'Click the button to update all blocks with specified font';
```

Description property

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Description of folder

Values

Character vector

Default: `''` (empty character vector)

Description

The `Description` property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.Description='Sample Factory Group';
```

Description property

Class: ModelAdvisor.Group
Package: ModelAdvisor

Description of folder

Values

Character vector

Default: `''` (empty character vector)

Description

The `Description` property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
MAG.Description='This is my group';
```

Description property

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Description of input parameter

Values

Character vector.

Default: `''` (empty character vector)

Description

The `Description` property specifies a description of the input parameter. Details about the check are displayed in the right pane of the Model Advisor.

Examples

```
% define input parameters
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
```

Description property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Description of task

Values

Character vector

Default: `''` (empty character vector)

Description

The `Description` property is a description of the task that the Model Advisor displays in the **Analysis** box.

When adding checks as tasks, the Model Advisor uses the task `Description` property instead of the check `TitleTips` property.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.DisplayName='Example task 1';
MAT1.Description='This is the first example task.'

MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT2.DisplayName='Example task 2';
MAT2.Description='This is the second example task.'

MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
MAT3.DisplayName='Example task 3';
MAT3.Description='This is the third example task.'
```

DisplayName property

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Name of folder

Values

Character vector

Default:`''` (empty character vector)

Description

The `DisplayName` specifies the name of the folder that is displayed in the Model Advisor.

Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='Sample Factory Group';
```

DisplayName property

Class: ModelAdvisor.Group
Package: ModelAdvisor

Name of folder

Values

Character vector

Default:`''` (empty character vector)

Description

The `DisplayName` specifies the name of the folder that is displayed in the Model Advisor.

Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
MAG.DisplayName='My Group';
```

DisplayName property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Name of task

Values

Character vector

Default: `''` (empty character vector)

Description

The `DisplayName` property specifies the name of the task. The Model Advisor displays each custom task in the tree using the name of the task. Therefore, you should specify a unique name for each task. When you specify the same name for multiple tasks, the Model Advisor generates a warning.

When adding checks as tasks, the Model Advisor uses the task `DisplayName` property instead of the check `Title` property.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.DisplayName='Example task with input parameter and auto-fix ability';

MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');
MAT2.DisplayName='Example task 2';

MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
MAT3.DisplayName='Example task 3';
```

Enable property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Indicate whether user can enable or disable check

Values

`true` (default)
`false`

Description

The `Enable` property specifies whether the user can enable or disable the check.

`true`	Display the check box control
`false`	Hide the check box control

Enable property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Indicate if user can enable and disable task

Values

`true` (default)
`false`

Description

The `Enable` property specifies whether the user can enable or disable a task.

`true` (default)	Display the check box control for task
`false`	Hide the check box control for task

When adding checks as tasks, the Model Advisor uses the task `Enable` property instead of the check `Enable` property.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.Enable = false;
```

Entries property

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Drop-down list entries

Values

Depends on the value of the `Type` property.

Description

The `Entries` property is valid only when the `Type` property is one of the following:

- `Enum`
- `ComboBox`
- `PushButton`

Examples

```
inputParam3 = ModelAdvisor.InputParameter;
inputParam3.Name='Valid font';
inputParam3.Type='Combobox';
inputParam3.Description='sample tooltip';
inputParam3.Entries={'Arial', 'Arial Black'};
```

ID property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Identifier for check

Values

Character vector

Default: `''` (empty character vector)

Description

The `ID` property specifies a permanent, unique identifier for the check. Note the following about the `ID` property:

- You must specify this property.
- The value of `ID` must remain constant.
- The Model Advisor generates an error if `ID` is not unique.
- Tasks and factory group definitions must refer to checks by `ID`.

ID property

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Identifier for folder

Values

Character vector

Description

The `ID` property specifies a permanent, unique identifier for the folder.

Note

- You must specify this field.
- The value of `ID` must remain constant.
- The Model Advisor generates an error if `ID` is not unique.
- Group definitions must refer to other groups by `ID`.

ID property

Class: ModelAdvisor.Group
Package: ModelAdvisor

Identifier for folder

Values

Character vector

Description

The `ID` property specifies a permanent, unique identifier for the folder.

Note

- You must specify this field.
- The value of `ID` must remain constant.
- The Model Advisor generates an error if `ID` is not unique.
- Group definitions must refer to other groups by `ID`.

ID property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Identifier for task

Values

Character vector

Default: `''` (empty character vector)

Description

The `ID` property specifies a permanent, unique identifier for the task.

Note

- The Model Advisor automatically assigns a unique identifier to `ID` if you do not specify it.
- The value of `ID` must remain constant.
- The Model Advisor generates an error if `ID` is not unique.
- Group definitions must refer to tasks using `ID`.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.ID='Task_ID_1234';
```

LicenseName property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Product license names required to display and run check

Values

Cell array of product license names
`{}`(empty cell array) (default)

Description

The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

Tip To find the text for license strings, type `help license` at the MATLAB command line.

LicenseName property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Product license names required to display and run task

Values

Cell array of product license names

Default: `{}` (empty cell array)

Description

The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

If you specify `ModelAdvisor.Check.LicenseName`, the Model Advisor displays the check when the union of both properties is true.

Tip To find the text for license strings, type `help license` at the MATLAB command line.

ListViewVisible property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Status of **Explore Result** button

Values

`false` (default)
`true`

Description

The `ListViewVisible` property is a Boolean value that sets the status of the **Explore Result** button.

`true`	Display the **Explore Result** button.
`false`	Hide the **Explore Result** button.

Examples

```
% add 'Explore Result' button
rec.ListViewVisible = true;
```

MAObj property

Class: ModelAdvisor.FactoryGroup
Package: ModelAdvisor

Model Advisor object

Values

Handle to a `Simulink.ModelAdvisor` object

Description

The `MAObj` property specifies a handle to the current Model Advisor object.

MAObj property

Class: ModelAdvisor.Group
Package: ModelAdvisor

Model Advisor object

Values

Handle to `Simulink.ModelAdvisor` object

Description

The `MAObj` property specifies a handle to the current Model Advisor object.

MAObj property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Model Advisor object

Values

Handle to a `Simulink.ModelAdvisor` object

Description

The `MAObj` property specifies the current Model Advisor object.

When adding checks as tasks, the Model Advisor uses the task `MAObj` property instead of the check `MAObj` property.

Name property

Class: ModelAdvisor.Action
Package: ModelAdvisor

Action button label

Values

Character vector

Default: `''` (empty character vector)

Description

The `Name` property specifies the label for the action button. This property is required.

Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
%Specify a callback function for the action
myAction.setCallbackFcn(@sampleActionCB);
myAction.Name='Fix block fonts';
```

Name property

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Input parameter name

Values

Character vector.

Default: `''` (empty character vector)

Description

The `Name` property specifies the name of the input parameter in the custom check.

Examples

```
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
```

Name property

Class: ModelAdvisor.ListViewParameter
Package: ModelAdvisor

Drop-down list entry

Values

Character vector

Default: `''` (empty character vector)

Description

The `Name` property specifies an entry in the **Show** drop-down list in the Model Advisor Result Explorer.

Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
```

Result property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Results cell array

Values

Cell array

Default: `{}` (empty cell array)

Description

The `Result` property specifies the cell array for storing the results that are returned by the callback function specified in `CallbackHandle`.

Tip To set the icon associated with the check, use the `Simulink.ModelAdvisor` `setCheckResultStatus` and `setCheckErrorSeverity` methods.

---

supportExclusion property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Set to support exclusions

Values

Boolean value specifying that the check supports exclusions.
`true` The check supports exclusions.
`false` (default). The check does not support exclusions.

Description

The `supportExclusion` property specifies whether the check supports exclusions.

`'true'`	Check supports exclusions.
`'false'`	Check does not support exclusions.

Examples

```
% specify that a check supports exclusions
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.supportExclusion = true;
```

SupportLibrary property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Set to support library models

Values

Boolean value specifying that the check supports library models.
`true`. The check supports library models.
`false` (default). The check does not support library models.

Description

The `SupportLibrary` property specifies whether the check supports library models.

`'true'`	Check supports library models.
`'false'`	Check does not support library models.

Examples

```
% specify that a check supports library models
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.SupportLibrary = true;
```

Title property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Name of check

Values

Character vector

Default: `''` (empty character vector)

Description

The `Title` property specifies the name of the check in the Model Advisor. The Model Advisor displays each custom check in the tree using the title of the check. Therefore, you should specify a unique title for each check. When you specify the same title for multiple checks, the Model Advisor generates a warning.

Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
```

TitleTips property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Description of check

Values

Character vector

Default: `''` (empty character vector)

Description

The `TitleTips` property specifies a description of the check. Details about the check are displayed in the right pane of the Model Advisor.

Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
```

Type property

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Input parameter type

Values

character vector

Default: `''`

Description

The `Type` property specifies the type of input parameter.

Use the `Type` property with the `Value` and `Entries` properties to define input parameters.

Valid values are listed in the following table.

Type	Data Type	Default Value	Description
`Bool`	Boolean	`false`	A check box
`ComboBox`	Cell array	First entry in the list	A drop-down menu • Use `Entries` to define the entries in the list. • Use `Value` to indicate a specific entry in the menu or to enter a value not in the list.

Type	Data Type	Default Value	Description
`Enum`	Cell array	First entry in the list	A drop-down menu • Use `Entries` to define the entries in the list. • Use `Value` to indicate a specific entry in the list.
`PushButton`	N/A	N/A	A button When you click the button, the callback function specified by `Entries` is called.
`String`	Character vector	`''`	A text box

Examples

```
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
```

validate

Class: Advisor.authoring.DataFile
Package: Advisor.authoring

Validate XML data file used for model configuration check

Syntax

`msg = Advisor.authoring.DataFile.validate(dataFile)`

Description

`msg = Advisor.authoring.DataFile.validate(dataFile)` validates the syntax of the XML data file used for model configuration checks.

Input Arguments

`dataFile`	XML data file name (character vector)

Examples

```
dataFile = 'myDataFile.xml';
msg = Advisor.authoring.DataFile.validate(dataFile);

if isempty(msg)
   disp('Data file passed the XSD schema validation.');
else
   disp(msg);
end
```

See Also

`Advisor.authoring.CustomCheck` | `Advisor.authoring.generateConfigurationParameterDataFile`

Topics

"Create Check for Model Configuration Parameters"

Value property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Status of check

Values

`'true'` (default)
`'false'`

Description

The `Value` property specifies the initial status of the check. When you use the `Value` property to specify the initial status of the check, you enable or disable **Run This Check** in the Model Advisor window.

If you want to specify the initial status of a check in the **By Product** folder, before starting Model Advisor, make sure `ModelAdvisor.Preferences.DeselectByProduct` is `false`.

`'true'`	Check is enabled
`'false'`	Check is disabled

Examples

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```

See Also

`ModelAdvisor.Preferences`

Value property

Class: ModelAdvisor.InputParameter
Package: ModelAdvisor

Value of input parameter

Values

Depends on the `Type` property.

Description

The `Value` property specifies the initial value of the input parameter. This property is valid only when the `Type` property is one of the following:

- `'Bool'`
- `'String'`
- `'Enum'`
- `'ComboBox'`

Examples

```
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
```

Value property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Status of task

Values

`'true'` (default) — Initial status of task is enabled
`'false'` — Initial status of task is disabled

Description

The `Value` property indicates the initial status of a task—whether it is enabled or disabled.

When adding checks as tasks, the Model Advisor uses the task `Value` property instead of the check `Value` property.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.Value ='false';
```

view

View Model Advisor run results for checks

Syntax

`view(CheckResultObj)`

Description

`view(CheckResultObj)` opens a web browser and displays the results of the check specified by `CheckResultObj`. `CheckResultObj` is a `ModelAdvisor.CheckResult` object returned by `ModelAdvisor.run`.

Input Arguments

`CheckResultObj`

`ModelAdvisor.CheckResult` object which is a part of a `ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

Examples

View the Model Advisor run results for the first check in the `slvnvdemo_mdladv_config` configuration file:

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% View the 'Identify unconnected...' check result.
view(SysResultObjArray{1}.CheckResultObjs(1))
```

Alternatives

"View Model Advisor Report"

See Also

`ModelAdvisor.run` | `ModelAdvisor.summaryReport` | `viewReport`

Topics

"Checking Systems Programmatically"
"Check Multiple Systems in Parallel"
"Create a Function for Checking Multiple Systems in Parallel"
"Automate Model Advisor Check Execution"
"Archive and View Model Advisor Run Results"

Introduced in R2010b

viewReport

View Model Advisor run results for systems

Syntax

```
viewReport(SysResultObjArray)
viewReport(SysResultObjArray,'MA')
viewReport(SysResultObjArray,'Cmd')
```

Description

`viewReport(SysResultObjArray)` opens the Model Advisor Report for the system specified by `SysResultObjArray`. `SysResultObjArray` is a `ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

`viewReport(SysResultObjArray,'MA')` opens the Model Advisor and displays the results of the run for the system specified by `SysResultObjArray`.

`viewReport(SysResultObjArray,'Cmd')` displays the Model Advisor run summary in the Command Window for the systems specified by `SysResultObjArray`.

Input Arguments

`SysResultObjArray`

`ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

Default:

Examples

Open the Model Advisor report for `sldemo_auto_climatecontrol/Heater Control`.

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Open the Model Advisor report.
viewReport(SysResultObjArray{1})
```

Open Model Advisor and display results for `sldemo_auto_climatecontrol/Heater Control`.

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Open the Model Advisor and display results.
viewReport(SysResultObjArray{1}, 'MA')
```

Display results in the Command Window for `sldemo_auto_climatecontrol/Heater Control`.

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvnvdemo_mdladv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
    'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Display results in the Command Window.
viewReport(SysResultObjArray{1}, 'Cmd')
```

Alternatives

- “View Model Advisor Report”
- “View Results in Model Advisor GUI”
- “View Results in Command Window”

See Also

`ModelAdvisor.run` | `ModelAdvisor.summaryReport` | `view`

Topics

"Checking Systems Programmatically"
"Check Multiple Systems in Parallel"
"Create a Function for Checking Multiple Systems in Parallel"
"Automate Model Advisor Check Execution"
"Archive and View Model Advisor Run Results"

Introduced in R2010b

Visible property

Class: ModelAdvisor.Check
Package: ModelAdvisor

Indicate to display or hide check

Values

`'true'` (default)
`'false'`

Description

The `Visible` property specifies whether the Model Advisor displays the check.

`'true'`	Display the check
`'false'`	Hide the check

Examples

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```

Visible property

Class: ModelAdvisor.Task
Package: ModelAdvisor

Indicate to display or hide task

Values

`'true'` (default) — Display task in the Model Advisor
`'false'` — Hide task

Description

The `Visible` property specifies whether the Model Advisor displays the task.

Caution When adding checks as tasks, you cannot specify both the task and check `Visible` properties, you must specify one or the other. If you specify both properties, the Model Advisor generates an error when the check `Visible` property is `false`.

Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');
MAT1.Visible ='false';
```

slmetric.metric.registerMetric

Package: slmetric.metric

Register a custom model metric with the model metric repository

Syntax

```
[MetricID,err_msg] = slmetric.metric.registerMetric(classname)
```

Description

`[MetricID,err_msg] = slmetric.metric.registerMetric(classname)` register a custom model metric with the model metric repository. The new metric class must be on the MATLAB search path and derived from `slmetric.metric.Metric`.

Examples

Register a Custom Model Metric with the Model Metric Repository

This example shows how to register a custom model metric.

Create a new metric class, derived from `slmetric.metric.Metric`, called `my_metric`.

```
slmetric.metric.createNewMetricClass('my_metric')
```

Finish the custom model metric implementation and testing.

Register the new custom metric in the model metric repository.

```
[MetricID, err_msg] = slmetric.metric.registerMetric('my_metric');
```

Input Arguments

`classname` — Metric class name
character vector

New metric class name.

Data Types: `char`

Output Arguments

`MetricID` — Metric ID
character vector

Unique metric identifier.

Data Types: `char`

`err_msg` — Error message
character vector

If you cannot register a new class, the function returns an error message.

Data Types: `char`

See Also

`slmetric.metric.Metric` | `slmetric.metric.createNewMetricClass` | `slmetric.metric.refresh` | `slmetric.metric.unregisterMetric`

Introduced in R2016a

slmetric.metric.unregisterMetric

Package: slmetric.metric

Unregister a custom model metric from the model metric repository

Syntax

`slmetric.metric.unregisterMetric(MetricID)`

Description

`slmetric.metric.unregisterMetric(MetricID)` unregister a custom model metric from the model metric repository.

Input Arguments

`MetricID` — Unique metric identifier
character vector

Metric identifier for a custom model metric that you created.

See Also

`slmetric.metric.Metric` | `slmetric.metric.createNewMetricClass` | `slmetric.metric.refresh` | `slmetric.metric.registerMetric`

Introduced in R2016a

slmetric.metric.refresh

Package: slmetric.metric

Update available model metrics

Syntax

```
slmetric.metric.refresh()
```

Description

`slmetric.metric.refresh()` updates available metrics after manual updates to the metric registration file.

See Also

`slmetric.metric.Metric` | `slmetric.metric.createNewMetricClass` | `slmetric.metric.registerMetric` | `slmetric.metric.unregisterMetric`

Introduced in R2016a

slmetric.metric.createNewMetricClass

Package: slmetric.metric

Create new metric class for a custom model metric

Syntax

```
slmetric.metric.createNewMetricClass(class_name)
```

Description

`slmetric.metric.createNewMetricClass(class_name)` creates a `slmetric.metric.Metric` class in the current working folder. The new metric class is used to define a custom model metric and supports the following `Advisor.component.Types`:

- `Model`
- `SubSystem`
- `ModelBlock`
- `Chart`
- `MATLABFunction`

Examples

Create a Custom Model Metric Class

This example shows how to create a new metric class `my_metric`.

Call the function and provide a name for the new metric class:

```
slmetric.metric.createNewMetricClass('my_metric')
```

The function creates a `my_metric.m` file in the current working folder.

```
slmetric.metric.createNewMetricClass('my_metric')
```

The file contains the class definition for `my_metric`, which includes the constructor and an empty metric `algorithm` method.

```
classdef my_metric < slmetric.metric.Metric
        % my_metric Summary of this metric class goes here
        %   Detailed explanation goes here
        properties
        end

        methods
            function this = my_metric()
                this.ID = 'my_metric';
                this.Description = '';
                this.ComponentScope = [Advisor.component.Types.Model, ...
                    Advisor.component.Types.SubSystem];
                this.AggregationMode = slmetric.AggregationMode.Sum;
                this.AggregateComponentDetails = true;
                this.CompileContext = 'None';
                this.Version = 1;
            end

            function res = algorithm(this, component)
                res = slmetric.metric.Result();
                res.ComponentID = component.ID;
                res.MetricID = this.ID;
                res.Value = 0;
            end
        end
end
```

Write your custom metric algorithm in `algorithm`.

When your custom metric class is working and tested, register your metric using `slmetric.metric.registerMetric`.

Input Arguments

`class_name` — Name of the new metric class

character vector

Name of the new metric class you are creating for a custom metric.

Data Types: `char`

See Also

`Advisor.component.Types` | `slmetric.metric.Metric` | `slmetric.metric.registerMetric` | `slmetric.metric.unregisterMetric`

Introduced in R2016a

exportMetrics

Class: slmetric.Engine
Package: slmetric

Export model metrics

Syntax

```
exportMetrics(metric_engine,filename)
exportMetrics(metric_engine,filename,filelocation)
```

Description

Export model metric data to an XML file.

`exportMetrics(metric_engine,filename)` exports an XML `filename` containing metric data to your current folder.

`exportMetrics(metric_engine,filename,filelocation)` exports an XML `filename` containing metric data to `filelocation`.

Input Arguments

`metric_engine` — Collects and accesses metric data
`slmetric.Engine` object

When you call `slmetric.Engine.execute`, `metric_engine` collects metric data for available metrics or for the specified `MetricIDs`. Calling `slmetric.Engine.getMetrics` accesses the collected metric data in `metric_engine`.

`filename` — XML file name
character vector

Name of XML file.

Example: 'MyMetrics.xml'

filelocation — File path
character vector

Path to XML file

Example: 'C:/mywork'

Examples

Export Metrics to Current Folder

This example shows how to export metrics for model vdp to XML file MyMetrics.xml in your current folder.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'vdp', 'RootType', 'Model');

% Generate and collect model metrics
execute(metric_engine);
rc = getMetrics(metric_engine);

% Export metrics to XML file myMetrics.xml
exportMetrics(metric_engine, 'MyMetrics.xml');
```

Export Metrics to Specified Location

This example shows how to export metrics for model vdp to XML file MyMetrics.xml in a specified folder, C:/work.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'vdp', 'RootType', 'Model');
```

```
% Collect model metrics
execute(metric_engine);
rc = getMetrics(metric_engine);

% Export metrics to XML file myMetrics.xml
exportMetrics(metric_engine, 'MyMetrics.xml', 'C:/work');
```

See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

Topics

"Collect Model Metrics Programmatically"
"Model Metrics" on page 2-309

Introduced in R2016a

clonedetection

Open Identify Modeling Clones tool

Syntax

```
clonedetection(model)
```

Description

`clonedetection(model)` opens the Identifying Modeling Clones tool for a model specified by `model`. If the specified model is not open, this command opens it.

Examples

Open Identify Modeling Clones tool for a model

Open the Identify Modeling Clones tool for `rtwdemo_preprocessor_subsys` example model:

```
clonedetection('rtwdemo_preprocessor_subsys')
```

Input Arguments

`model` — Model name
character vector

Model name or handle, specified as a character vector.

Data Types: `char`

See Also

“Enable Component Reuse with Clone Detection”

Introduced in R2017a

Model Advisor Checks

Simulink Check Checks

In this section...

Simulink Check Checks

Simulink Check checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements and modeling guidelines.

For descriptions of the modeling standards checks, see

- "DO-178C/DO-331 Checks" on page 2-7
- "IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks" on page 2-101
- "MathWorks Automotive Advisory Board Checks" on page 2-203
- "MISRA C:2012 Checks" on page 2-268
- "Secure Coding Checks for CERT C, CWE, and ISO/IEC TS 17961 Standards" on page 2-283

See Also

- "Run Model Checks" (Simulink)
- "Simulink Checks" (Simulink)

Simulink Requirements Checks

Simulink Requirements™ checks facilitate linking between requirements documentation and your model .

For descriptions of the requirements consistency checks, see "Check Requirements Consistency in Model Advisor" on page 2-303.

See Also

- "Run Model Checks" (Simulink)
- "Simulink Checks" (Simulink)

Modeling Standards Checks

Modeling standards checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements or MathWorks® Automotive Advisory Board (MAAB) modeling guidelines.

The Model Advisor performs a checkout of the Simulink Check license when you run the modeling standards checks.

For descriptions of the modeling standards checks, see

- "DO-178C/DO-331 Checks" on page 2-7
- "IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks" on page 2-101
- "MathWorks Automotive Advisory Board Checks" on page 2-203

See Also

- "Run Model Checks" (Simulink)
- "Simulink Checks" (Simulink)
- "Simulink Coder Checks" (Simulink Coder)

Modeling Standards for MAAB

Group of MathWorks Automotive Advisory Board (MAAB) checks. MAAB checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run the modeling standards for MAAB checks.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

Naming Conventions

Group of MathWorks Automotive Advisory Board (MAAB) checks related to naming conventions.

The Model Advisor performs a checkout of the Simulink Check license when you run the naming conventions checks.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

Model Architecture

Group of MathWorks Automotive Advisory Board (MAAB) checks related to model architecture.

The Model Advisor performs a checkout of the Simulink Check license when you run the model architecture checks.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)

- “MAAB Control Algorithm Modeling” (Simulink) guidelines

Model Configuration Options

Group of MathWorks Automotive Advisory Board (MAAB) checks related to model configuration options.

The Model Advisor performs a checkout of the Simulink Check license when you run the model configuration options checks.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

Simulink

Group of MathWorks Automotive Advisory Board (MAAB) checks related to the Simulink product.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to the Simulink product.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

Stateflow

Group of MathWorks Automotive Advisory Board (MAAB) checks related to the Stateflow product.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to the Stateflow product.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

MATLAB Functions

MathWorks Automotive Advisory Board (MAAB) checks related to MATLAB functions.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to MATLAB functions.

See Also

- “Run Model Checks” (Simulink)
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

DO-178C/DO-331 Checks

In this section...

In this section...
“Check Stateflow debugging options” on page 2-57
“Check Stateflow charts for transition paths that cross parallel state boundaries” on page 2-59
“Check Stateflow charts for strong data typing” on page 2-60
“Check usage of lookup table blocks” on page 2-61
“Check MATLAB Code Analyzer messages” on page 2-63
“Check MATLAB code for global variables” on page 2-64
“Check for inconsistent vector indexing methods” on page 2-65
“Check for MATLAB Function interfaces with inherited properties” on page 2-66
“Check MATLAB Function metrics” on page 2-67
“Check for blocks not recommended for C/C++ production code deployment” on page 2-69
“Check for variant blocks with 'Generate preprocessor conditionals' active” on page 2-70
“Check Stateflow charts for uniquely defined data objects” on page 2-71
“Check usage of Math Operations blocks” on page 2-72
“Check usage of Signal Routing blocks” on page 2-75
“Check usage of Logic and Bit Operations blocks” on page 2-76
“Check usage of Ports and Subsystems blocks” on page 2-78
“Display model version information” on page 2-81
“Check for root Inports with missing properties” on page 2-82
“Check for root Inports with missing range definitions” on page 2-84
“Check for root Outports with missing range definitions” on page 2-85
“Check usage of Stateflow constructs” on page 2-87
“Check safety-related solver settings for simulation time” on page 2-90
“Check safety-related solver settings for solver options” on page 2-91
“Check usage of shift operations for Stateflow data” on page 2-92
“Check assignment operations in Stateflow Charts” on page 2-93
“Check Stateflow charts for unary operators” on page 2-94

In this section...

DO-178C/DO-331 Checks

DO-178C/DO-331 checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements.

The Model Advisor performs a checkout of the Simulink Check license when you run the DO-178C/DO-331 checks.

These checks are qualified by the DO Qualification Kit for use in projects involving the DO-178 standard and related standards.

See Also

- "Simulink Checks" (Simulink)
- "Simulink Coder Checks" (Simulink Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

Check model object names

Check ID: `mathworks.do178.hisl_0032`

Check model object names.

Description

This check verifies that the following model object names comply with your own modeling guidelines or the high-integrity modeling guidelines. The check also verifies that the model object does not use a reserved name.

- Blocks
- Signals
- Parameters

- Busses
- Stateflow objects

Reserved names:

- MATLAB keywords
- Reserved keywords for C, C++, and code generation. For complete list, see “Reserved Keywords” (Simulink Coder)
- `int8` , `uint8`
- `int16`, `uint16`
- `int32`, `uint32`
- `inf`, `Inf`
- `NaN`, `nan`
- `eps`
- `intmin`, `intmax`
- `realmin`, `realmax`
- `pi`
- `infinity`
- `Nil`

Note For some cases, the Model Advisor reports an issue in multiple subchecks of this check.

Available with Simulink Check.

Input Parameters

To specify the naming standard and model object names that the check flags, use the Model Advisor Configuration Editor.

1 Open the Model Configuration Editor and navigate to **Check model object names**. In the **Input Parameters** pane, for each of the model objects, select one of the following:

- MAAB to use the MAAB naming standard. When you select MAAB, the check uses the regular expression `(^.{32,}$)|([^a-zA-Z_0-9])|(^\d)|(^ )|(__)|(^_)|(_$)` to verify that names:
  - Use these characters: a-z, A-Z, 0-9, and the underscore (_).
  - Do not start with a number.
  - Do not use underscores at the beginning or end of a string.
  - Do not use more than one consecutive underscore.
  - Use strings that are less than 32 characters.
- Custom to use your own naming standard. When you select Custom, you can enter your own **Regular expression for prohibited *<model object>* names**. For example, if you want to allow more than one consecutive underscore, enter `(^.{32,}$)|([^a-zA-Z_0-9])|(^\d)|(^ )|(^_)|(_$)`
- None if you do not want the check to verify the model object name

2 Click **Apply**.

3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

Results and Recommended Actions

Condition	Recommended Action
The model object names do not comply with the naming standard specified in the input parameters.	Update the model object names to comply with your own guidelines or the high-integrity guidelines.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- “hisl_0032: Model object names” (Simulink)
- MAAB guideline, Version 3.0: jc_0201: Usable characters for Subsystem names

- MAAB guideline, Version 3.0: jc_0211: Usable characters for Inport blocks and Outport blocks
- MAAB guideline, Version 3.0: jc_0221: Usable characters for signal line names
- MAAB guideline, Version 3.0: jc_0231: Usable characters for block names
- MAAB guideline, Version 3.0: na_0030: Usable characters for Simulink Bus names

Check safety-related optimization settings

Check ID: `mathworks.do178.OptionSet`

Check model configuration for optimization settings that can impact safety.

Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Block reduction optimization is selected. This optimization can remove blocks from generated code, resulting in requirements without associated code and violations for traceability requirements. (See DO-331, Section MB.6.3.4.e—Source code is traceable to low-level requirements.)	Clear the **Block reduction** (Simulink) parameter in the Configuration Parameters dialog box or set parameter `BlockReduction` to `off`.
Implementation of logic signals as Boolean data is cleared. Strong data typing is recommended for safety-related code. (See DO-331, Section MB.6.3.1.e—High-level requirements conform to standards, DO-331, Section MB.6.3.2.e—Low-level requirements conform to standards, and MISRA C:2012, Rule 10.1.)	Select **Implement logic signals as boolean data (vs. double)** (Simulink) in the Configuration Parameters dialog box or set the parameter `BooleanDataType` to `on`.

Condition	Recommended Action
The model includes blocks that depend on elapsed or absolute time and is configured to minimize the amount of memory allocated for the timers. Such a configuration limits the number of days the application can execute before a timer overflow occurs. Many aerospace products are powered on continuously and timers should not assume a limited lifespan. (See DO-331, Section MB. 6.3.1.g—Algorithms are accurate and DO-331, Section MB.6.3.2.g—Algorithms are accurate.)	Set **Application lifespan (days)** (Simulink) on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `LifeSpan` to `inf`.
The optimization that suppresses the generation of initialization code for root-level inports and outports that are set to zero is selected. For safety-related code, you should explicitly initialize all variables. (See DO-331, Section MB.6.3.3.b—Software architecture is consistent.)	If you have an Embedded Coder® license, and you are using an ERT-based system target file, clear the **Remove root level I/O zero initialization** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `ZeroExternalMemoryAtStartup` to `on`. Alternatively, integrate external, hand-written code that initializes all I/O variables to zero explicitly.
The optimization that suppresses the generation of initialization code for internal work structures, such as block states and block outputs that are set to zero, is selected. For safety-related code, you should explicitly initialize every variable. (See DO-331, Section MB.6.3.3.b—Software architecture is consistent.)	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Remove internal data zero initialization** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `ZeroInternalMemoryAtStartup` to `on`. Alternatively, integrate external, hand-written code that initializes every state variable to zero explicitly.

Condition	Recommended Action
The optimization that suppresses generation of code resulting from floating-point to integer conversions that wrap out-of-range values is cleared. You must avoid overflows for safety-related code. When this optimization is off and your model includes blocks that disable the **Saturate on overflow** parameter, the code generator wraps out-of-range values for those blocks. This can result in unreachable and, therefore, untestable code. (See DO-331, Section MB. 6.3.1.g—Algorithms are accurate and DO-331, Section MB.6.3.2.g—Algorithms are accurate.)	If you have a Simulink Coder™ license, select **Remove code from floating-point to integer conversions that wraps out-of-range values** (Simulink) on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `EfficientFloat2IntCast` to `on`.
The optimization that suppresses generation of code that guards against division by zero for fixed-point data is selected. You must avoid division-by-zero exceptions in safety-related code. (See DO-331, Section MB.6.3.1.g—Algorithms are accurate, DO-331, Section MB.6.3.2.g—Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Remove code that protects against division arithmetic exceptions** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `NoFixptDivByZeroProtection` to `off`.
The optimization that uses the specified minimum and maximum values for signals and parameters to optimize the generated code is selected. This might result in requirements without traceable code. (See DO-331 Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Optimize using the specified minimum and maximum values** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box.

Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)” (Simulink)
- “hisl_0046: Configuration Parameters > Optimization > Block reduction” (Simulink)
- “hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)” (Simulink)
- “hisl_0052: Configuration Parameters > Optimization > Data initialization” (Simulink)
- “hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values” (Simulink)
- “hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions” (Simulink)

Check safety-related solver settings for tasking and sample-time

Check ID: `mathworks.do178.hisl_0042`

Check solver settings in the model configuration that apply to periodic sample time constraints and might impact safety.

Description

This check verifies that model configuration parameters are set optimally to ensure that the model operates at a specific set of prioritized periodic sample times for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver settings that select constraints on the sample times defined by the model is set to `Unconstrained` or `Ensure sample time independent`.	• In the Configuration Parameters dialog box, set “Periodic sample time constraint” (Simulink) or set the parameter `SampleTimeConstraint` to `Specified` and assign a value to **Sample time properties**. • If you use a referenced model as a reusable function, set “Periodic sample time constraint” (Simulink) to `Ensure sample time independent`.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.4.e Source code is traceable to low-level requirements
- hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
- “Periodic sample time constraint” (Simulink)

Check safety-related diagnostic settings for solvers

Check ID: `mathworks.do178.SolverDiagnosticsSet`

Check model configuration for diagnostic settings that apply to solvers and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic for detecting automatic breakage of algebraic loops is set to `none` or `warning`. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur. (See DO-331, Section MB. 6.3.3.e – Software architecture conforms to standards.)	Set **Algebraic loop** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `AlgebraicLoopMsg` to `error`. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.
The diagnostic for detecting automatic breakage of algebraic loops for Model blocks, atomic subsystems, and enabled subsystems is set to `none` or `warning`. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur. (See DO-331, Section MB.6.3.3.e – Software architecture conforms to standards.)	Set **Minimize algebraic loop** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `ArtificialAlgebraicLoopMsg` to `error`. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.
The diagnostic for detecting potential conflict in block execution order is set to `none` or `warning`. For safety-related applications, block execution order must be predictable. A model developer needs to know when conflicting block priorities exist. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Block priority violation** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `BlockPriorityViolationMsg` to `error`.

Condition	Recommended Action
The diagnostic for detecting whether a model contains an S-function that has not been specified explicitly to inherit sample time is set to `none` or `warning`. These settings can result in unpredictable behavior. A model developer needs to know when such an S-function exists in a model so it can be modified to produce predictable behavior. (See DO-331, Section MB.6.3.3.e – Software architecture conforms to standards.)	Set **Unspecified inheritability of sample time** (Simulink) in the Configuration Parameters dialog box or set parameter `UnknownTsInhSupMsg` to `error`.
The diagnostic for detecting whether the Simulink software automatically modifies the solver, step size, or simulation stop time is set to `none` or `warning`. Such changes can affect the operation of generated code. For safety-related applications, it is better to detect such changes so a model developer can explicitly set the parameters to known values. (See DO-331, Section MB.6.3.3.e – Software architecture conforms to standards.)	Set **Automatic solver parameter selection** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `SolverPrmCheckMsg` to `error`.
The diagnostic for detecting when a name is used for more than one state in the model is set to `none`. State names within a model should be unique. For safety-related applications, it is better to detect name clashes so a model developer can fix them. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **State name clash** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `StateNameClashWarn` to `warning`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.

- Does not allow exclusions of blocks or charts.

See Also

- "hisl_0043: Configuration Parameters > Diagnostics > Solver" (Simulink)
- "Model Configuration Parameters: Diagnostics" (Simulink)
- "View Diagnostics" (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

Check safety-related diagnostic settings for sample time

Check ID: `mathworks.do178.SampleTimeDiagnosticsSet`

Check model configuration for diagnostic settings that apply to sample time and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to sample times are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic for detecting when a source block, such as a Sine Wave block, inherits a sample time (specified as -1) is set to `none` or `warning`. The use of inherited sample times for a source block can result in unpredictable execution rates for the source block and blocks connected to it. For safety-related applications, source blocks should have explicit sample times to prevent incorrect execution sequencing. (See DO-331, Section MB.6.3.3.e – Software architecture conforms to standards.)	Set **Source block specifies -1 sample time** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `InheritedTslnSrcMsg` to `error`.

Condition	Recommended Action
The diagnostic for detecting invalid rate transitions between two blocks operating in multitasking mode is set to `none` or `warning`. Such rate transitions should not be used for embedded real-time code. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Multitask rate transition** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `MultiTaskRateTransMsg` to `error`.
The diagnostic for detecting subsystems that can cause data corruption or nondeterministic behavior is set to `none` or `warning`. This diagnostic detects whether conditionally executed multirate subsystems (enabled, triggered, or function-call subsystems) operate in multitasking mode. Such subsystems can corrupt data and behave unpredictably in real-time environments that allow preemption. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Multitask conditionally executed subsystem** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `MultiTaskCondExecSysMsg` to `error`.
The diagnostic for checking sample time consistency between a Signal Specification block and the connected destination block is set to `none` or `warning`. An over-specified sample time can result in an unpredictable execution rate. (See DO-331, Section MB. 6.3.3.e – Software architecture conforms to standards.)	Set **Enforce sample times specified by Signal Specification blocks** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `SigSpecEnsureSampleTimeMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to sample time and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- "Model Configuration Parameters: Sample Time Diagnostics" (Simulink)
- "View Diagnostics" (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- "hisl_0044: Configuration Parameters > Diagnostics > Sample Time" (Simulink)

Check safety-related diagnostic settings for signal data

Check ID: `mathworks.do178.DataValiditySignalsDiagnosticsSet`

Check model configuration for diagnostic settings that apply to signal data and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to signal data are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that specifies how the Simulink software resolves signals associated with `Simulink.Signal` objects is set to `Explicit and implicit` or `Explicit and warn implicit`. For safety-related applications, model developers should be required to define signal resolution explicitly. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Signal resolution** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalResolutionControl` to `Explicit only`. This provides predictable operation by requiring users to define each signal and block setting that must resolve to `Simulink.Signal` objects in the workspace. Alternatively, to disable the use of `Simulink.Signal` objects, set the configuration parameter to `None`.

Condition	Recommended Action
The Product block diagnostic that detects a singular matrix while inverting one of its inputs in matrix multiplication mode is set to `none` or `warning`. Division by a singular matrix can result in numeric exceptions when executing generated code. This is not acceptable in safety-related systems. (See DO-331, Section MB.6.3.1.g – Algorithms are accurate, DO-331, Section MB.6.3.2.g – Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set **Division by singular matrix** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `CheckMatrixSingularityMsg` to `error`.
The diagnostic that detects when the Simulink software cannot infer the data type of a signal during data type propagation is set to `none` or `warning`. For safety-related applications, model developers must verify the data types of signals. (See DO-331, Section MB.6.3.1.e – High-level requirements conform to standards, and DO-331, Section MB.6.3.2.e – Low-level requirements conform to standards.)	Set **Underspecified data types** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `UnderSpecifiedDataTypeMsg` to `error`.
The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type is set to `none` or `warning`. Undetected numeric overflows can result in unexpected application behavior. (See DO-331, Section MB.6.3.1.g – Algorithms are accurate, DO-331, Section MB.6.3.2.g – Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set **Wrap on overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `IntegerOverflowMsg` to `error`.
The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type, resulting in a saturation, is set to `none` or `warning`. Undetected numeric overflows can result in unexpected application behavior. (See DO-331, Section MB.6.3.1.g – Algorithms are accurate, DO-331, Section MB. 6.3.2.g – Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set **Saturate on overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `IntegerSaturationMsg` to `error`.

Condition	Recommended Action
The diagnostic that detects when the value of a block output signal is `Inf` or `NaN` at the current time step is set to `none` or `warning`. When this type of block output signal condition occurs, numeric exceptions can result, and numeric exceptions are not acceptable in safety-related applications. (See DO-331, Section MB.6.3.1.g – Algorithms are accurate, DO-331, Section MB. 6.3.2.g – Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set **Inf or NaN block output** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalInfNanChecking` to `error`.
The diagnostic that detects Simulink object names that begin with `rt` is set to `none` or `warning`. This diagnostic prevents name clashes with generated signal names that have an `rt` prefix. (See DO-331, Section MB.6.3.1.e – High-level requirements conform to standards, and DO-331, Section MB.6.3.2.e – Low-level requirements conform to standards.)	Set **"rt" prefix for identifiers** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `RTPrefix` to `error`.
The diagnostic that detects simulation range checking is set to `none` or `warning`. This diagnostic detects when signals exceed their specified ranges during simulation. Simulink compares the signal values that a block outputs with the specified range and the block data type. (See DO-331, Section MB.6.3.1.g – Algorithms are accurate, DO-331, Section MB. 6.3.2.g – Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set **Simulation range checking** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalRangeChecking` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal data and that can impact safety.

Capabilities and Limitations

- Does not run on library models.

- Does not allow exclusions of blocks or charts.

See Also

- "Model Configuration Parameters: Data Validity Diagnostics" (Simulink)
- "View Diagnostics" (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- "hisl_0005: Usage of Product blocks" (Simulink)

Check safety-related diagnostic settings for parameters

Check ID: `mathworks.do178.DataValidityParamDiagnosticsSet`

Check model configuration for diagnostic settings that apply to parameters and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to parameters are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a parameter downcast occurs is set to `none` or `warning`. A downcast to a lower signal range can result in numeric overflows of parameters, resulting in unexpected behavior.	Set **Detect downcast** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterDowncastMsg` to `error`.
The diagnostic that detects when a parameter underflow occurs is set to `none` or `warning`. When the data type of a parameter does not have enough resolution, the parameter value is zero instead of the specified value. This can lead to incorrect operation of generated code.	Set **Detect underflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterUnderflowMsg` to `error`.

Condition	Recommended Action
The diagnostic that detects when a parameter overflow occurs is set to `none` or `warning`. Numeric overflows can result in unexpected application behavior and should be detected and fixed in safety-related applications.	Set **Detect overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterOverflowMsg` to `error`.
The diagnostic that detects when a parameter loses precision is set to `none` or `warning`. Not detecting such errors can result in a parameter being set to an incorrect value in the generated code.	Set **Detect precision loss** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterPrecisionLossMsg` to `error`.
The diagnostic that detects when an expression with tunable variables is reduced to its numerical equivalent is set to `none` or `warning`. This can result in a tunable parameter unexpectedly not being tunable in generated code.	Set **Detect loss of tunability** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterTunabilityLossMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to parameters and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.1.g – Algorithms are accurate
  DO-331, Section MB.6.3.2.g – Algorithms are accurate
- "Model Configuration Parameters: Data Validity Diagnostics" (Simulink)
- "View Diagnostics" (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C, Software Considerations in Airborne Systems and Equipment Certification and related standards

- “hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters” (Simulink)

Check safety-related diagnostic settings for data used for debugging

Check ID: `mathworks.do178.DataValidityDebugDiagnosticsSet`

Check model configuration for diagnostic settings that apply to data used for debugging and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to debugging are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that enables model verification blocks is set to `Use local settings` or `Enable all`. Such blocks should be disabled because they are assertion blocks, which are for verification only. Model developers should not use assertions in embedded code.	In the Configuration Parameters dialog box, set **Model Verification block enabling** (Simulink) or set parameter `AssertControl` to `Disable All`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data used for debugging and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.1.e – High-level requirements conform to standards
- DO-331, Section MB.6.3.2.e – Low-level requirements conform to standards

- "Model Configuration Parameters: Data Validity Diagnostics" (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- "hisl_0305: Configuration Parameters > Diagnostics > Debugging" (Simulink)

Check safety-related diagnostic settings for data store memory

Check ID: `mathworks.do178.DataStoreMemoryDiagnosticsSet`

Check model configuration for diagnostic settings that apply to data store memory and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to data store memory are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether the model attempts to read data from a data store in which it has not stored data in the current time step is set to a value other than `Enable all as errors`. Reading data before it is written can result in use of stale data or data that is not initialized.	Set **Detect read before write** (Simulink) in the Configuration Parameters dialog box or set the parameter `ReadBeforeWriteMsg` to `Enable all as errors`.
The diagnostic that detects whether the model attempts to store data in a data store, after previously reading data from it in the current time step, is set to a value other than `Enable all as errors`. Writing data after it is read can result in use of stale or incorrect data.	Set **Detect write after read** (Simulink) in the Configuration Parameters dialog box or set the parameter `WriteAfterReadMsg` to `Enable all as errors`.

Condition	Recommended Action
The diagnostic that detects whether the model attempts to store data in a data store twice in succession in the current time step is set to a value other than `Enable all as errors`. Writing data twice in one time step can result in unpredictable data.	Set **Detect write after write** (Simulink) in the Configuration Parameters dialog box or set the parameter `WriteAfterWriteMsg` to `Enable all as errors`.
The diagnostic that detects when one task reads data from a Data Store Memory block to which another task writes data is set to `none` or `warning`. Reading or writing data in different tasks in multitask mode can result in corrupted or unpredictable data.	Set **Multitask data store** (Simulink) in the Configuration Parameters dialog box or set the parameter `MultiTaskDSMMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data store memory and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b – Software architecture is consistent
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0013: Usage of data store blocks” (Simulink)

Check safety-related diagnostic settings for type conversions

Check ID: `mathworks.do178.TypeConversionDiagnosticsSet`

Check model configuration for diagnostic settings that apply to type conversions and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to type conversions are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects Data Type Conversion blocks when the type conversion is set to `none`. The Simulink software might remove unnecessary Data Type Conversion blocks from generated code, which might result in requirements without corresponding code. The removal of these blocks needs to be identified so model developers can explicitly remove the unnecessary blocks .	Set the **Unnecessary type conversions** (Simulink) Configuration Parameter or`UnnecessaryDatatypeConvMsg` parameter to `warning`.
The diagnostic that detects vector-to-matrix or matrix-to-vector conversions at block inputs is set to `none` or `warning`. When the Simulink software automatically converts between vector and matrix dimensions, unintended operations or unpredictable behavior can occur.	Set the **Vector/matrix block input conversion** (Simulink) Configuration Parameter or `VectorMatrixConversionMsg` parameter to `error`
The diagnostic that detects when a 32-bit integer value is converted to a floating-point value is set to `none`. This type of conversion can result in a loss of precision due to truncation of the least significant bits for large integer values.	Set the **32-bit integer to single precision float conversion** (Simulink) Configuration Parameter or `Int32ToFloatConvMsg` parameter to `warning`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to type conversions and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.1.g – Algorithms are accurate
  DO-331, Section MB.6.3.2.g – Algorithms are accurate
- “Model Configuration Parameters: Type Conversion Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0309: Configuration Parameters > Diagnostics > Type Conversion” (Simulink)

Check safety-related diagnostic settings for signal connectivity

Check ID: `mathworks.do178.ConnectivitySignalsDiagnosticsSet`

Check model configuration for diagnostic settings that apply to signal connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to signal connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects virtual signals that have a common source signal but different labels is set to `none` or `warning`. This diagnostic pertains to virtual signals only and has no effect on generated code. However, signal label mismatches can lead to confusion during model reviews.	Set **Signal label mismatch** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `SignalLabelMismatchMsg` to `error`.

Condition	Recommended Action
The diagnostic that detects when the model contains a block with an unconnected input signal is set to `none` or `warning`. This must be detected because code is not generated for unconnected block inputs.	Set **Unconnected block input ports** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedInputMsg` to `error`.
The diagnostic that detects when the model contains a block with an unconnected output signal is set to `none` or `warning`. This must be detected because dead code can result from unconnected block output signals.	Set **Unconnected block output ports** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedOutputMsg` to `error`.
The diagnostic that detects unconnected signal lines and unmatched Goto or From blocks is set to `none` or `warning`. This error must be detected because code is not generated for unconnected lines.	Set **Unconnected line** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedLineMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.1.e – High-level requirements conform to standards
- DO-331, Section MB.6.3.2.e – Low-level requirements conform to standards
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- “Signal Basics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals” (Simulink)

Check safety-related diagnostic settings for bus connectivity

Check ID: `mathworks.do178.ConnectivityBussesDiagnosticsSet`

Check model configuration for diagnostic settings that apply to bus connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to bus connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a Model block's root Outport block is connected to a bus but does not specify a bus object is set to `none` or `warning`. For a bus signal to cross a model boundary, the signal must be defined as a bus object for compatibility with higher level models that use a model as a reference model.	Set **Unspecified bus object at root Outport block** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `RootOutportRequireBusObject` to `error`.
The diagnostic that detects whether the name of a bus element matches the name specified by the corresponding bus object is set to `none` or `warning`. This diagnostic prevents the use of incompatible buses in a bus-capable block such that the output names are inconsistent.	Set **Element name mismatch** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `BusObjectLabelMismatch` to `error`.
The diagnostic that detects when some blocks treat a signal as a mux/vector, while other blocks treat the signal as a bus, is set to `none` or `warning`. When the Simulink software automatically converts a muxed signal to a bus, it is possible for an unintended operation or unpredictable behavior to occur.	Set **Bus signal treated as vector** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box to `error`, or the parameter `StrictBusMsg` to `ErrorOnBusTreatedAsVector`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to bus connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b – Software architecture is consistent
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- `Simulink.Bus` in the Simulink reference documentation.
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses” (Simulink)

Check safety-related diagnostic settings that apply to function-call connectivity

Check ID: `mathworks.do178.FcnCallDiagnosticsSet`

Check model configuration for diagnostic settings that apply to function-call connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to function-call connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects incorrect use of a function-call subsystem is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Invalid function-call connection** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `InvalidFcnCallConMsg` to `error`.
The diagnostic that specifies whether the Simulink software has to compute inputs of a function-call subsystem directly or indirectly while executing the subsystem is set to `Use local settings` or `Disable all`. This diagnostic detects unpredictable data coupling between a function-call subsystem and the inputs of the subsystem in the generated code.	Set **Context-dependent inputs** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `FcnCallInpInsideContextMsg` to `Enable all as errors`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to function-call connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b – Software architecture is consistent
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls” (Simulink)

Check safety-related diagnostic settings for compatibility

Check ID: `mathworks.do178.CompatibilityDiagnosticsSet`

Check model configuration for diagnostic settings that affect compatibility and that might impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to compatibility are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a block has not been upgraded to use features of the current release is set to `none` or `warning`. An S-function written for an earlier version might not be compatible with the current version and generated code could operate incorrectly.	Set **S-function upgrades needed** (Simulink) on the **Diagnostics** > **Compatibility** pane in the Configuration Parameters dialog box or set the parameter `SFcnCompatibilityMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that affect compatibility and that might impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b – Software architecture is consistent
- “View Diagnostics” (Simulink)
- “Model Configuration Parameters: Compatibility Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0301: Configuration Parameters > Diagnostics > Compatibility” (Simulink)

Check safety-related diagnostic settings for model initialization

Check ID: `mathworks.do178.InitDiagnosticsSet`

In the model configuration, check diagnostic settings that affect model initialization and might impact safety.

Description

This check verifies that model diagnostic configuration parameters for initialization are optimally set to generate code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. The “Check undefined subsystem initial output” (Simulink) diagnostic is cleared. This diagnostic specifies whether Simulink displays a warning if the model contains a conditionally executed subsystem, in which a block with a specified initial condition drives an Outport block with an undefined initial condition. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic` and select **Check undefined subsystem initial output** (Simulink). • Set the parameter `CheckSSInitialOutputMsg` to `on`.

Condition	Recommended Action
In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. This diagnostic detects potential initial output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic`. • Set the parameter `CheckExecutionContextPreStartOutputMsg` to `on`.
In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. The “Check runtime output of execution context” (Simulink) diagnostic is cleared. This diagnostic detects potential output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized and feeds into a block with a tunable parameter. If undetected, this condition can cause the behavior of the downstream block to be nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic` and select **Check runtime output of execution context** (Simulink). • Set the parameter `CheckExecutionContextRuntimeOutputMsg` to `on`.

Action Results

To configure the diagnostic settings that affect model initialization and might impact safety, click **Modify Settings**.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b – Software architecture is consistent
- “View Diagnostics” (Simulink)
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- “hisl_0304: Configuration Parameters > Diagnostics > Model initialization” (Simulink)

Check safety-related diagnostic settings for model referencing

Check ID: `mathworks.do178.MdlrefDiagnosticsSet`

Check model configuration for diagnostic settings that apply to model referencing and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects a mismatch between the version of the model that creates or refreshes a Model block and the current version of the referenced model is set to `error` or `warning`. The detection occurs during load and update operations. When you get the latest version of the referenced model from the software configuration management system, rather than an older version that was used in a previous simulation, if this diagnostic is set to `error`, the simulation is aborted. If the diagnostic is set to `warning`, a warning message is issued. To resolve the issue, the user must resave the model being simulated, which may not be the desired action. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Model block version mismatch** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceVersionMismatchMessage` to `none`.
The diagnostic that detects port and parameter mismatches during model loading and updating is set to `none` or `warning`. If undetected, such mismatches can lead to incorrect simulation results because the parent and referenced models have different interfaces. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Port and parameter mismatch** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceIOMismatchMessage` to `error`.

Condition	Recommended Action
The diagnostic that detects invalid internal connections to the current model's root-level Inport and Outport blocks is set to `none` or `warning`. When this condition is detected, the Simulink software might automatically insert hidden blocks into the model to fix the condition. The hidden blocks can result in generated code without traceable requirements. Setting the diagnostic to `error` forces model developers to fix the referenced models manually. (See DO-331, Section MB. 6.3.3.b – Software architecture is consistent.)	Set **Invalid root Inport/Outport block connection** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceIOMessage` to `error`.
The diagnostic that detects whether To Workspace or Scope blocks are logging data in a referenced model is set to `none` or `warning`. Data logging is not supported for To Workspace and Scope blocks in referenced models. (See DO-331, Section MB.6.3.1.d – High-level requirements are verifiable and DO-331, Section MB.6.3.2.d – Low-level requirements are verifiable.)	Set **Unsupported data logging** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceDataLoggingMessage` to `error`. To log data, remove the blocks and log the referenced model signals. For more information, see “Logging Referenced Model Signals” (Simulink).

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to model referencing and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- “View Diagnostics” (Simulink)
- “Model Configuration Parameters: Model Referencing Diagnostics” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

- "Logging Referenced Model Signals" (Simulink)
- "hisl_0310: Configuration Parameters > Diagnostics > Model Referencing" (Simulink)

Check safety-related model referencing settings

Check ID: `mathworks.do178.MdlrefOptSet`

Check model configuration for model referencing settings that can impact safety.

Description

This check verifies that model configuration parameters for model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The referenced model is configured such that its target is rebuilt whenever you update, simulate, or generate code for the model, or if the Simulink software detects changes in known dependencies. These configuration settings can result in unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references. (See DO-331, Section MB.6.3.1.b – High-level requirements are accurate and consistent and DO-331, Section MB.6.3.2.b – Low-level requirements are accurate and consistent.)	Set **Rebuild** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `UpdateModelReferenceTargets` to `Never` or `If any changes detected`.

Condition	Recommended Action
The diagnostic that detects whether a target needs to be rebuilt is set to `None` or `Warn if targets require rebuild`. For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent. This diagnostic parameter is available only if **Rebuild** is set to `Never`. (See DO-331, Section MB.6.3.1.b – High-level requirements are accurate and consistent and DO-331, Section MB.6.3.2.b – Low-level requirements are accurate and consistent.)	Set **Never rebuild diagnostic** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `CheckModelReferenceTargetMessage` to `error`.
The ability to pass scalar root input by value is off. This capability should be off because scalar values can change during a time step and result in unpredictable data. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	Set **Pass fixed-size scalar root inputs by value for Real-Time Workshop** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferencePassRootInputsByReference` to `off`.
The model is configured to minimize algebraic loop occurrences. This configuration is incompatible with the recommended setting of **Single output/update function** for embedded systems code. (See DO-331, Section MB.6.3.3.b – Software architecture is consistent.)	In the Configuration Parameters dialog box, set **Minimize algebraic loop occurrences** (Simulink) or set parameter `ModelReferenceMinAlgLoopOccurrences` to `off`.

Action Results

Clicking **Modify Settings** configures model referencing settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- “hisl_0037: Configuration Parameters > Model Referencing” (Simulink)
- “Analyze Model Dependencies” (Simulink)
- “Model Configuration Parameters: Model Referencing” (Simulink)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

Check safety-related code generation settings

Check ID: `mathworks.do178.CodeSet`

Check model configuration for code generation settings that can impact safety.

Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The option to include comments in the generated code is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Select **Include comments** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `GenerateComments` to `on`.
The option to include comments that describe the code for blocks is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB. 6.3.4.e – Source code is traceable to low-level requirements.)	Select **Simulink block comments** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `SimulinkBlockComments` to `on`.

Condition	Recommended Action
The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Select **Show eliminated blocks** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `ShowEliminatedStatement` to `on`.
The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in *model*`_prm.h` is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Select **Verbose comments for SimulinkGlobal storage class** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `ForceParamTrailComments` to `on`.
The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Select **Requirements in block comments** (Simulink Coder) on the **Code Generation** > **Custom comments** pane in the Configuration Parameters dialog box or set the parameter `ReqsInCode` to `on`.
The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems. (See DO-331, Section MB. 6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Support: non-finite numbers** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportNonFinite` to `off`.

Condition	Recommended Action
The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB. 6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Support: absolute time** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportAbsoluteTime` to `off`.
The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Support: continuous time** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportContinuousTime` to `off`.
The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB. 6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Support: non-inlined S-functions** (Simulink Coder) in the Configuration Parameters dialog box or set the parameter `SupportNonInlinedSFcns` to `off`.
The option to generate model function calls compatible with the main program module of the pre-R2012a GRT target is selected. This option is inappropriate for real-time safety-related systems. (See DO-331, Section MB. 6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Classic call call interface** (Simulink Coder) on the **Code Generation** > **Interface**pane in the Configuration Parameters dialog box or set the parameter `GRTInterface` to `off`.

Condition	Recommended Action
The option to generate the *model*_update function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code. (See DO-331, Section MB. 6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer.)	Select **Single output/update function** (Simulink Coder) on the **Code Generation** > **Interface**pane in the Configuration Parameters dialog box or set the parameter `CombineOutputUpdateFcns` to `on`.
The option to generate the *model*_terminate function is selected. This function deallocates dynamic memory, which is unsuitable for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB. 6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **Terminate function** (Simulink Coder) on the **Code Generation** pane in the Configuration Parameters dialog box or set the parameter `IncludeMdlTerminateFcn` to `off`.
The option to log or monitor error status is cleared. If you do not select this option, the Simulink Coder product generates extra code that might not be reachable for testing. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer.)	Select **Remove error status field in real-time model data structure** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SuppressErrorStatus` to `on`.
MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer and DO-331, Section MB. 6.3.2.c – Low-level requirements are compatible with target computer.)	Clear **MAT-file logging** (Simulink Coder) in the Configuration Parameters dialog box or set the parameter `MatFileLogging` to `off`.

Condition	Recommended Action
The option that specifies the style for parenthesis usage is set to `Minimum (Rely on C/C++ operators precedence)` or to `Nominal (Optimize for readability)`. For safety-related applications, explicitly specify precedence with parentheses. (See DO-331, Section MB.6.3.1.c – High-level requirements are compatible with target computer, DO-331, Section MB.6.3.2.c – Low-level requirements are compatible with target computer, and MISRA C:2012, Rule 12.1.)	Set parameter `ParenthesesLevel` to `Maximum (Specify precedence with parentheses)`.
The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Set parameter `PreserveExpressionOrder` to `on`.
The option that specifies whether to preserve empty primary condition expressions in `if` statements is cleared. This option increases the traceability of the generated code. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Set parameter `PreserveIfCondition` to `on`.
The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews. (See DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements.)	Set **Minimum mangle length** (Simulink Coder) on the **Code Generation** > **Symbols** pane in the Configuration Parameters dialog box or the parameter `MangleLength` to a value of 4 or greater.

Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- “hisl_0038: Configuration Parameters > Code Generation > Comments” (Simulink)
- “hisl_0039: Configuration Parameters > Code Generation > Interface” (Simulink)
- “hisl_0047: Configuration Parameters > Code Generation > Code Style” (Simulink)
- “hisl_0049: Configuration Parameters > Code Generation > Symbols” (Simulink)
- “Model Configuration Parameters: Code Generation Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Symbols” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Code Style” (Embedded Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

Check safety-related optimization settings for Loop unrolling threshold

Check ID: `mathworks.do178.hisl_0051`

Check optimization settings in the model configuration that apply to Loop unrolling threshold and might impact safety.

Description

This check verifies that the model optimization configuration parameters pertaining to the minimum signal or parameter width for which a `for` loop is generated is set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The optimization setting to specify the minimum signal or parameter width for which a `for` loop is generated is set to a value less than `2`.	In the Configuration Parameters dialog box, set “Loop unrolling threshold” (Simulink) or set the parameter `RollThreshold` to a value equal to or greater than `2`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- DO-331 Section MB.6.3.4.e—Source code is traceable to low-level requirements.

  MISRA C:2012, Rule 6.1
- “Loop unrolling threshold” (Simulink)
- “hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold” (Simulink)

Check safety-related diagnostic settings for saving

Check ID: `mathworks.do178.SavingDiagnosticsSet`

Check model configuration for diagnostic settings that apply to saving model files

Description

This check verifies that model configuration parameters are set optimally for saving a model for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains disabled library links before the model is saved is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Block diagram contains disabled library links** (Simulink) in the Configuration Parameters dialog box or set parameter `SaveWithDisabledLinkMsg` to `error`.
The diagnostic that detects whether a model contains library links that are using parameters not in a mask before the model is saved is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Block diagram contains parameterized library links** (Simulink) in the Configuration Parameters dialog box or set parameter `SaveWithParameterizedLinkMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to saving a model file.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.3.b - Software architecture is consistent
- “hisl_0036: Configuration Parameters > Diagnostics > Saving” (Simulink)
- “Identify disabled library links” (Simulink)
- “Save a Model” (Simulink)

- “Model Parameters” (Simulink)

Check safety-related diagnostic settings for Merge blocks

Check ID: `mathworks.do178.hisl_0303`

Check model configuration for diagnostic settings that apply to Merge blocks

Description

This check verifies that model configuration parameters are set optimally for Merge blocks for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains Merge blocks with more than one driving block executing at the same time step is set to `none` or `warning` and in the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`.	In the Configuration Parameters dialog box, set “Detect multiple driving blocks executing at the same time step” (Simulink) or set parameter `MergeDetectMultiDrivingBlocksExec` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331 MB.6.3.2 (b) Accuracy and Consistency
- “hisl_0303: Configuration Parameters > Diagnostics > Merge block” (Simulink)

- “Detect multiple driving blocks executing at the same time step” (Simulink)
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)

Check safety-related diagnostic settings for Stateflow

Check ID: `mathworks.do178.hisl_0311`

Check safety-related diagnostic settings for Stateflow

Description

This check verifies that model configuration parameters are set optimally for Stateflow for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a chart configuration leads to unwanted backtracking during simulation is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Unexpected backtracking” (Simulink) or set the parameter `SFUnexpectedBacktrackingDiag` to `error`.
The diagnostic that detects whether a chart configuration has blocks that connect to chart input ports do not initialize their outputs during initialization is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Invalid input data access in chart initialization” (Simulink) or set the parameter `SFInvalidInputDataAccessInChartInitDiag` to `error`.
The diagnostic that detects whether a chart has an unconditional default transition to a state or a junction is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “No unconditional default transitions” (Simulink) or set the parameter `SFNoUnconditionalDefaultTransitionDiag` to `error`.
The diagnostic that detects whether a chart contains a transition that loops outside of the parent state or junction is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Transition outside natural parent” (Simulink) or set the parameter `SFTransitionOutsideNaturalParentDiag` to `error`.

Condition	Recommended Action
The diagnostic that detects whether a chart constructs on a valid execution path is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Unreachable execution path” (Simulink) or set the parameter `SFUnreachableExecutionPathDiag` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards'
  DO-331, Section MB.6.3.1.g 'Algorithms are accurate'
  DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
  DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable'
  DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'
  DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
- “hisl_0311: Configuration Parameters > Diagnostics > Stateflow” (Simulink)
- “Diagnostics Parameters: Stateflow” (Simulink)

Check for model elements that do not link to requirements

Check ID: `mathworks.do178.RequirementInfo`

Check whether Simulink model elements link to a requirements document.

Description

This check verifies whether model objects link to a document containing engineering requirements for traceability.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Blocks do not link to a requirements document.	Link to requirements document. See “Link to Requirements Document Using Selection-Based Linking” (Simulink Requirements).

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.
- Does not allow the exclusion of Stateflow elements.

Tip

Run this check from the top model or subsystem that you want to check.

See Also

- DO-331, Section MB.6.3.1.f - High-level requirements trace to system requirements
- DO-331, Section MB.6.3.2.f - Low-level requirements trace to high-level requirements
- hisl_0070: Placement of requirement links in a model
- “Requirements Traceability” (Simulink)
- “Requirements Traceability in Simulink” (Simulink)
- “Requirements Traceability Links” (Simulink Requirements)
- “Find Model Elements in Simulink Models” (Simulink)

Check state machine type of Stateflow charts

Check ID: `mathworks.do178.hisf_0001`

Identify whether Stateflow charts are all Mealy or all Moore charts.

Description

Compares the state machine type of all Stateflow charts to the type that you specify in the input parameters.

Available with Simulink Check.

Input Parameters

Mealy or Moore

Check whether charts use the same state machine type, and are all Mealy or all Moore charts.

Mealy

Check whether all charts are Mealy charts.

Moore

Check whether all charts are Moore charts.

Results and Recommended Actions

Condition	Recommended Action
The input parameter is set to `Mealy or More` and charts in the model use either of the following: • Classic state machine types. • Multiple state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to either `Mealy` or `Moore`. Use the same state machine type for all charts in the model.
The input parameter is set to `Mealy` and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to `Mealy`.
The input parameter is set to `Moore` and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to `Moore`.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.

See Also

- DO-331, Section MB.6.3.1.b - High-level requirements are accurate and consistent
- DO-331, Section MB.6.3.1.e - High-level requirements conform to standards
- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.e - Low-level requirements conform to standards
- DO-331, Section MB.6.3.3.b - Software architecture is consistent
- DO-331, Section MB.6.3.3.e - Software architecture conform to standards
- “hisf_0001: Mealy and Moore semantics” (Simulink)
- “Overview of Mealy and Moore Machines” (Stateflow)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow charts for ordering of states and transitions

Check ID: `mathworks.do178.hisf_0002`

Identify Stateflow charts that have **User specified state/transition execution order** cleared.

Description

Identify Stateflow charts that have **User specified state/transition execution order** cleared, and therefore do not use explicit ordering of parallel states and transitions.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Stateflow charts have **User specified state/transition execution order** cleared.	For the specified charts, in the Chart Properties dialog box, select **User specified state/transition execution order**.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** selects **User specified state/transition execution order** for the specified charts.

See Also

- DO-331, Section MB.6.3.3.b 'Software architecture is consistent'
  DO-331, Section MB.6.3.3.e 'Software architecture conform to standards '
- “hisf_0002: User-specified state/transition execution order” (Simulink)

  “Transition Testing Order in Multilevel State Hierarchy” (Stateflow)
- “Execution Order for Parallel States” (Stateflow)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow debugging options

Check ID: `mathworks.do178.hisf_0011`

Check the Stateflow debugging settings.

Description

Verify the following debugging settings.

- **Wrap on overflow**
- **Simulation range checking**
- **Detect Cycles**

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Any of the following: • **Wrap on overflow** is not set to `error`. • **Simulation range checking** is not set to `error`. • **Detect Cycles** is cleared.	In the Configuration Parameters dialog box, set: • Wrap on overflow to `error`. • Simulation range checking to `error`. In the model window, select: • **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Detect Cycles**.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** selects the specified debugging options.

See Also

- DO-331, Section MB.6.3.1.b - High-level requirements are accurate and consistent
- DO-331, Section MB.6.3.1.e - High-level requirements conform to standards
- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.e - Low-level requirements conform to standards
- “hisf_0011: Stateflow debugging settings” (Simulink)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow charts for transition paths that cross parallel state boundaries

Check ID: `mathworks.do178.hisf_0013`

Identify transition paths that cross parallel state boundaries in Stateflow charts.

Description

Identify transition paths that cross parallel state boundaries in Stateflow charts. Using such transition paths creates diagrams that consist of transition executions, which are difficult to understand.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow charts have transition paths that cross parallel state boundaries.	Modify the Stateflow charts so that transitions do not cross parallel state boundaries. For more information see, “Defining Transitions Between States” (Stateflow).

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.
- Analyzes content in all masked subsystems.

See Also

- DO-331, Section MB.6.3.1.b High-level requirements are accurate and consistent
- DO-331, Section MB.6.3.1.e High-level requirements conform to standards
- DO-331, Section MB.6.3.2.b Low-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.e Low-level requirements conform to standards
- “hisf_0013: Usage of transition paths (crossing parallel state boundaries)” (Simulink)

- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)
- “Process for Entering, Executing, and Exiting States” (Stateflow)

Check Stateflow charts for strong data typing

Check ID: `mathworks.do178.hisf_0015`

Identify variables and parameters in expressions with different data types in Stateflow objects.

Description

To facilitate strong data typing, this check identifies the variables and parameters in expressions with different data types in Stateflow states and transitions.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow objects have variables and parameters in expressions with different data types.	Explicitly cast variables and parameters in expressions to the same data types. For more information see, `cast`.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.
- Analyzes content in all masked subsystems.
- Does not analyze the type of literals in expressions in Stateflow objects. Explicitly casts types of literals to the intended data type.
- Does not flag expressions with true and false keywords. For more information, see “Reserved Keywords for Code Generation” (Embedded Coder).

See Also

- DO-331, Section MB.6.3.1.b High-level requirements are accurate and consistent

- DO-331, Section MB.6.3.1.e High-level requirements conform to standards
- DO-331, Section MB.6.3.1.g Algorithms are accurate
- DO-331, Section MB.6.3.2.b Low-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.e Low-level requirements conform to standards
- DO-331, Section MB.6.3.2.g Algorithms are accurate
- “hisf_0015: Strong data typing (casting variables and parameters in expressions)” (Simulink)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)
- “Use Data Types in Stateflow” (Stateflow)

Check usage of lookup table blocks

Check ID: `mathworks.do178.LUTRangeCheckCode`

Check for lookup table blocks that do not generate out-of-range checking code.

Description

This check verifies that the following blocks generate code to protect against inputs that fall outside the range of valid breakpoint values:

- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table
- Prelookup

This check also verifies that Interpolation Using Prelookup blocks generate code to protect against inputs that fall outside the range of valid index values.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The lookup table block does not generate out-of-range checking code.	Change the setting on the block dialog box so that out-of-range checking code is generated. • For the 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, and Prelookup blocks, clear the check box for **Remove protection against out-of-range input in generated code**. • For the Interpolation Using Prelookup block, clear the check box for **Remove protection against out-of-range index in generated code**.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** verifies that lookup table blocks are set to generate out-of-range checking code.

See Also

- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- “hisl_0033: Usage of Lookup Table blocks” (Simulink)
- n-D Lookup Table block
- Prelookup block
- Interpolation Using Prelookup block

Check MATLAB Code Analyzer messages

Check ID: `mathworks.do178.himl_0004`

Check MATLAB Functions for `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs.

Description

Verifies `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs for:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
For MATLAB code in MATLAB Function blocks, either of the following: • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.
For MATLAB functions defined in Stateflow charts, either of the following: • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.

Condition	Recommended Action
For called MATLAB functions: • Code does not have the `%#codegen` directive. • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Insert `%#codegen` directive in the MATLAB code. • Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Sections MB.6.3.1.b and MB.6.3.2.b - Accuracy and consistency
- "Check Code for Errors and Warnings" (MATLAB)
- "himl_0004: MATLAB Code Analyzer recommendations for code generation" (Simulink)

Check MATLAB code for global variables

Check ID: `mathworks.do178.himl_0005`

Check for global variables in MATLAB code.

Description

Verifies that global variables are not used in any of the following:

- MATLAB code in MATLAB Function blocks

- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Global variables are used in one or more of the following: • MATLAB code in MATLAB Function blocks • MATLAB functions defined in Stateflow charts • Called MATLAB functions	Replace global variables with signal lines, function arguments, or persistent data.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Sections MB.6.3.3.b ‘Consistency’
- “himl_0005: Usage of global variables in MATLAB functions” (Simulink)

Check for inconsistent vector indexing methods

Check ID: `mathworks.do178.hisl_0021`

Identify blocks with inconsistent indexing method.

Description

Using inconsistent block indexing methods can result in modeling errors. You should use a consistent vector indexing method for all blocks. This check identifies blocks with inconsistent indexing methods. The indexing methods are zero-based, one-based or user-specified.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks with inconsistent indexing methods. The indexing methods are zero-based, one-based or user-specified.	Modify the model to use a single consistent indexing method.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- “hisl_0021: Consistent vector indexing method” (Simulink)

Check for MATLAB Function interfaces with inherited properties

Check ID: `mathworks.do178.himl_0002`

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Functions identified in the results. If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink): • Change **Complexity** from `Inherited` to `On` or `Off`. • Change **Type** from `Inherit: Same as Simulink` to an explicit type.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- “himl_0002: Strong data typing at MATLAB function boundaries” (Simulink)

Check MATLAB Function metrics

Check ID: `mathworks.do178.himl_0003`

Display complexity and code metrics for MATLAB Functions. Report metric violations.

Description

This check provides complexity and code metrics for MATLAB Functions. The check additionally reports metric violations. A results table provides links to MATLAB Functions that violate the complexity input parameters.

Available with Simulink Check.

Input Parameters

Maximum effective lines of code per function

Provide the maximum effective lines of code per function. Effective lines do not include empty lines, comment lines, or lines with a function `end` keyword.

Minimum density of comments

Provide minimum density of comments. Density is ratio of comment lines to total lines of code.

Maximum cyclomatic complexity per function

Provide maximum cyclomatic complexity per function. Cyclomatic complexity is the number of linearly independent paths through the source code.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Function violates the complexity input parameters.	For the MATLAB Function: • If effective lines of code is too high, further divide the MATLAB Function. • If comment density is too low, add comment lines. • If cyclomatic complexity per function is too high, further divide the MATLAB Function.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.

See Also

- DO-331, Sections MB.6.3.1.e - High-level requirements conform to standards
- DO-331, Sections MB.6.3.2.e - Low-level requirements conform to standards
- “himl_0003: Limitation of MATLAB function complexity” (Simulink)

Check for blocks not recommended for C/C++ production code deployment

Check ID: `mathworks.do178.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

Description

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder. If you are using blocks with support notes for code generation, review the information and follow the given advice.

Available with Simulink Check and Embedded Coder.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- “hisl_0020: Blocks not recommended for MISRA C:2012 compliance” (Simulink)
- “Blocks and Products Supported for C Code Generation” (Simulink Coder)

Check for variant blocks with 'Generate preprocessor conditionals' active

Check ID: `mathworks.do178.VariantBlock`

Check variant block parameters for settings that might result in code that does not trace to requirements.

Description

This check verifies that variant block parameters for code generation are set to trace to requirements.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The option to generate preprocessor conditionals is selected in one or more variant blocks in the model.	In order to simplify the tracing of code to requirements, consider clearing the option to generate preprocessor conditionals in variant blocks.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'

- “hisl_0023: Verification of model and subsystem variants” (Simulink)

Check Stateflow charts for uniquely defined data objects

Check ID: `mathworks.do178.hisl_0061`

Identify Stateflow charts that include data objects that are not uniquely defined.

Description

This check searches your model for local data in Stateflow charts that is not uniquely defined.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart contains a data object identifier defined in two or more scopes.	For the identified chart, do one of the following: • Create a unique data object identifier within each of the scopes. • Create a unique data object identifier within the chart, at the parent level.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- “hisl_0061: Unique identifiers for clarity” (Simulink)

Check usage of Math Operations blocks

Check ID: `mathworks.do178.MathOperationsBlocksUsage`

Identify usage of Math Operation blocks that might impact safety.

Description

This check inspects the usage of the following blocks:

- Abs
- Gain
- Math Function
  - Natural logarithm
  - Common (base 10) logarithm
  - Remainder after division
  - Reciprocal
- Assignment

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains an Absolute Value block that is operating on one of the following: • A boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code • A signed integer value with the **Saturate on integer overflow** check box not selected. For signed data types, the absolute value of the most negative value is problematic because it is not representable by the data type. This condition results in an overflow in the generated code.	If the identified Absolute Value block is operating on a boolean or unsigned data type, do one of the following: • Change the input of the Absolute Value block to a signed input type. • Remove the Absolute Value block from the model. If the identified Absolute Value block is operating on a signed data type, in the **Block Parameters > Signal Attributes** dialog box, select **Saturate on integer overflow**.
The model or subsystem contains Gain blocks with a of value 1 or an identity matrix.	If you are using Gain blocks as buffers, consider replacing them with Signal Conversion blocks.
The model or subsystem contains Math Function - Natural logarithm `(log)` blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a `log` function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a `NaN` or `-Inf` and result in a run-time error in the generated code.
The model or subsystem contains Math Function - Common (base 10) `(base 10 logarithm)` blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a `log10` function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a `NaN` or `-Inf` and result in a run-time error in the generated code.

Condition	Recommended Action
The model or subsystem contains Math Function - Remainder after division `(rem)` blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a `rem` function, protect the second input to the block from being equal to zero. Otherwise the output can produce a `Inf` or `-Inf` and result in a run-time error in the generated code.
The model or subsystem contains Math Function - Reciprocal `(reciprocal)` blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a `reciprocal` function, protect the input to the block from being equal to zero. Otherwise the output can produce a `Inf` or `-Inf` and result in a run-time error in the generated code.
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.	Set block parameter **Action if any output element is not assigned** to one of the recommended values: • **Error**, if Assignment block is not in an Iterator subsystem. • **Warning**, if Assignment block is in an Iterator subsystem.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- DO-331 Section MB.6.3.1.d – High-level requirements are verifiable
- DO-331 Section MB.6.3.2.d – Low-level requirements are verifiable
- MISRA C:2012, Dir 4.1
- MISRA C:2012, Rule 9.1
- “hisl_0001: Usage of Abs block” (Simulink)

- “hisl_0002: Usage of Math Function blocks (rem and reciprocal)” (Simulink)
- “hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)” (Simulink)
- “hisl_0029: Usage of Assignment blocks” (Simulink)

Check usage of Signal Routing blocks

Check ID: `mathworks.do178.SignalRoutingBlockUsage`

Identify usage of Signal Routing blocks that might impact safety.

Description

This check identifies model or subsystem Switch blocks that might generate code with inequality operations (~=) in expressions that contain a floating-point variable or constant.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a Switch block that might generate code with inequality operations (~=) in expressions where at least one side of the expression contains a floating-point variable or constant. The Switch block might cause floating-point inequality comparisons in the generated code.	For the identified block, do one of the following: • For the control input block, change the **Data type** parameter setting. • Change the Switch block **Criteria for passing first input** parameter setting. This might change the algorithm.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- MISRA C:2012, Dir 1.1
- “hisl_0034: Usage of Signal Routing blocks” (Simulink)

Check usage of Logic and Bit Operations blocks

Check ID: `mathworks.do178.LogicBlockUsage`

Identify usage of Logical Operator and Bit Operations blocks that might impact safety.

Description

This check inspects the usage of:

- Blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare To Zero, and Detect Change blocks
- Logical Operator blocks

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.	For the identified blocks, use common data types as inputs. You can use Data Type Conversion blocks to change input data types.
The model or subsystem contains a block computing a relational operator that does not have Boolean output. The condition can lead to unpredictable results in the generated code.	For the specified blocks, on the Block Parameters > Signal Attributes pane, set the **Output data type** to `boolean`.

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues. These operators can lead to unpredictable results in the generated code.	For the identified block, do one of the following: • Change the signal data type. • Rework the model to eliminate using == or ~= operators on floating-point signals.
The model or subsystem contains a Logical Operator block that has inputs or outputs that are not Boolean inputs or outputs. The block might result in floating-point equality or inequality comparisons in the generated code.	• Modify the Logical Operator block so that all inputs and outputs are Boolean. On the Block Parameters > Signal Attributes pane, consider selecting **Require all inputs to have the same data type** and setting **Output data type** to `boolean`. • In the Configuration Parameters dialog box, consider selecting the **Implement logic signals as boolean data (vs. double)**.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- MISRA C:2012, Dir 1.1
- MISRA C:2012, Rule 10.1
- “hisl_0016: Usage of blocks that compute relational operators” (Simulink)
- “hisl_0017: Usage of blocks that compute relational operators (2)” (Simulink)

- “hisl_0018: Usage of Logical Operator block” (Simulink)

Check usage of Ports and Subsystems blocks

Check ID: `mathworks.do178.PortsSubsystemsUsage`

Identify usage of Ports and Subsystems blocks that might impact safety.

Description

This check inspects the usage of these blocks:

- For Iterator
- While Iterator
- If
- Switch Case

The check does not flag Switch Case blocks that do not use integer data types or enumeration values for inputs. To comply with “hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks” (Simulink) – C, use an integer data type or an enumeration value for the inputs to Switch Case blocks.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.	For the identified For Iterator blocks, do one of the following: • Set the **Iteration limit source** parameter to `internal`. • If the **Iteration limit source** parameter must be `external`, use a Constant, Probe, or Width block as the source. • Clear the Set next **i (iteration variable) externally** check box. • Consider selecting the **Show iteration variable** check box and observe the iteration value during simulation.
The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code.	For the identified While Iterator blocks: • Set the **Maximum number of iterations (-1 for unlimited)** parameter to a positive integer value. • Consider selecting the **Show iteration number port** check box and observe the iteration value during simulation.
The model or subsystem contains an If block with an If expression or Elseif expressions that might cause floating-point equality or inequality comparisons in generated code.	Modify the expressions in the If block to avoid floating-point equality or inequality comparisons in generated code.
The model or subsystem contains an If block using Elseif expressions without an Else condition.	In the If block Block Parameters dialog box, select **Show else condition**. Connect the resulting Else output port to an If Action Subsystem block.
The model or subsystem contains an If block with output ports that do not connect to If Action Subsystem blocks.	Verify that output ports of the If block connect to If Action Subsystem blocks.

Condition	Recommended Action
The model or subsystem contains an Switch Case block without a default case.	In the Switch Case block Block Parameters dialog box, select **Show default case**. Connect the resulting default output port to a Switch Case Action Subsystem block.
The model or subsystem contains a Switch Case block with an output port that does not connect to a Switch Case Action Subsystem block.	Verify that output ports of the Switch Case blocks connect to Switch Case Action Subsystem blocks.
The model or subsystem contains one of the following time-dependent blocks in a For Iterator or While Iterator subsystem: • Discrete Filter • Discrete FIR Filter • Discrete State-Space • Discrete Transfer Fcn • Discrete Zero-Pole • Transfer Fcn First Order • Transfer Fcn Lead or Lag • Transfer Fnc Real Zero • Discrete Derivative • Discrete Transfer Fcn (with initial outputs) • Discrete Transfer Fcn (with initial states) • Discrete Zero-Pole (with initial outputs) • Discrete Zero-Pole (with initial states)	In the model or subsystem, consider removing the time-dependent blocks.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.

See Also

- DO-331, Section MB.6.3.3.b—Software architecture is consistent
- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- DO-331, Section MB.6.3.1.e – High-level requirements conform to standards
- DO-331, Section MB.6.3.2.e – Low-level requirements conform to standards
- DO-331, Section MB.6.3.1.b - High-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- MISRA C:2012, Rule 14.2
- MISRA C:2012, Rule 16.4
- MISRA C:2012, Dir 4.1
- "hisl_0006: Usage of While Iterator blocks" (Simulink)
- "hisl_0007: Usage of While Iterator subsystems" (Simulink)
- "hisl_0008: Usage of For Iterator Blocks" (Simulink)
- "hisl_0009: Usage of For Iterator Subsystem blocks" (Simulink)
- "hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks" (Simulink)

Display model version information

Check ID: `mathworks.do178.MdlChecksum`

Display model version information in your report.

Description

This check displays the following information for the current model:

- Version number
- Author
- Date
- Model checksum

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Could not retrieve model version and checksum information.	This summary is provided for your information. No action is required.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- "Reports for Code Generation" (Simulink Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

Check for root Inports with missing properties

Check ID: `mathworks.iec61508.RootLevelInports`

Identify root model Inport blocks with missing or inherited sample times, data types or port dimensions.

Description

Using root model Inport blocks that do not have defined sample time, data types or port dimensions can lead to undesired simulation results. Simulink back-propagates dimensions, sample times, and data types from downstream blocks unless you explicitly assign these values. You can specify Inport block properties with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. When you run the check, a results table provides links to Inport blocks and signal objects that do not pass, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing port dimension — Model contains Inport blocks with inherited port dimensions.	For the listed Inport blocks and Simulink signal objects, specify port dimensions.
Missing signal data type — Model contains Inport blocks with inherited data types.	For the listed Inport blocks and Simulink signal objects, specify data types.
Missing port sample time — Model contains Inport blocks with inherited sample times.	For the listed Inport blocks and Simulink signal objects, specify sample times. The sample times for root Inports with bus type must match the sample times specified at the leaf elements of the bus object.
Implicit resolution to a Simulink signal object — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

Tips

The following configurations pass this check:

- **Configuration Parameters > Solver > Periodic sample time constraint** is set to `Ensure sample time independent`
- For export-function models, *inherited sample time* is not flagged.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- “About Data Types in Simulink” (Simulink)

- “Determine Output Signal Dimensions” (Simulink)
- “Specify Sample Time” (Simulink)
- “hisl_0024: Inport interface definition” (Simulink)

Check for root Inports with missing range definitions

Check ID: `mathworks.iec61508.InportRange`

Identify root level Inport blocks with missing or erroneous minimum or maximum range values.

Description

The check identifies root level Inport blocks with missing or erroneous minimum or maximum range values. You can specify Inport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Inport blocks and signal objects that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing range — Model contains Inport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Inport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.
Missing range(s) for bus object — Bus objects defining the Inport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters .
Range specified will be ignored — Minimum or maximum values at Inports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.

Condition	Recommended Action
No data type specified — Model contains Inport blocks or Simulink signal objects with inherited data types.	Specify one of the supported data types: • `Enum` • `Simulink.AliasType` • `Simulink.Bus` • `Simulink.NumericType` • build-in
Implicit resolution to a Simulink signal object — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- "hisl_0025: Design min/max specification of input interfaces" (Simulink)

Check for root Outports with missing range definitions

Check ID: `mathworks.iec61508.OutportRange`

Identify root level Outport blocks with missing or erroneous minimum or maximum range values.

Description

The check identifies root level Outport blocks with missing or erroneous minimum or maximum range values. You can specify Outport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Outport blocks that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing range — Model contains Outport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Outport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.
Missing range(s) for bus object — Bus objects defining the Outport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters.
Range specified at Outport will be ignored — Minimum or maximum values at Outports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.
No bus data type specified — Model contains Outport block or Simulink signal objects with inherited bus data types.	For the Outport blocks and Simulink signal objects, specify one of the supported data types: • `Enum` • `Simulink.AliasType` • `Simulink.Bus` • `Simulink.NumericType` • build-in
Implicit resolution to a Simulink signal object — Model contains Outport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- “hisl_0026: Design min/max specification of output interfaces” (Simulink)

Check usage of Stateflow constructs

Check ID: `mathworks.iec61508.StateflowProperUsage`

Identify usage of Stateflow constructs that might impact safety.

Description

This check identifies instances of Stateflow software being used in a way that can impact an application's safety, including:

- Use of strong data typing
- Port name mismatches
- Scope of data objects and events
- Formatting of state action statements
- Ordering of states and transitions
- Unreachable code
- Indeterminate execution time

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
A Stateflow chart is not configured for strong data typing on boundaries between a Simulink model and the Stateflow chart. See “hisf_0009: Strong data typing (Simulink and Stateflow boundary)” (Simulink).	In the Chart properties dialog box, select **Use Strong Data Typing with Simulink I/O** for the Stateflow chart. When you select this check box, the Stateflow chart accepts input signals of any data type that Simulink models support, provided that the type of the input signal matches the type of the corresponding Stateflow input data object.
Signals have names that differ from those of their corresponding Stateflow ports.	• Check whether the ports are connected and, if not, fix the connections. • Change the names of the signals or the Stateflow ports so that the names match.
Local data is not defined in the Stateflow hierarchy at the chart level or below.	Define local data at the chart level or below.
A new line is missing from a state action after: • An `entry` (`en`), `during` (`du`), or `exit` (`ex`) statement • The semicolon (;) at the end of an assignment statement	Add missing new lines.
Stateflow charts have **User specified state/transition execution order** cleared. See “hisf_0002: User-specified state/transition execution order” (Simulink).	For the specified charts, in the Chart Properties dialog box, select **User specified state/transition execution order**.

Condition	Recommended Action
Any of the following: • **Wrap on overflow** is not set to `error`. • **Simulation range checking** is not set to `error`. • **Detect Cycles** is cleared. See “hisf_0011: Stateflow debugging settings” (Simulink).	In the Configuration Parameters dialog box, set: • **Diagnostics** > **Data Validity** > **Wrap on overflow** to `error`. • **Diagnostics** > **Data Validity** > **Simulation range checking** to `error`. In the model window, select: • **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Detect Cycles**.
The Stateflow chart contains a data object identifier defined in two or more scopes. See “hisl_0061: Unique identifiers for clarity” (Simulink).	For the identified chart, do one of the following: • Create a unique data object identifier within each of the scopes. • Create a unique data object identifier within the chart, at the parent level.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts. Exclusions will not work for library linked charts.

See Also

See the following topics in the Stateflow documentation:

- DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards'
  DO-331, Section MB.6.3.1.g 'Algorithms are accurate'
  DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
  DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'

DO-331, Section MB.6.3.2.g 'Algorithms are accurate'

- “Strong Data Typing with Simulink I/O” (Stateflow)
- “Property Fields” (Stateflow)
- “How Events Work in Stateflow Charts” (Stateflow)
- “Add Stateflow Data” (Stateflow)
- “Label States” (Stateflow)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check safety-related solver settings for simulation time

Check ID: `mathworks.iec61508.hisl_0040`

Check solver settings in the model configuration that apply to simulation time and might impact safety.

Description

This check verifies that the model solver configuration parameters pertaining to simulation time are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver setting to specify the start time for the simulation or generated code is set to a value other than `0.0`.	In the Configuration Parameters dialog box, set “Start time” (Simulink) or set the parameter `StartTime` to `0.0`.

Condition	Recommended Action
The solver setting to specify the stop time for the simulation or generated code is set to a negative value or a positive value greater than the value of "Application lifespan (days)" (Simulink). By default, "Application lifespan (days)" (Simulink) is `auto`. If you do not change this setting, any positive value for "Stop time" (Simulink) is valid.	In the Configuration Parameters dialog box, , set "Stop time" (Simulink) or set the parameter `StopTime` to a positive value that is less than the value of "Application lifespan (days)" (Simulink).

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- DO-331 Section MB.6.3.1.g—Algorithms are accurate
  DO-331 Section MB.6.3.2.g—Algorithms are accurate
- "Solver Pane" (Simulink)
- "Application lifespan (days)" (Simulink)
- "hisl_0040: Configuration Parameters > Solver > Simulation time" (Simulink)
- "hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)" (Simulink)

Check safety-related solver settings for solver options

Check ID: `mathworks.iec61508.hisl_0041`

Check solver settings in the model configuration that apply to solvers and might impact safety.

Description

This check verifies that the model solver configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver setting to specify the type of solver to simulate model is set to `Variable-step`.	In the Configuration Parameters dialog box, set “Type” (Simulink) or set the parameter `SolverType` to `Fixed-step`.
The solver setting to specify the solver to compute the states of the model during simulation or code generation is set to a value other than `Discrete(no continuous states)`.	In the Configuration Parameters dialog box, set “Solver” (Simulink) to `discrete(no continuous states)` or set the parameter `Solver` to `FixedStepDiscrete`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- “Solver Pane” (Simulink)
- “hisl_0041: Configuration Parameters > Solver > Solver options” (Simulink)

Check usage of shift operations for Stateflow data

Check ID: `mathworks.do178.hisf_0064`

Identify usage of shift operations for Stateflow data that might impact safety.

Description

This check inspects the shift operations that have shift operand values greater than the bit-width of the input or output type or a shift operand that has a negative value.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Right-shift operations are greater than the bit-width of the input type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.
Left-shift operations are greater than the bit-width of the output type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not support the shift operation that has the shift size defined as a Simulink signal or a variable.
- Does not support the shift operations that consist of shift size decided at run time.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- hisf_0064: Shift operations for Stateflow data to improve code compliance

Check assignment operations in Stateflow Charts

Check ID: `mathworks.do178.hisf_0065`

Identify assignment operations in Stateflow objects.

Description

This check identifies the assignment operations in Stateflow objects that cast integer and fixed-point calculations to wider data types than the input data types.

This check identifies only the assignments with arithmetic operations.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow object consists of assignment operations that cast integer and fixed-point calculations to wider data types than the input data types.	Explicitly replace assignment operator (=) to := operator in Stateflow objects.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- hisf_0065: Type cast operations in Stateflow to improve code compliance
- “Assignment (=, :=) Operations” (Stateflow)

Check Stateflow charts for unary operators

Check ID: `mathworks.do178.hisf_0211`

Identify unary operators in Stateflow charts.

Description

This check identifies the unary minus operators on unsigned data types in Stateflow charts.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart consists of a unary minus operator on unsigned data types.	Explicitly modify the unary operator on unsigned data types. For more information, see “Unary Operations” (Stateflow).

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not flag expressions with bitwise and arithmetic operators. For example, `-(u1/u2)` is not flagged.

See Also

- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

Check for blocks not recommended for MISRA C:2012

Check ID: `mathworks.misra.BlkSupport`

Identify blocks that are not supported or recommended for MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem.	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystemer. The deprecated Lookup Table blocks are Lookup and Lookup2D.	Consider replacing the deprecated Lookup Table blocks.
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- “hisl_0020: Blocks not recommended for MISRA C:2012 compliance” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “What Is a Model Advisor Exclusion?”

Check configuration parameters for MISRA C:2012

Check ID: `mathworks.misra.CodeGenSettings`

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model Verification block enabling is set to `Use local settings` or `Enable All`.	In the Configuration Parameters, set **Model Verification block enabling** to `Disable All`.
System target file is set to a GRT-based target.	In the Configuration Parameters dialog box, on the **Code Generation** pane, set **System target file** to an ERT-based target.
Code Generation > **Interface** parameters are not set to the recommended values.	In the Configuration Parameters dialog box: • Set **Code replacement library** to `None` or `AUTOSAR 4.0` • Set **Shared code placement** to `Shared location` • Clear **Support: non-finite numbers** • Clear **Support: continuous time** (ERT-based target only) • Clear **Support non-inlined S-functions** (ERT-based target only) • Clear **MAT-file logging**
Parenthesis level is not set to `Maximum (Specify precedence with parentheses)`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Parentheses level** to `Maximum (Specify precedence with parentheses)`.

Condition	Recommended Action
Casting Modes is not set to `Standards Compliant`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Casting Modes** to `Standards Compliant`.
`GenerateSharedConstants` is set to `on`.	Use `get_param` to set `GenerateSharedConstants` to `off`.
System-generated identifiers is set to `Classic`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Symbols** pane, set **System-generated identifiers** to `Shortened`.
Pack Boolean data into bitfields is selected and **Bitfield declarator type specifier** is set to `uchar_T`.	In the Configuration Parameters dialog box, on the **Optimization** > **Signals and Parameters** pane, if **Pack Boolean data into bitfields** is selected, set **Bitfield declarator type specifier** to `uint_T`.
Signed integer division rounds to is not set to `Zero` or `Floor`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, set **Signed integer division rounds to** to `Zero` or `Floor`.
Use division for fixed-point net slope computation is not set to `On` or `Use division for reciprocals of integers only`.	In the Configuration Parameters dialog box, on the **Optimization** pane, set **Use division for fixed-point net slope computation** to `On` or `Use division for reciprocals of integers only`.
Replace multiplications by powers of two with signed bitwise shifts is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, clear **Replace multiplications by powers of two with signed bitwise shifts**.
Allow right shifts on signed integers is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, Clear **Allow right shifts on signed integers**.

Condition	Recommended Action
Use dynamic memory allocation for model initialization is selected.	In the Configuration Parameters dialog box, clear **Use dynamic memory allocation for model initialization**.
Wrap on overflow is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Wrap on overflow** to `warning` or `error`.
Inf or NaN block output is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Inf or NaN block output** to `warning` or `error`.
Dynamic momory allocation in MATLAB Function blocks is selected.	In the Configuration Parameters dialog box, clear **Dynamic memory allocation in MATLAB Function blocks**.
`ERTFilePackagingFormat` is set to `Modular`.	Use `get_param` to set `ERTFilePackagingFormat` to `CompactWithDataFile` or `Compact`. If you click **Modify** to automatically fix the parameter setting, the value is set to `Compact`.
`PreserveStaticInFcnDecls` is set to `off`.	Use `get_param` to set `PreserveStaticInFcnDecls` to `on`. To set this value, `ERTFilePackagingFormat` must be set to `CompactWithDataFile` or `Compact`.

Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

This check does not review referenced models.

See Also

- “hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)

IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks

In this section...

In this section...

In this section...

IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks

IEC 61508, IEC 62304, ISO 26262, and EN 50128 checks facilitate designing and troubleshooting models, subsystems, and the corresponding generated code for applications to comply with IEC 61508-3, IEC 62304, ISO 26262-6, or EN 50128.

The Model Advisor performs a checkout of the Simulink Check license when you run the IEC 61508, IEC 62304, ISO 26262, or EN 50128 checks.

These checks are certified by the IEC Certification Kit for use in development processes that must comply with IEC 61508, ISO 26262, EN 50128, or derivative standards.

Tips

If your model uses model referencing, run the IEC 61508, IEC 62304, ISO 26262, or EN 50128 checks on all referenced models before running them on the top-level model.

See Also

- IEC 61508-3 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements
- IEC 62304 Medical device software - Software life cycle processes
- ISO 26262-6 Road vehicles - Functional safety - Part 6: Product development: Software level
- EN 50128 Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems
- Embedded Coder documentation:
  - “IEC 61508 Standard” (Embedded Coder)
  - “IEC 62304 Standard” (Embedded Coder)
  - “ISO 26262 Standard” (Embedded Coder)
  - “EN 50128 Standard” (Embedded Coder)

Check model object names

Check ID: `mathworks.iec61508.hisl_0032`

Check model object names.

Description

This check verifies that the following model object names comply with your own modeling guidelines or the high-integrity modeling guidelines. The check also verifies that the model object does not use a reserved name.

- Blocks
- Signals
- Parameters
- Busses
- Stateflow objects

Reserved names:

- MATLAB keywords
- Reserved keywords for C, C++, and code generation. For complete list, see “Reserved Keywords” (Simulink Coder) in the Simulink Coder documentation.
- `int8` , `uint8`
- `int16`, `uint16`
- `int32`, `uint32`
- `inf`, `Inf`
- `NaN`, `nan`
- `eps`
- `intmin`, `intmax`
- `realmin`, `realmax`
- `pi`
- `infinity`
- `Nil`

Note For some cases, the Model Advisor reports an issue in multiple subchecks of this check.

Available with Simulink Check.

Input Parameters

To specify the naming standard and model object names that the check flags, use the Model Advisor Configuration Editor.

1 Open the Model Configuration Editor and navigate to **Check model object names**. In the **Input Parameters** pane, for each of the model objects, select one of the following:

   - `MAAB` to use the MAAB naming standard. When you select `MAAB`, the check uses the regular expression `(^.{32,}$)|([^a-zA-Z_0-9])|(^\d)|(^ )|(__)|(^_)|(_$)` to verify that names:

     - Use these characters: `a-z`, `A-Z`, `0-9`, and the underscore `(_)`.
     - Do not start with a number.
     - Do not use underscores at the beginning or end of a string.
     - Do not use more than one consecutive underscore.
     - Use strings that are less than 32 characters.

   - `Custom` to use your own naming standard. When you select `Custom`, you can enter your own **Regular expression for prohibited *<model object>* names**. For example, if you want to allow more than one consecutive underscore, enter `(^.{32,}$)|([^a-zA-Z_0-9])|(^\d)|(^ )|(^_)|(_$)`
   - `None` if you do not want the check to verify the model object name

2 Click **Apply**.

3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

Results and Recommended Actions

Condition	Recommended Action
The model object names do not comply with the naming standard specified in the input parameters.	Update the model object names to comply with your own or the high-integrity guidelines.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- “hisl_0032: Model object names” (Simulink)
- MAAB guideline, Version 3.0: jc_0201: Usable characters for Subsystem names
- MAAB guideline, Version 3.0: jc_0211: Usable characters for Inport blocks and Outport blocks
- MAAB guideline, Version 3.0: jc_0221: Usable characters for signal line names
- MAAB guideline, Version 3.0: jc_0231: Usable characters for block names
- MAAB guideline, Version 3.0: na_0030: Usable characters for Simulink Bus names

Check safety-related optimization settings

Check ID: `mathworks.do178.OptionSet`

Check model configuration for optimization settings that can impact safety.

Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Block reduction optimization is selected. This optimization can remove blocks from generated code, resulting in requirements without associated code and violations for traceability requirements. See IEC 61508-3, Clauses 7.4.7.2, 7.4.8.3, and 7.7.2.8 which require to demonstrate that no unintended functionality has been introduced	Clear the **Block reduction** (Simulink) parameter in the Configuration Parameters dialog box or set parameter `BlockReduction` to `off`.
Implementation of logic signals as Boolean data is cleared. Strong data typing is recommended for safety-related code. See: IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language'	Select **Implement logic signals as boolean data (vs. double)** (Simulink) in the Configuration Parameters dialog box or set the parameter `BooleanDataType` to `on`.

Condition	Recommended Action
The model includes blocks that depend on elapsed or absolute time and is configured to minimize the amount of memory allocated for the timers. Such a configuration limits the number of days the application can execute before a timer overflow occurs. Many aerospace products are powered on continuously and timers should not assume a limited lifespan. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	Set **Application lifespan (days)** (Simulink) on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `LifeSpan` to `inf`.
The optimization that suppresses the generation of initialization code for root-level inports and outports that are set to zero is selected. For safety-related code, you should explicitly initialize all variables. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Remove root level I/O zero initialization** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `ZeroExternalMemoryAtStartup` to `on`. Alternatively, integrate external, hand-written code that initializes all I/O variables to zero explicitly.

Condition	Recommended Action
The optimization that suppresses the generation of initialization code for internal work structures, such as block states and block outputs that are set to zero, is selected. For safety-related code, you should explicitly initialize every variable. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Remove internal data zero initialization** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `ZeroInternalMemoryAtStartup` to `on`. Alternatively, integrate external, hand-written code that initializes every state variable to zero explicitly.
The optimization that suppresses generation of code resulting from floating-point to integer conversions that wrap out-of-range values is cleared. You must avoid overflows for safety-related code. When this optimization is off and your model includes blocks that disable the **Saturate on overflow** parameter, the code generator wraps out-of-range values for those blocks. This can result in unreachable and, therefore, untestable code. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	If you have a Simulink Coder license, select **Remove code from floating-point to integer conversions that wraps out-of-range values** (Simulink) on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `EfficientFloat2IntCast` to `on`.

Condition	Recommended Action
The optimization that suppresses generation of code that guards against division by zero for fixed-point data is selected. You must avoid division-by-zero exceptions in safety-related code. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming’ IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Remove code that protects against division arithmetic exceptions** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box or set the parameter `NoFixptDivByZeroProtection` to `off`.
The optimization that uses the specified minimum and maximum values for signals and parameters to optimize the generated code is selected. This might result in requirements without traceable code. See: IEC 61508-3, Table A.4 (3) 'Defensive Programming’ IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.3 (1) 'Defensive Programming'	If you have an Embedded Coder license, and you are using an ERT-based system target file, clear the **Optimize using the specified minimum and maximum values** (Simulink) check box on the **Optimization** pane in the Configuration Parameters dialog box.

Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- "Optimization Pane: General" (Simulink)
- "Optimize Generated Code Using Minimum and Maximum Values" (Embedded Coder)
- "hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)" (Simulink)
- "hisl_0046: Configuration Parameters > Optimization > Block reduction" (Simulink)
- "hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)" (Simulink)
- "hisl_0052: Configuration Parameters > Optimization > Data initialization" (Simulink)
- "hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values" (Simulink)
- "hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions" (Simulink)

Display model metrics and complexity report

Check ID: `mathworks.iec61508.MdlMetricsInfo`

Display number of elements and name, level, and depth of subsystems for the model or subsystem.

Description

The IEC 61508, ISO 26262, and EN 50128 standards recommend the usage of size and complexity metrics to assess the software under development. This check provides metrics information for the model. The provided information can be used to inspect whether the size or complexity of the model or subsystem exceeds given limits. The check displays:

- A block count for each Simulink block type contained in the given model, including library linked blocks.

- A count of Stateflow constructs in the given model (if applicable).
- Name, level, and depth of the subsystems contained in the given model (if applicable).
- The maximum subsystem depth of the given model.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
N/A	This summary is provided for your information. No action is required.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table B.9 (1) - Software module size limit, Table B.9 (2) - Software complexity control
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1a) - Enforcement of low complexity, Table 3 (a) - Hierarchical structure of software components, Table 3 (b) - Restricted size of software components, and Table 3 (c) - Restricted size of interfaces
- EN 50128, Table A.12 (8) - Limited size and complexity of Functions, Subroutines and Methods and (9) Limited number of subroutine parameters
- `sldiagnostics` in the Simulink documentation
- “Cyclomatic Complexity for Stateflow Charts” (Simulink Coverage) in the Simulink Check documentation

Check for unconnected objects

Check ID: `mathworks.iec61508.UnconnectedObjects`

Identify unconnected lines, input ports, and output ports in the model.

Description

Unconnected objects are likely to cause problems propagating signal attributes such as data, type, sample time, and dimensions.

Ports connected to Ground or Terminator blocks pass this check.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
There are unconnected lines, input ports, or output ports in the model or subsystem.	• Double-click an element in the list of unconnected items to locate the item in the model diagram. • Connect the objects identified in the results.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques
- EN 50128, Table A.4 (11) - Language Subset
- “Signal Basics” (Simulink)

Check for root Inports with missing properties

Check ID: `mathworks.iec61508.RootLevelInports`

Identify root model Inport blocks with missing or inherited sample times, data types or port dimensions.

Description

Using root model Inport blocks that do not have defined sample time, data types or port dimensions can lead to undesired simulation results. Simulink back-propagates dimensions, sample times, and data types from downstream blocks unless you explicitly assign these values. You can specify Inport block properties with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. When you run the check, a results table provides links to Inport blocks and signal objects that do not pass, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing port dimension — Model contains Inport blocks with inherited port dimensions.	For the listed Inport blocks and Simulink signal objects, specify port dimensions.
Missing signal data type — Model contains Inport blocks with inherited data types.	For the listed Inport blocks and Simulink signal objects, specify data types.
Missing port sample time — Model contains Inport blocks with inherited sample times.	For the listed Inport blocks and Simulink signal objects, specify sample times. The sample times for root Inports with bus type must match the sample times specified at the leaf elements of the bus object.
Implicit resolution to a Simulink signal object — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

Tips

The following configurations pass this check:

- **Configuration Parameters > Solver > Periodic sample time constraint** is set to `Ensure sample time independent`
- For export-function models, *inherited sample time* is not flagged.

See Also

- IEC 61508-3, Table B.9 (6) - Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-4, Table 2 (2) - Precisely defined interfaces
- EN 50128, Table A.3 (19) - Fully Defined Interface
- “About Data Types in Simulink” (Simulink)
- “Determine Output Signal Dimensions” (Simulink)
- “Specify Sample Time” (Simulink)
- “hisl_0024: Inport interface definition” (Simulink)

Check for MATLAB Function interfaces with inherited properties

Check ID: `mathworks.iec61508.himl_0002`

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Functions identified in the results. If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink): • Change **Complexity** from `Inherited` to `On` or `Off`. • Change **Type** from `Inherit: Same as Simulink` to an explicit type.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table B.9 (6) - Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation
- EN 50128, Table A.1 (11) - Software Interface Specifications
- “himl_0002: Strong data typing at MATLAB function boundaries” (Simulink)

Check MATLAB Function metrics

Check ID: `mathworks.iec61508.himl_0003`

Display complexity and code metrics for MATLAB Functions. Report metric violations.

Description

The IEC 61508, ISO 26262, and EN 50128 standards recommend the usage of size and complexity metrics to assess the software under development. This check provides complexity and code metrics for MATLAB Functions. The check additionally reports metric violations.

A results table provides links to MATLAB Functions that violate the complexity input parameters.

Available with Simulink Check.

Input Parameters

Maximum effective lines of code per function

Provide the maximum effective lines of code per function. Effective lines do not include empty lines, comment lines, or lines with a function `end` keyword.

Minimum density of comments

Provide minimum density of comments. Density is ratio of comment lines to total lines of code.

Maximum cyclomatic complexity per function

Provide maximum cyclomatic complexity per function. Cyclomatic complexity is the number of linearly independent paths through the source code.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Function violates the complexity input parameters.	For the MATLAB Function: • If effective lines of code is too high, further divide the MATLAB Function. • If comment density is too low, add comment lines. • If cyclomatic complexity per function is too high, further divide the MATLAB Function.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table B.9 (6) - Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation
- EN 50128, Table A.1(11) - Software Interface Specifications
- “himl_0003: Limitation of MATLAB function complexity” (Simulink)

Check for root Inports with missing range definitions

Check ID: `mathworks.iec61508.InportRange`

Identify root level Inport blocks with missing or erroneous minimum or maximum range values.

Description

The check identifies root level Inport blocks with missing or erroneous minimum or maximum range values. You can specify Inport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Inport blocks and signal objects that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing range — Model contains Inport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Inport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.

Condition	Recommended Action
Missing range(s) for bus object — Bus objects defining the Inport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters.
Range specified will be ignored — Minimum or maximum values at Inports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.
No data type specified — Model contains Inport blocks or Simulink signal objects with inherited data types.	Specify one of the supported data types: • `Enum` • `Simulink.AliasType` • `Simulink.Bus` • `Simulink.NumericType` • build-in
Implicit resolution to a Simulink signal object — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table B.9 (6) – Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-4, Table 2 (2) – Precisely defined interfaces
- EN 50128, Table A.1(11) – Software Interface Specifications
  EN 50128 Table A.3 (19) ‘Fully Defined Interface‘
- “hisl_0025: Design min/max specification of input interfaces” (Simulink)

Check for root Outports with missing range definitions

Check ID: `mathworks.iec61508.OutportRange`

Identify root level Outport blocks with missing or erroneous minimum or maximum range values.

Description

The check identifies root level Outport blocks with missing or erroneous minimum or maximum range values. You can specify Outport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Outport blocks that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Missing range — Model contains Outport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Outport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.
Missing range(s) for bus object — Bus objects defining the Outport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters.
Range specified at Outport will be ignored — Minimum or maximum values at Outports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.

Condition	Recommended Action
No bus data type specified — Model contains Outport block or Simulink signal objects with inherited bus data types.	For the Outport blocks and Simulink signal objects, specify one of the supported data types: • `Enum` • `Simulink.AliasType` • `Simulink.Bus` • `Simulink.NumericType` • build-in
Implicit resolution to a Simulink signal object — Model contains Outport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property **Signal name must resolve to Simulink signal object**.

Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table B.9 (6) – Fully defined interface
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-4, Table 2 (2) - Precisely defined interfaces
- EN 50128, Table A.1(11) – Software Interface Specifications
  EN 50128 Table A.3 (19) 'Fully Defined Interface'
- "hisl_0026: Design min/max specification of output interfaces" (Simulink)

Check for blocks not recommended for C/C++ production code deployment

Check ID: `mathworks.iec61508.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

Description

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder. If you are using blocks with support notes for code generation, review the information and follow the given advice.

Available with Simulink Check and Embedded Coder.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “Blocks and Products Supported for C Code Generation” (Simulink Coder)

Check for variant blocks with 'Generate preprocessor conditionals' active

Check ID: `mathworks.do178.VariantBlock`

Check variant block parameters for settings that might result in code that does not trace to requirements.

Description

This check verifies that variant block parameters for code generation are set to trace to requirements.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The option to generate preprocessor conditionals is selected in one or more variant blocks in the model.	In order to simplify the tracing of code to requirements, consider clearing the option to generate preprocessor conditionals in variant blocks.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508–3, Table A.4 (7) 'Use of trusted / verified software modules and components'
- “hisl_0023: Verification of model and subsystem variants” (Simulink)

Check Stateflow charts for uniquely defined data objects

Check ID: `mathworks.do178.hisl_0061`

Identify Stateflow charts that include data objects that are not uniquely defined.

Description

This check searches your model for local data in Stateflow charts that is not uniquely defined.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart contains a data object identifier defined in two or more scopes.	For the identified chart, do one of the following: • Create a unique data object identifier within each of the scopes. • Create a unique data object identifier within the chart, at the parent level.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508–3, Table A.3 (3) 'Language subset'
  IEC 61508–3, Table A.4 (5) 'Design and coding standards'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1e) 'Use of established design principles'
  ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation'
  ISO 26262-6, Table 1 (1g) 'Use of style guides'
  ISO 26262-6, Table 1 (1h) 'Use of naming conventions'
- EN 50128, Table A.4 (11) 'Language Subset'
  EN 50128, Table A.12 (1) 'Coding Standard'
  EN 50128, Table A.12 (2) 'Coding Style Guide'
- “hisl_0061: Unique identifiers for clarity” (Simulink)

Check usage of Stateflow constructs

Check ID: `mathworks.iec61508.StateflowProperUsage`

Identify usage of Stateflow constructs that might impact safety.

Description

This check identifies instances of Stateflow software being used in a way that can impact an application's safety, including:

- Use of strong data typing
- Port name mismatches
- Scope of data objects and events
- Formatting of state action statements
- Ordering of states and transitions
- Unreachable code
- Indeterminate execution time

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
A Stateflow chart is not configured for strong data typing on boundaries between a Simulink model and the Stateflow chart. See: • "hisf_0009: Strong data typing (Simulink and Stateflow boundary)" (Simulink) • IEC 61508-3, Table A.3 (2) - Strongly typed programming language • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) - Enforcement of strong typing • EN 50128, Table A.4 (8) - Strongly Typed Programming Language	In the Chart properties dialog box, select **Use Strong Data Typing with Simulink I/O** for the Stateflow chart. When you select this check box, the Stateflow chart accepts input signals of any data type that Simulink models support, provided that the type of the input signal matches the type of the corresponding Stateflow input data object.

Condition	Recommended Action
Signals have names that differ from those of their corresponding Stateflow ports. See: • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset	• Check whether the ports are connected and, if not, fix the connections. • Change the names of the signals or the Stateflow ports so that the names match.
Local data is not defined in the Stateflow hierarchy at the chart level or below. See: • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset	Define local data at the chart level or below.

Condition	Recommended Action
A new line is missing from a state action after: • An `entry` (`en`), `during` (`du`), or `exit` (`ex`) statement • The semicolon (;) at the end of an assignment statement See: • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset	Add missing new lines.
Stateflow charts have **User specified state/transition execution order** cleared. See: • "hisf_0002: User-specified state/ transition execution order" (Simulink) • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.4 (11) - Language Subset	For the specified charts, in the Chart Properties dialog box, select **User specified state/transition execution order**.

Condition	Recommended Action
Any of the following: • **Wrap on overflow** is not set to `error`. • **Simulation range checking** is not set to `error`. • **Detect Cycles** is cleared. See: • "hisf_0011: Stateflow debugging settings" (Simulink) • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1d) - Use of defensive implementation techniques • EN 50128, Table A.3 (1) - Defensive Programming • EN 50128, Table A.4 (11) - Language Subset	In the Configuration Parameters dialog box, set: • **Diagnostics** > **Data Validity** > **Wrap on overflow** to `error`. • **Diagnostics** > **Data Validity** > **Simulation range checking** to `error`. In the model window, select: • **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Detect Cycles**.

Condition	Recommended Action
The Stateflow chart contains a data object identifier defined in two or more scopes. See: • “hisl_0061: Unique identifiers for clarity” (Simulink) • IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (5) - Design and coding standards • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1e) - Use of established design principles, Table 1 (1f) - Use of unambiguous graphical representation, Table 1 (1g) - Use of style guides, Table 1 (1h) - Use of naming conventions • EN 50128, Table A.4 (11) - Language Subset, Table A.12 (1) - Coding Standard, Table A.12 (2) - Coding Style Guide	For the identified chart, do one of the following: • Create a unique data object identifier within each of the scopes. • Create a unique data object identifier within the chart, at the parent level.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts. Exclusions will not work for library linked charts.

See Also

See the following topics in the Stateflow documentation:

- “Strong Data Typing with Simulink I/O” (Stateflow)
- “Property Fields” (Stateflow)

- “How Events Work in Stateflow Charts” (Stateflow)
- “Add Stateflow Data” (Stateflow)
- “Label States” (Stateflow)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check state machine type of Stateflow charts

Check ID: `mathworks.iec61508.hisf_0001`

Identify whether Stateflow charts are all Mealy or all Moore charts.

Description

Compares the state machine type of all Stateflow charts to the type that you specify in the input parameters.

Available with Simulink Check.

Input Parameters

Mealy or Moore

Check whether charts use the same state machine type, and are all Mealy or all Moore charts.

Mealy

Check whether all charts are Mealy charts.

Moore

Check whether all charts are Moore charts.

Results and Recommended Actions

Condition	Recommended Action
The input parameter is set to `Mealy or Moore` and charts in the model use either of the following: • Classic state machine types. • Multiple state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to either `Mealy` or `Moore`. Use the same state machine type for all charts in the model.

Condition	Recommended Action
The input parameter is set to `Mealy` and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to `Mealy`.
The input parameter is set to `Moore` and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify **State Machine Type** to `Moore`.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “hisf_0001: Mealy and Moore semantics” (Simulink)
- “Overview of Mealy and Moore Machines” (Stateflow) in the Stateflow documentation.
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow charts for ordering of states and transitions

Check ID: `mathworks.do178.hisf_0002`

Identify Stateflow charts that have **User specified state/transition execution order** cleared.

Description

Identify Stateflow charts that have **User specified state/transition execution order** cleared, and therefore do not use explicit ordering of parallel states and transitions.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Stateflow charts have **User specified state/transition execution order** cleared.	For the specified charts, in the Chart Properties dialog box, select **User specified state/transition execution order**.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** selects **User specified state/transition execution order** for the specified charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation'
- EN 50128, Table A.4 (11) 'Language Subset'
- “hisf_0002: User-specified state/transition execution order” (Simulink)

  “Transition Testing Order in Multilevel State Hierarchy” (Stateflow)
- “Execution Order for Parallel States” (Stateflow)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow debugging options

Check ID: `mathworks.do178.hisf_0011`

Check the Stateflow debugging settings.

Description

Verify the following debugging settings.

- **Wrap on overflow**
- **Simulation range checking**
- **Detect Cycles**

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Any of the following: • **Wrap on overflow** is not set to `error`. • **Simulation range checking** is not set to `error`. • **Detect Cycles** is cleared.	In the Configuration Parameters dialog box, set: • Wrap on overflow to `error`. • Simulation range checking to `error`. In the model window, select: • **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Detect Cycles**.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** selects the specified debugging options.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1d) - Use of defensive implementation techniques
- EN 50128, Table A.3 (1) - Defensive Programming
  EN 50128, Table A.4 (11) - Language Subset
- “hisf_0011: Stateflow debugging settings” (Simulink)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)

Check Stateflow charts for transition paths that cross parallel state boundaries

Check ID: `mathworks.iec61508.hisf_0013`

Identify transition paths that cross parallel state boundaries in Stateflow charts.

Description

Identify transition paths that cross parallel state boundaries in Stateflow charts. Using such transition paths creates diagrams that consist of transition executions, which are difficult to understand.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow charts have transition paths that cross parallel state boundaries.	Modify the Stateflow chart so that transitions do not cross parallel state boundaries. For more information see, “Defining Transitions Between States” (Stateflow).

Capabilities and Limitations

- Does not run on library models.

- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.
- Analyzes content in all masked subsystems.

See Also

- IEC 61508-3, Table A.3 (3) Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
- EN 50128, Table A.4 (11) Language Subset
- “hisf_0013: Usage of transition paths (crossing parallel state boundaries)” (Simulink)
- “Chart Properties” (Simulink)
- “Chart Architecture” (Simulink)
- “Process for Entering, Executing, and Exiting States” (Stateflow)

Check Stateflow charts for strong data typing

Check ID: `mathworks.iec61508.hisf_0015`

Identify variables and parameters in expressions with different data types in Stateflow objects.

Description

To facilitate strong data typing, this check identifies the variables and parameters in expressions with different data types in Stateflow states and transitions.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow objects have variables and parameters in expressions with different data types.	Explicitly cast variables and parameters in expressions to the same data types. For more information see, `cast`.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.
- Analyzes content in all masked subsystems.
- Does not analyze the type of literals in expressions in Stateflow objects. Explicitly casts types of literals to the intended data type.
- Does not flag expressions with true and false keywords. For more information, see "Reserved Keywords for Code Generation" (Embedded Coder).

See Also

- IEC 61508-3, Table A.3 (2) Strongly typed programming language
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1c) Enforcement of strong typing
- EN 50128, Table A.4 (8) Strongly Typed Programming Language
- "hisf_0015: Strong data typing (casting variables and parameters in expressions)" (Simulink)
- "Chart Properties" (Simulink)
- "Chart Architecture" (Simulink)
- "Use Data Types in Stateflow" (Stateflow)

Check usage of lookup table blocks

Check ID: `mathworks.do178.LUTRangeCheckCode`

Check for lookup table blocks that do not generate out-of-range checking code.

Description

This check verifies that the following blocks generate code to protect against inputs that fall outside the range of valid breakpoint values:

- 1-D Lookup Table
- 2-D Lookup Table

- n-D Lookup Table
- Prelookup

This check also verifies that Interpolation Using Prelookup blocks generate code to protect against inputs that fall outside the range of valid index values.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The lookup table block does not generate out-of-range checking code.	Change the setting on the block dialog box so that out-of-range checking code is generated. • For the 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, and Prelookup blocks, clear the check box for **Remove protection against out-of-range input in generated code**. • For the Interpolation Using Prelookup block, clear the check box for **Remove protection against out-of-range index in generated code**.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

Action Results

Clicking **Modify** verifies that lookup table blocks are set to generate out-of-range checking code.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
  IEC 61508-3, Table A.4 (3) 'Defensive programming'

- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques'
- EN 50128, Table A.4 (11) 'Language Subset'
  EN 50128, Table A.3 (1) 'Defensive Programming'
- “hisl_0033: Usage of Lookup Table blocks” (Simulink)
- n-D Lookup Table block
- Prelookup block
- Interpolation Using Prelookup block

Check for model elements that do not link to requirements

Check ID: `mathworks.iec61508.RequirementInfo`

Check whether Simulink model elements link to a requirements document.

Description

This check verifies whether model objects link to a document containing engineering requirements for traceability.

This check verifies whether model objects link to a document containing engineering requirements for traceability.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Blocks do not link to a requirements document.	Link to requirements document. See “Link to Requirements Document Using Selection-Based Linking” (Simulink Requirements) .

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.

- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.
- Does not allow the exclusion of Stateflow elements.

Tip

Run this check from the top model or subsystem that you want to check.

See Also

- IEC 61508-3, Table A.2 (12) - Computer-aided specification and design tools, Table A. 2 (9) - Forward traceability between the software safety requirements specification and software architecture, Table A.2 (10) - Backward traceability between the software safety requirements specification and software architecture, Table A.4 (8) - Forward traceability between the software safety requirements specification and software design, Table A.8 (1) - Impact analysis
- IEC 62304, 5.2 - Software requirements analysis, 7.4.2 - Analyze impact of software changes on existing risk control measures
- ISO 26262-6, Table 8 (1a) - Documentation of the software unit design in natural language, ISO 26262-6: 7.4.2.a - The verifiability of the software architectural design, ISO 26262-8: 8.4.3 Change request analysis
- EN 50128, Table A.3 (23) - Modeling supported by computer aided design and specification tools, Table D.58 - Traceability, Table A.10 (1) - Impact Analysis
- hisl_0070: Placement of requirement links in a model
- “Requirements Traceability” (Simulink)
- “Requirements Traceability in Simulink” (Simulink)
- “Requirements Traceability Links” (Simulink Requirements)
- “Find Model Elements in Simulink Models” (Simulink)

Check for inconsistent vector indexing methods

Check ID: `mathworks.iec61508.hisl_0021`

Identify blocks with inconsistent indexing method.

Description

Using inconsistent block indexing methods can result in modeling errors. You should use a consistent vector indexing method for all blocks. This check identifies blocks with inconsistent indexing methods. The indexing methods are zero-based, one-based or user-specified.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks with inconsistent indexing methods. The indexing methods are zero-based, one-based or user-specified.	Modify the model to use a single consistent indexing method.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508–3, Table A.3 (3) - Language subset, Table A.4 (5) - Design and coding standards
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1e) - Use of established design principles, Table 1 (1f) - Use of unambiguous graphical representation, Table 1 (1g) - Use of style guides, Table 1 (1h) - Use of naming conventions
- EN 50128, Table A.4 (11) - Language Subset, Table A.12 (1) - Coding Standard
- “hisl_0021: Consistent vector indexing method” (Simulink)

Check safety-related solver settings for simulation time

Check ID: `mathworks.iec61508.SimulationTimeOptions`

Check solver settings in the model configuration that apply to simulation time and might impact safety.

Description

This check verifies that the model solver configuration parameters pertaining to simulation time are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver setting to specify the start time for the simulation or generated code is set to a value other than `0.0`.	In the Configuration Parameters dialog box,, set “Start time” (Simulink) or set the parameter `StartTime` to `0.0`.
The solver setting to specify the stop time for the simulation or generated code is set to a negative value or a positive value greater than the value of “Application lifespan (days)” (Simulink). By default, “Application lifespan (days)” (Simulink) is `auto`. If you do not change this setting, any positive value for “Stop time” (Simulink) is valid.	In the Configuration Parameters dialog box, set “Stop time” (Simulink) or set the parameter `StopTime` to a positive value that is less than the value of “Application lifespan (days)” (Simulink).

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- IEC 61508-3, Table A.3 (3) Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets

- EN 50128, Table A.4 (11) Language Subset
- “Solver Pane” (Simulink)
- “Application lifespan (days)” (Simulink)
- “hisl_0040: Configuration Parameters > Solver > Simulation time” (Simulink)
- “hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)” (Simulink)

Check safety-related solver settings for solver options

Check ID: `mathworks.iec61508.hisl_0041`

Check solver settings in the model configuration that apply to solvers and might impact safety.

Description

This check verifies that the model solver configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver setting to specify the type of solver to simulate model is set to `Variable-step`.	In the Configuration Parameters dialog box, set “Type” (Simulink) or set the parameter `SolverType` to `Fixed-step`.
The solver setting to specify the solver to compute the states of the model during simulation or code generation is set to a value other than `Discrete(no continuous states)`.	In the Configuration Parameters dialog box, set “Solver” (Simulink) to `discrete(no continuous states)` or set the parameter `Solver` to `FixedStepDiscrete`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- IEC 61508-3, Table A.3 (3) Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
- EN 50128, Table A.4 (11) Language Subset
- “Solver Pane” (Simulink)
- “hisl_0041: Configuration Parameters > Solver > Solver options” (Simulink)

Check safety-related solver settings for tasking and sample-time

Check ID: `mathworks.iec61508.hisl_0042`

Check solver settings in the model configuration that apply to periodic sample time constraints and might impact safety.

Description

This check verifies that model configuration parameters are set optimally to ensure that the model operates at a specific set of prioritized periodic sample times for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The solver settings that select constraints on the sample times defined by the model is set to `Unconstrained` or `Ensure sample time independent`.	• In the Configuration Parameters dialog box, set "Periodic sample time constraint" (Simulink) or set the parameter `SampleTimeConstraint` to `Specified` and assign a value to **Sample time properties**. • If you use a referenced model as a reusable function, set "Periodic sample time constraint" (Simulink) to `Ensure sample time independent`.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
- EN 50128, Table A.4 (11) Language Subset
- hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
- "Periodic sample time constraint" (Simulink)

Check safety-related diagnostic settings for solvers

Check ID: `mathworks.do178.SolverDiagnosticsSet`

Check model configuration for diagnostic settings that apply to solvers and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic for detecting automatic breakage of algebraic loops is set to `none` or `warning`. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur.	Set **Algebraic loop** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `AlgebraicLoopMsg` to `error`. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.
The diagnostic for detecting automatic breakage of algebraic loops for Model blocks, atomic subsystems, and enabled subsystems is set to `none` or `warning`. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur.	Set **Minimize algebraic loop** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `ArtificialAlgebraicLoopMsg` to `error`. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.
The diagnostic for detecting potential conflict in block execution order is set to `none` or `warning`. For safety-related applications, block execution order must be predictable. A model developer needs to know when conflicting block priorities exist.	Set **Block priority violation** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `BlockPriorityViolationMsg` to `error`.
The diagnostic for detecting whether a model contains an S-function that has not been specified explicitly to inherit sample time is set to `none` or `warning`. These settings can result in unpredictable behavior. A model developer needs to know when such an S-function exists in a model so it can be modified to produce predictable behavior.	Set **Unspecified inheritability of sample time** (Simulink) in the Configuration Parameters dialog box or set the parameter `UnknownTsInhSupMsg` to `error`.

Condition	Recommended Action
The diagnostic for detecting whether the Simulink software automatically modifies the solver, step size, or simulation stop time is set to `none` or `warning`. Such changes can affect the operation of generated code. For safety-related applications, it is better to detect such changes so a model developer can explicitly set the parameters to known values.	Set **Automatic solver parameter selection** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `SolverPrmCheckMsg` to `error`.
The diagnostic for detecting when a name is used for more than one state in the model is set to `none`. State names within a model should be unique. For safety-related applications, it is better to detect name clashes so a model developer can fix them.	Set **State name clash** (Simulink) on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box or set the parameter `StateNameClashWarn` to `warning`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
- EN 50128, Table A.4 (11) 'Language Subset'
- “hisl_0043: Configuration Parameters > Diagnostics > Solver” (Simulink)
- “Model Configuration Parameters: Diagnostics” (Simulink)
- “View Diagnostics” (Simulink)

Check safety-related diagnostic settings for sample time

Check ID: `mathworks.do178.SampleTimeDiagnosticsSet`

Check model configuration for diagnostic settings that apply to sample time and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to sample times are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic for detecting when a source block, such as a Sine Wave block, inherits a sample time (specified as -1) is set to `none` or `warning`. The use of inherited sample times for a source block can result in unpredictable execution rates for the source block and blocks connected to it. For safety-related applications, source blocks should have explicit sample times to prevent incorrect execution sequencing.	Set **Source block specifies -1 sample time** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `InheritedTslnSrcMsg` to `error`.
The diagnostic for detecting invalid rate transitions between two blocks operating in multitasking mode is set to `none` or `warning`. Such rate transitions should not be used for embedded real-time code.	Set **Multitask rate transition** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `MultiTaskRateTransMsg` to `error`.

Condition	Recommended Action
The diagnostic for detecting subsystems that can cause data corruption or nondeterministic behavior is set to `none` or `warning`. This diagnostic detects whether conditionally executed multirate subsystems (enabled, triggered, or function-call subsystems) operate in multitasking mode. Such subsystems can corrupt data and behave unpredictably in real-time environments that allow preemption.	Set **Multitask conditionally executed subsystem** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `MultiTaskCondExecSysMsg` to `error`.
The diagnostic for checking sample time consistency between a Signal Specification block and the connected destination block is set to `none` or `warning`. An over-specified sample time can result in an unpredictable execution rate.	Set **Enforce sample times specified by Signal Specification blocks** (Simulink) on the **Diagnostics** > **Sample Time** pane in the Configuration Parameters dialog box or set the parameter `SigSpecEnsureSampleTimeMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to sample time and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
- EN 50128, Table A.4 (11) 'Language Subset'
- “Model Configuration Parameters: Sample Time Diagnostics” (Simulink)
- “View Diagnostics” (Simulink)
- “hisl_0044: Configuration Parameters > Diagnostics > Sample Time” (Simulink)

Check safety-related diagnostic settings for signal data

Check ID: `mathworks.do178.DataValiditySignalsDiagnosticsSet`

Check model configuration for diagnostic settings that apply to signal data and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to signal data are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that specifies how the Simulink software resolves signals associated with `Simulink.Signal` objects is set to `Explicit and implicit` or `Explicit and warn implicit`. For safety-related applications, model developers should be required to define signal resolution explicitly.	Set **Signal resolution** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalResolutionControl` to `Explicit only`. This provides predictable operation by requiring users to define each signal and block setting that must resolve to `Simulink.Signal` objects in the workspace. Alternatively, to disable the use of `Simulink.Signal` objects, set the configuration parameter to `None`.
The Product block diagnostic that detects a singular matrix while inverting one of its inputs in matrix multiplication mode is set to `none` or `warning`. Division by a singular matrix can result in numeric exceptions when executing generated code. This is not acceptable in safety-related systems.	Set **Division by singular matrix** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `CheckMatrixSingularityMsg` to `error`.

Condition	Recommended Action
The diagnostic that detects when the Simulink software cannot infer the data type of a signal during data type propagation is set to `none` or `warning`. For safety-related applications, model developers must verify the data types of signals.	Set **Underspecified data types** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `UnderSpecifiedDataTypeMsg` to `error`.
The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type is set to `none` or `warning`. Undetected numeric overflows can result in unexpected application behavior.	Set **Wrap on overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `IntegerOverflowMsg` to `error`.
The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type, resulting in a saturation, is set to `none` or `warning`. Undetected numeric overflows can result in unexpected application behavior.	Set **Saturate on overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `IntegerSaturationMsg` to `error`.
The diagnostic that detects when the value of a block output signal is `Inf` or `NaN` at the current time step is set to `none` or `warning`. When this type of block output signal condition occurs, numeric exceptions can result, and numeric exceptions are not acceptable in safety-related applications.	Set **Inf or NaN block output** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalInfNanChecking` to `error`.
The diagnostic that detects Simulink object names that begin with `rt` is set to `none` or `warning`. This diagnostic prevents name clashes with generated signal names that have an `rt` prefix.	Set **"rt" prefix for identifiers** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `RTPrefix` to `error`.

Condition	Recommended Action
The diagnostic that detects simulation range checking is set to `none` or `warning`. This diagnostic detects when signals exceed their specified ranges during simulation. Simulink compares the signal values that a block outputs with the specified range and the block data type.	Set **Simulation range checking** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `SignalRangeChecking` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal data and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques'
- EN 50128, Table A.4 (11) 'Language Subset'
  EN 50128, Table A.3 (1) 'Defensive Programming'
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- “View Diagnostics” (Simulink)
- “hisl_0005: Usage of Product blocks” (Simulink)

Check safety-related diagnostic settings for compatibility

Check ID: `mathworks.do178.CompatibilityDiagnosticsSet`

Check model configuration for diagnostic settings that affect compatibility and that might impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to compatibility are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a block has not been upgraded to use features of the current release is set to `none` or `warning`. An S-function written for an earlier version might not be compatible with the current version and generated code could operate incorrectly.	Set **S-function upgrades needed** (Simulink) on the **Diagnostics** > **Compatibility** pane in the Configuration Parameters dialog box or set the parameter `SFcnCompatibilityMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that affect compatibility and that might impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.4 (3) 'Defensive Programming’
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques'
- EN 50128, Table A.3 (1) 'Defensive Programming'
- “View Diagnostics” (Simulink)
- “Model Configuration Parameters: Compatibility Diagnostics” (Simulink)
- “hisl_0301: Configuration Parameters > Diagnostics > Compatibility” (Simulink)

Check safety-related diagnostic settings for parameters

Check ID: `mathworks.do178.DataValidityParamDiagnosticsSet`

Check model configuration for diagnostic settings that apply to parameters and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to parameters are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a parameter downcast occurs is set to `none` or `warning`. A downcast to a lower signal range can result in numeric overflows of parameters, resulting in unexpected behavior.	Set **Detect downcast** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterDowncastMsg` to `error`.
The diagnostic that detects when a parameter underflow occurs is set to `none` or `warning`. When the data type of a parameter does not have enough resolution, the parameter value is zero instead of the specified value. This can lead to incorrect operation of generated code.	Set **Detect underflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterUnderflowMsg` to `error`.
The diagnostic that detects when a parameter overflow occurs is set to `none` or `warning`. Numeric overflows can result in unexpected application behavior and should be detected and fixed in safety-related applications.	Set **Detect overflow** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterOverflowMsg` to `error`.
The diagnostic that detects when a parameter loses precision is set to `none` or `warning`. Not detecting such errors can result in a parameter being set to an incorrect value in the generated code.	Set **Detect precision loss** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterPrecisionLossMsg` to `error`.

Condition	Recommended Action
The diagnostic that detects when an expression with tunable variables is reduced to its numerical equivalent is set to `none` or `warning`. This can result in a tunable parameter unexpectedly not being tunable in generated code.	Set **Detect loss of tunability** (Simulink) on the **Diagnostics** > **Data Validity** pane in the Configuration Parameters dialog box or set the parameter `ParameterTunabilityLossMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to parameters and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.4 (3) 'Defensive Programming’
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques'
- EN 50128, Table A.3 (1) 'Defensive Programming'
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- “View Diagnostics” (Simulink)
- “hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters” (Simulink)

Check safety-related diagnostic settings for model initialization

Check ID: `mathworks.do178.InitDiagnosticsSet`

In the model configuration, check diagnostic settings that affect model initialization and might impact safety.

Description

This check verifies that model diagnostic configuration parameters for initialization are optimally set to generate code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. The “Check undefined subsystem initial output” (Simulink) diagnostic is cleared. This diagnostic specifies whether Simulink displays a warning if the model contains a conditionally executed subsystem, in which a block with a specified initial condition drives an Outport block with an undefined initial condition. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic` and select **Check undefined subsystem initial output** (Simulink). • Set the parameter `CheckSSInitialOutputMsg` to `on`.
In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. This diagnostic detects potential initial output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic`. • Set the parameter `CheckExecutionContextPreStartOutputMsg` to `on`.

Condition	Recommended Action
In the Configuration Parameters dialog box, the "Underspecified initialization detection" (Simulink) diagnostic is set to `Classic`, ensuring compatibility with previous releases of Simulink. The "Check runtime output of execution context" (Simulink) diagnostic is cleared. This diagnostic detects potential output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized and feeds into a block with a tunable parameter. If undetected, this condition can cause the behavior of the downstream block to be nondeterministic.	Do one of the following: • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Simplified`. • In the Configuration Parameters dialog box, set **Underspecified initialization detection** (Simulink) to `Classic` and select **Check runtime output of execution context** (Simulink). • Set the parameter `CheckExecutionContextRuntimeOutputMsg` to `on`.

Action Results

To configure the diagnostic settings that affect model initialization and might impact safety, click **Modify Settings**.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- "View Diagnostics" (Simulink)
- "Model Configuration Parameters: Data Validity Diagnostics" (Simulink)
- "hisl_0304: Configuration Parameters > Diagnostics > Model initialization" (Simulink)

Check safety-related diagnostic settings for data used for debugging

Check ID: `mathworks.do178.DataValidityDebugDiagnosticsSet`

Check model configuration for diagnostic settings that apply to data used for debugging and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to debugging are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that enables model verification blocks is set to `Use local settings` or `Enable all`. Such blocks should be disabled because they are assertion blocks, which are for verification only. Model developers should not use assertions in embedded code.	In the Configuration Parameters dialog box, set **Model Verification block enabling** (Simulink) or set parameter `AssertControl` to `Disable All`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data used for debugging and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset

- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- “hisl_0305: Configuration Parameters > Diagnostics > Debugging” (Simulink)

Check safety-related diagnostic settings for data store memory

Check ID: `mathworks.do178.DataStoreMemoryDiagnosticsSet`

Check model configuration for diagnostic settings that apply to data store memory and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to data store memory are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether the model attempts to read data from a data store in which it has not stored data in the current time step is set to a value other than `Enable all as errors`. Reading data before it is written can result in use of stale data or data that is not initialized.	Set **Detect read before write** (Simulink) in the Configuration Parameters dialog box or set the parameter `ReadBeforeWriteMsg` to `Enable all as errors`.
The diagnostic that detects whether the model attempts to store data in a data store, after previously reading data from it in the current time step, is set to a value other than `Enable all as errors`. Writing data after it is read can result in use of stale or incorrect data.	Set **Detect write after read** (Simulink) in the Configuration Parameters dialog box or set the parameter `WriteAfterReadMsg` to `Enable all as errors`.
The diagnostic that detects whether the model attempts to store data in a data store twice in succession in the current time step is set to a value other than `Enable all as errors`. Writing data twice in one time step can result in unpredictable data.	Set **Detect write after write** (Simulink) in the Configuration Parameters dialog box or set the parameter `WriteAfterWriteMsg` to `Enable all as errors`.

Condition	Recommended Action
The diagnostic that detects when one task reads data from a Data Store Memory block to which another task writes data is set to `none` or `warning`. Reading or writing data in different tasks in multitask mode can result in corrupted or unpredictable data.	Set **Multitask data store** (Simulink) in the Configuration Parameters dialog box or set the parameter `MultiTaskDSMMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data store memory and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques'
- EN 50128, Table A.4 (11) 'Language Subset'
  EN 50128, Table A.3 (1) 'Defensive Programming'
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- “hisl_0013: Usage of data store blocks” (Simulink)

Check safety-related diagnostic settings for signal connectivity

Check ID: `mathworks.do178.ConnectivitySignalsDiagnosticsSet`

Check model configuration for diagnostic settings that apply to signal connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to signal connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects virtual signals that have a common source signal but different labels is set to `none` or `warning`. This diagnostic pertains to virtual signals only and has no effect on generated code. However, signal label mismatches can lead to confusion during model reviews.	Set **Signal label mismatch** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `SignalLabelMismatchMsg` to `error`.
The diagnostic that detects when the model contains a block with an unconnected input signal is set to `none` or `warning`. This must be detected because code is not generated for unconnected block inputs.	Set **Unconnected block input ports** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedInputMsg` to `error`.
The diagnostic that detects when the model contains a block with an unconnected output signal is set to `none` or `warning`. This must be detected because dead code can result from unconnected block output signals.	Set **Unconnected block output ports** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedOutputMsg` to `error`.
The diagnostic that detects unconnected signal lines and unmatched Goto or From blocks is set to `none` or `warning`. This error must be detected because code is not generated for unconnected lines.	Set **Unconnected line** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `UnconnectedLineMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- “Signal Basics” (Simulink)
- “hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals” (Simulink)

Check safety-related diagnostic settings for bus connectivity

Check ID: `mathworks.do178.ConnectivityBussesDiagnosticsSet`

Check model configuration for diagnostic settings that apply to bus connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to bus connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a Model block's root Outport block is connected to a bus but does not specify a bus object is set to `none` or `warning`. For a bus signal to cross a model boundary, the signal must be defined as a bus object for compatibility with higher level models that use a model as a reference model.	Set **Unspecified bus object at root Outport block** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `RootOutportRequireBusObject` to `error`.
The diagnostic that detects whether the name of a bus element matches the name specified by the corresponding bus object is set to `none` or `warning`. This diagnostic prevents the use of incompatible buses in a bus-capable block such that the output names are inconsistent.	Set **Element name mismatch** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `BusObjectLabelMismatch` to `error`.
The diagnostic that detects when some blocks treat a signal as a mux/vector, while other blocks treat the signal as a bus, is set to `none` or `warning`. When the Simulink software automatically converts a muxed signal to a bus, it is possible for an unintended operation or unpredictable behavior to occur.	Set **Bus signal treated as vector** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box to `error`, or the parameter `StrictBusMsg` to `ErrorOnBusTreatedAsVector`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to bus connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets

- EN 50128, Table A.4 (11) - Language Subset
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- `Simulink.Bus` in the Simulink reference documentation.
- “hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses” (Simulink)

Check safety-related diagnostic settings that apply to function-call connectivity

Check ID: `mathworks.do178.FcnCallDiagnosticsSet`

Check model configuration for diagnostic settings that apply to function-call connectivity and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to function-call connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects incorrect use of a function-call subsystem is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Invalid function-call connection** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `InvalidFcnCallConMsg` to `error`.
The diagnostic that specifies whether the Simulink software has to compute inputs of a function-call subsystem directly or indirectly while executing the subsystem is set to `Use local settings` or `Disable all`. This diagnostic detects unpredictable data coupling between a function-call subsystem and the inputs of the subsystem in the generated code.	Set **Context-dependent inputs** (Simulink) on the **Diagnostics** > **Connectivity** pane in the Configuration Parameters dialog box or set the parameter `FcnCallInpInsideContextMsg` to `Enable all as errors`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to function-call connectivity and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “Model Configuration Parameters: Connectivity Diagnostics” (Simulink)
- “hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls” (Simulink)

Check safety-related diagnostic settings for type conversions

Check ID: `mathworks.do178.TypeConversionDiagnosticsSet`

Check model configuration for diagnostic settings that apply to type conversions and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to type conversions are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects Data Type Conversion blocks when the type conversion is set to `none`. The Simulink software might remove unnecessary Data Type Conversion blocks from generated code, which might result in requirements without corresponding code. The removal of these blocks needs to be identified so model developers can explicitly remove the unnecessary blocks .	Set the **Unnecessary type conversions** (Simulink) Configuration Parameter or`UnnecessaryDatatypeConvMsg` parameter to `warning`.
The diagnostic that detects vector-to-matrix or matrix-to-vector conversions at block inputs is set to `none` or `warning`. When the Simulink software automatically converts between vector and matrix dimensions, unintended operations or unpredictable behavior can occur.	Set the **Vector/matrix block input conversion** (Simulink) Configuration Parameter or `VectorMatrixConversionMsg` parameter to `error`
The diagnostic that detects when a 32-bit integer value is converted to a floating-point value is set to `none`. This type of conversion can result in a loss of precision due to truncation of the least significant bits for large integer values.	Set the **32-bit integer to single precision float conversion** (Simulink) Configuration Parameter or `Int32ToFloatConvMsg` parameter to `warning`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to type conversions and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508–3, Table A.3 (2) Strongly typed programming language
  IEC 61508–3, Table A.4 (3) Defensive programming

- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
  ISO 26262-6, Table 1 (1c) Enforcement of strong typing
  ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques
- EN 50128, Table A.4 (8) Strongly Typed Programming Language
  EN 50128, Table A.3 (1) Defensive Programming
- “Model Configuration Parameters: Type Conversion Diagnostics” (Simulink)
- “hisl_0309: Configuration Parameters > Diagnostics > Type Conversion” (Simulink)

Check safety-related diagnostic settings for model referencing

Check ID: `mathworks.do178.MdlrefDiagnosticsSet`

Check model configuration for diagnostic settings that apply to model referencing and that can impact safety.

Description

This check verifies that model diagnostic configuration parameters pertaining to model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects a mismatch between the version of the model that creates or refreshes a Model block and the current version of the referenced model is set to `error` or `warning`. The detection occurs during load and update operations. When you get the latest version of the referenced model from the software configuration management system, rather than an older version that was used in a previous simulation, if this diagnostic is set to `error`, the simulation is aborted. If the diagnostic is set to `warning`, a warning message is issued. To resolve the issue, the user must resave the model being simulated, which may not be the desired action.	Set **Model block version mismatch** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceVersionMismatchMessage` to `none`.
The diagnostic that detects port and parameter mismatches during model loading and updating is set to `none` or `warning`. If undetected, such mismatches can lead to incorrect simulation results because the parent and referenced models have different interfaces.	Set **Port and parameter mismatch** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceIOMismatchMessage` to `error`.
The diagnostic that detects invalid internal connections to the current model's root-level Inport and Outport blocks is set to `none` or `warning`. When this condition is detected, the Simulink software might automatically insert hidden blocks into the model to fix the condition. The hidden blocks can result in generated code without traceable requirements. Setting the diagnostic to `error` forces model developers to fix the referenced models manually.	Set **Invalid root Inport/Outport block connection** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceIOMessage` to `error`.

Condition	Recommended Action
The diagnostic that detects whether To Workspace or Scope blocks are logging data in a referenced model is set to `none` or `warning`. Data logging is not supported for To Workspace and Scope blocks in referenced models.	Set **Unsupported data logging** (Simulink) on the **Diagnostics** > **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferenceDataLoggingMessage` to `error`. To log data, remove the blocks and log the referenced model signals. For more information, see "Logging Referenced Model Signals" (Simulink).

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to model referencing and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- "View Diagnostics" (Simulink)
- "Model Configuration Parameters: Model Referencing Diagnostics" (Simulink)
- "Logging Referenced Model Signals" (Simulink)
- "hisl_0310: Configuration Parameters > Diagnostics > Model Referencing" (Simulink)

Check safety-related model referencing settings

Check ID: `mathworks.do178.MdlrefOptSet`

Check model configuration for model referencing settings that can impact safety.

Description

This check verifies that model configuration parameters for model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The referenced model is configured such that its target is rebuilt whenever you update, simulate, or generate code for the model, or if the Simulink software detects changes in known dependencies. These configuration settings can result in unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references. See: IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'	Set **Rebuild** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `UpdateModelReferenceTargets` to `Never` or `If any changes detected`.

Condition	Recommended Action
The diagnostic that detects whether a target needs to be rebuilt is set to `None` or `Warn if targets require rebuild`. For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent. This diagnostic parameter is available only if **Rebuild** is set to `Never`. See: IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'	Set **Never rebuild diagnostic** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `CheckModelReferenceTargetMessage` to `error`.
The ability to pass scalar root input by value is off. This capability should be off because scalar values can change during a time step and result in unpredictable data. See: IEC 61508-3, Table A.3 (3) 'Language subset' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1b) 'Use of language subsets' ' EN 50128, Table A.4 (11) 'Language Subset'	Set **Pass fixed-size scalar root inputs by value for Real-Time Workshop** (Simulink) on the **Model Referencing** pane in the Configuration Parameters dialog box or set the parameter `ModelReferencePassRootInputsByReference` to `off`.

Condition	Recommended Action
The model is configured to minimize algebraic loop occurrences. This configuration is incompatible with the recommended setting of **Single output/update function** for embedded systems code. See: IEC 61508-3, Table A.3 (3) 'Language subset' IEC 62304, 5.5.3 - Software Unit acceptance criteria ISO 26262-6, Table 1 (1b) 'Use of language subsets' ' EN 50128, Table A.4 (11) 'Language Subset'	In the Configuration Parameters dialog box, set **Minimize algebraic loop occurrences** (Simulink) or set parameter `ModelReferenceMinAlgLoopOccurrences` to `off`.

Action Results

Clicking **Modify Settings** configures model referencing settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- “hisl_0037: Configuration Parameters > Model Referencing” (Simulink)
- “Analyze Model Dependencies” (Simulink)
- “Model Configuration Parameters: Model Referencing” (Simulink)

Check safety-related code generation settings

Check ID: `mathworks.do178.CodeSet`

Check model configuration for code generation settings that can impact safety.

Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The option to include comments in the generated code is cleared. Comments provide good traceability between the code and the model.	Select **Include comments** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `GenerateComments` to `on`.
The option to include comments that describe the code for blocks is cleared. Comments provide good traceability between the code and the model.	Select “Simulink block comments” (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `SimulinkBlockComments` to `on`.
The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments provide good traceability between the code and the model.	Select **Show eliminated blocks** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `ShowEliminatedStatement` to `on`.
The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in `*model*_prm.h` is cleared. Comments provide good traceability between the code and the model.	Select **Verbose comments for SimulinkGlobal storage class** (Simulink Coder) on the **Code Generation** > **Comments** pane in the Configuration Parameters dialog box or set the parameter `ForceParamTrailComments` to `on`.
The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments provide good traceability between the code and the model.	Select **Requirements in block comments** (Simulink Coder) on the **Code Generation** > **Custom comments** pane in the Configuration Parameters dialog box or set the parameter `ReqsInCode` to `on`.
The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems.	Clear **Support: non-finite numbers** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportNonFinite` to `off`.

Condition	Recommended Action
The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems.	Clear **Support: absolute time** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportAbsoluteTime` to `off`.
The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems.	Clear **Support: continuous time** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SupportContinuousTime` to `off`.
The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems.	Clear **Support: non-inlined S-functions** (Simulink Coder) in the Configuration Parameters dialog box or set the parameter `SupportNonInlinedSFcns` to `off`.
The option to generate model function calls compatible with the main program module of the pre-R2012a GRT target is selected. This option is inappropriate for real-time safety-related systems.	Clear **Classic call call interface** (Simulink Coder) on the **Code Generation** > **Interface**pane in the Configuration Parameters dialog box or set the parameter `GRTInterface` to `off`.
The option to generate the *model*`_update` function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code.	Select **Single output/update function** (Simulink Coder) on the **Code Generation** > **Interface**pane in the Configuration Parameters dialog box or set the parameter `CombineOutputUpdateFcns` to `on`.
The option to generate the *model*`_terminate` function is selected. This function deallocates dynamic memory, which is unsuitable for real-time safety-related systems.	Clear **Terminate function** (Simulink Coder) on the **Code Generation** pane in the Configuration Parameters dialog box or set the parameter `IncludeMdlTerminateFcn` to `off`.
The option to log or monitor error status is cleared. If you do not select this option, the Simulink Coder product generates extra code that might not be reachable for testing.	Select **Remove error status field in real-time model data structure** (Simulink Coder) on the **Code Generation** > **Interface** pane in the Configuration Parameters dialog box or set the parameter `SuppressErrorStatus` to `on`.

Condition	Recommended Action
MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses.	Clear **MAT-file logging** (Simulink Coder) in the Configuration Parameters dialog box or set the parameter `MatFileLogging` to `off`.
The option that specifies the style for parenthesis usage is set to `Minimum (Rely on C/C++ operators precedence)` or to `Nominal (Optimize for readability)`. For safety-related applications, explicitly specify precedence with parentheses.	Set parameter `ParenthesesLevel` to `Maximum (Specify precedence with parentheses)`.
The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code.	Set parameter `PreserveExpressionOrder` to `on`.
The option that specifies whether to preserve empty primary condition expressions in `if` statements is cleared. This option increases the traceability of the generated code.	Set parameter `PreserveIfCondition` to `on`.
The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews.	Set **Minimum mangle length** (Simulink Coder) on the **Code Generation** > **Symbols** pane in the Configuration Parameters dialog box or the parameter `MangleLength` to a value of 4 or greater.

Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
- EN 50128, Table A.4 (11) 'Language Subset'
- “hisl_0038: Configuration Parameters > Code Generation > Comments” (Simulink)
- “hisl_0039: Configuration Parameters > Code Generation > Interface” (Simulink)
- “hisl_0047: Configuration Parameters > Code Generation > Code Style” (Simulink)
- “hisl_0049: Configuration Parameters > Code Generation > Symbols” (Simulink)
- “Model Configuration Parameters: Code Generation Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Symbols” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Code Style” (Embedded Coder)

Check usage of shift operations for Stateflow data

Check ID: `mathworks.iec61508.hisf_0064`

Identify usage of shift operations for Stateflow data that might impact safety.

Description

This check inspects the shift operations that have shift operand values greater than the bit-width of the input or output type or a shift operand that has a negative value.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Right-shift operations are greater than the bit-width of the input type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.
Left-shift operations are greater than the bit-width of the output type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not support the shift operation that has the shift size defined as a Simulink signal or a variable.
- Does not support the shift operations that consist of shift size decided at run time.

See Also

- IEC 61508–3, Table A.3 (2) Strongly typed programming language
  IEC 61508–3, Table A.4 (3) Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
  ISO 26262-6, Table 1 (1c) Enforcement of strong typing
  ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques
- EN 50128, Table A.4 (8) Strongly Typed Programming Language
  EN 50128, Table A.3 (1) Defensive Programming
- hisf_0064: Shift operations for Stateflow data to improve code compliance

Check assignment operations in Stateflow Charts

Check ID: `mathworks.iec61508.hisf_0065`

Identify assignment operations in Stateflow objects.

Description

This check identifies the assignment operations in Stateflow objects that implicitly cast integer and fixed-point arithmetic calculations to wider data types than the input data types.

This check identifies only the assignments with arithmetic operations.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow object consists of assignment operations that cast integer and fixed-point calculations to wider data types than the input data types.	Explicitly replace assignment operator (=) to := operator in Stateflow objects.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508–3, Table A.3 (2) Strongly typed programming language
  IEC 61508–3, Table A.4 (3) Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
  ISO 26262-6, Table 1 (1c) Enforcement of strong typing
  ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques
- EN 50128, Table A.4 (8) Strongly Typed Programming Language
  EN 50128, Table A.3 (1) Defensive Programming
- DO-331 Section MB.6.3.1.b 'High-level requirements are accurate and consistent'
  DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
- hisf_0065: Type cast operations in Stateflow to improve code compliance
- “Assignment (=, :=) Operations” (Stateflow)

Check Stateflow charts for unary operators

Check ID: `mathworks.iec61508.hisf_0211`

Identify unary operators in Stateflow charts.

Description

This check identifies the unary minus operators on unsigned data types in Stateflow charts.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart consists of a unary minus operator on unsigned data types.	Explicitly modify the unary operator on unsigned data types. For more information, see “Unary Operations” (Stateflow).

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not flag expressions with bitwise and arithmetic operators. For example, `–(u1/u2)` is not flagged.

See Also

- IEC 61508–3, Table A.3 (2) Strongly typed programming language
  IEC 61508–3, Table A.4 (3) Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets
  ISO 26262-6, Table 1 (1c) Enforcement of strong typing
  ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques
- EN 50128, Table A.4 (8) Strongly Typed Programming Language
  EN 50128, Table A.3 (1) Defensive Programming
- hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

Check safety-related optimization settings for Loop unrolling threshold

Check ID: `mathworks.iec61508.hisl_0051`

Check optimization settings in the model configuration that apply to Loop unrolling threshold and might impact safety.

Description

This check verifies that the model optimization configuration parameters pertaining to the minimum signal or parameter width for which a `for` loop is generated is set optimally for generating code for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The optimization setting to specify the minimum signal or parameter width for which a `for` loop is generated is set to a value less than `2`.	In the Configuration Parameters dialog box, set “Loop unrolling threshold” (Simulink) or set the parameter `RollThreshold` to a value equal to or greater than `2`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

See Also

- IEC 61508-3, Table A.3 (3) Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) Use of language subsets

- EN 50128, Table A.4 (11) Language Subset
- MISRA C:2012, Rule 6.1
- “Loop unrolling threshold” (Simulink)
- “hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold” (Simulink)

Check safety-related diagnostic settings for saving

Check ID: `mathworks.do178.SavingDiagnosticsSet`

Check model configuration for diagnostic settings that apply to saving model files

Description

This check verifies that model configuration parameters are set optimally for saving a model for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains disabled library links before the model is saved is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Block diagram contains disabled library links** (Simulink) in the Configuration Parameters dialog box or set parameter `SaveWithDisabledLinkMsg` to `error`.
The diagnostic that detects whether a model contains library links that are using parameters not in a mask before the model is saved is set to `none` or `warning`. If this condition is undetected, incorrect code might be generated.	Set **Block diagram contains parameterized library links** (Simulink) in the Configuration Parameters dialog box or set parameter `SaveWithParameterizedLinkMsg` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to saving a model file.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- EN 50128, Table A.4 (11) 'Language Subset'
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation'
- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- “hisl_0036: Configuration Parameters > Diagnostics > Saving” (Simulink)
- “Identify disabled library links” (Simulink)
- “Save a Model” (Simulink)
- “Model Parameters” (Simulink)

Check safety-related diagnostic settings for Merge blocks

Check ID: `mathworks.iec61508.hisl_0303`

Check model configuration for diagnostic settings that apply to Merge blocks

Description

This check verifies that model configuration parameters are set optimally for Merge blocks for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains Merge blocks with more than one driving block executing at the same time step is set to `none` or `warning`. In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to `Classic`.	In the Configuration Parameters dialog box, set “Detect multiple driving blocks executing at the same time step” (Simulink) or set the parameter `MergeDetectMultiDrivingBlocksExec` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- EN 50128, Table A.4 (11) - Language Subset
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- IEC 61508-3, Table A.3 (3) - Language subset
- “hisl_0303: Configuration Parameters > Diagnostics > Merge block” (Simulink)
- “Detect multiple driving blocks executing at the same time step” (Simulink)
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)

Check safety-related diagnostic settings for Stateflow

Check ID: `mathworks.iec61508.hisl_0311`

Check safety-related diagnostic settings for Stateflow

Description

This check verifies that model configuration parameters are set optimally for Stateflow for a safety-related application.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a chart configuration leads to unwanted backtracking during simulation is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Unexpected backtracking” (Simulink) or set the parameter `SFUnexpectedBacktrackingDiag` to `error`.
The diagnostic that detects whether a chart configuration has blocks that connect to chart input ports do not initialize their outputs during initialization is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Invalid input data access in chart initialization” (Simulink) or set the parameter `SFInvalidInputDataAccessInChartInitDiag` to `error`.
The diagnostic that detects whether a chart has an unconditional default transition to a state or a junction is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “No unconditional default transitions” (Simulink) or set the parameter `SFNoUnconditionalDefaultTransitionDiag` to `error`.
The diagnostic that detects whether a chart contains a transition that loops outside of the parent state or junction is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Transition outside natural parent” (Simulink) or set the parameter `SFTransitionOutsideNaturalParentDiag` to `error`.
The diagnostic that detects whether a chart is constructed on a valid execution path is set to `none` or `warning`.	In the Configuration Parameters dialog box, set “Unreachable execution path” (Simulink) or set the parameter `SFUnreachableExecutionPathDiag` to `error`.

Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- EN 50128, Table A.4 (11) - Language Subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- IEC 61508-3, Table A.3 (3) - Language subset
- “hisl_0311: Configuration Parameters > Diagnostics > Stateflow” (Simulink)
- “Diagnostics Parameters: Stateflow” (Simulink)

Check MATLAB Code Analyzer messages

Check ID: `mathworks.iec61508.himl_0004`

Check MATLAB Functions for `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs.

Description

Verifies `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs for:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
For MATLAB code in MATLAB Function blocks, either of the following: • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.
For MATLAB functions defined in Stateflow charts, either of the following: • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.
For called MATLAB functions: • Code does not have the `%#codegen` directive. • Code lines are not justified with a `%#ok` comment. • Codes lines justified with `%#ok` do not specify a message id.	• Insert `%#codegen` directive in the MATLAB code. • Implement MATLAB Code Analyzer recommendations. • Justify not following MATLAB Code Analyzer recommendations with a `%#ok` comment. • Specify justified code lines with a message id. For example, `%#ok<NOPRT>`.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (3) - Defensive programming, Table A.4 (5) - Design and coding standards
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques, Table 1 (1e) - Use of established design principles, Table 1 (1f) - Use of unambiguous graphical representation, Table 1 (1g) - Use of style guides, Table 1 (1h) - Use of naming conventions
- EN 50128, Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming, Table A.12 (1) - Coding Standard, Table A.12 (2) - Coding Style Guide
- “Check Code for Errors and Warnings” (MATLAB)
- “himl_0004: MATLAB Code Analyzer recommendations for code generation” (Simulink)

Check MATLAB code for global variables

Check ID: `mathworks.iec61508.himl_0005`

Check for global variables in MATLAB code.

Description

Verifies that global variables are not used in any of the following:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Global variables are used in one or more of the following: • MATLAB code in MATLAB Function blocks • MATLAB functions defined in Stateflow charts • Called MATLAB functions	Replace global variables with signal lines, function arguments, or persistent data.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.3 (3) – Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “himl_0005: Usage of global variables in MATLAB functions” (Simulink)

Check usage of Math Operations blocks

Check ID: `mathworks.iec61508.MathOperationsBlocksUsage`

Identify usage of Math Operation blocks that might impact safety.

Description

This check inspects the usage of the following blocks:

- Abs

- Assignment
- Gain

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains an Absolute Value block that is operating on one of the following: • A boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code • A signed integer value with the **Saturate on integer overflow** check box not selected. For signed data types, the absolute value of the most negative value is problematic because it is not representable by the data type. This condition results in an overflow in the generated code.	If the identified Absolute Value block is operating on a boolean or unsigned data type, do one of the following: • Change the input of the Absolute Value block to a signed input type. • Remove the Absolute Value block from the model. If the identified Absolute Value block is operating on a signed data type, in the **Block Parameters > Signal Attributes** dialog box, select **Saturate on integer overflow**.
The model or subsystem contains Gain blocks with a of value 1 or an identity matrix.	If you are using Gain blocks as buffers, consider replacing them with Signal Conversion blocks.
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.	Set block parameter **Action if any output element is not assigned** to one of the recommended values: • **Error**, if Assignment block is not in an Iterator subsystem. • **Warning**, if Assignment block is in an Iterator subsystem.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (3) - Defensive programming, Table A.3 (2) - Strongly typed programming language, Table B.8 (3) - Control Flow Analysis
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques, Table 9 (1f) - Control flow analysis
- EN 50128, Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming, EN 50128, Table A.4 (8) - Strongly Typed Programming Language, Table A.19 (3) - Control Flow Analysis
- MISRA C:2012, Dir 4.1
- MISRA C:2012, Rule 9.1
- “hisl_0001: Usage of Abs block” (Simulink)
- “hisl_0029: Usage of Assignment blocks” (Simulink)

Check usage of Signal Routing blocks

Check ID: `mathworks.iec61508.SignalRoutingBlockUsage`

Identify usage of Signal Routing blocks that might impact safety.

Description

This check identifies model or subsystem Switch blocks that might generate code with inequality operations (~=) in expressions that contain a floating-point variable or constant.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a Switch block that might generate code with inequality operations (~=) in expressions where at least one side of the expression contains a floating-point variable or constant. The Switch block might cause floating-point inequality comparisons in the generated code.	For the identified block, do one of the following: • For the control input block, change the **Data type** parameter setting. • Change the Switch block **Criteria for passing first input** parameter setting. This might change the algorithm.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) – Language subset, Table A.4 (3) – Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques
- EN 50128, Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming
- MISRA C:2012, Dir 1.1

Check usage of Logic and Bit Operations blocks

Check ID: `mathworks.iec61508.LogicBlockUsage`

Identify usage of Logical Operator and Bit Operations blocks that might impact safety.

Description

This check inspects the usage of:

- Blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare To Zero, and Detect Change blocks
- Logical Operator blocks

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.	On the Block Parameters > Signal Attributes pane, set the **Output data type** to `boolean` for the specified blocks.
The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues. These operators can lead to unpredictable results in the generated code.	For the identified block, do one of the following: • Change the signal data type. • Rework the model to eliminate using == or ~= operators on floating-point signals.
The model or subsystem contains a Logical Operator block that has inputs or outputs that are not Boolean inputs or outputs. The block might result in floating-point equality or inequality comparisons in the generated code.	• Modify the Logical Operator block so that the inputs and outputs are Boolean. On the Block Parameters > Signal Attributes pane, consider selecting **Require all inputs to have the same data type** and setting **Output data type** to `boolean`. • In the Configuration Parameters dialog box, consider selecting the **Implement logic signals as boolean data (vs. double)**.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’
  IEC 61508-3, Table A.3 (3) 'Language subset’
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
  ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing'
- EN 50128, Table A.4 (11) 'Language Subset'
  EN 50128, Table A.4 (8) 'Strongly Typed Programming Language'
  EN 50128, Table A.3 (1) 'Defensive Programming'
- MISRA C:2012, Dir 1.1
- MISRA C:2012, Rule 10.1
- “hisl_0016: Usage of blocks that compute relational operators” (Simulink)
- “hisl_0017: Usage of blocks that compute relational operators (2)” (Simulink)
- “hisl_0018: Usage of Logical Operator block” (Simulink)

Check usage of Ports and Subsystems blocks

Check ID: `mathworks.iec61508.PortsSubsystemsUsage`

Identify usage of Ports and Subsystems blocks that might impact safety.

Description

This check inspects the usage of:

- For Iterator blocks
- While Iterator blocks
- If blocks
- Switch Case blocks

The check does not flag Switch Case blocks that do not use integer data types or enumeration values for inputs. To comply with “hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks" (Simulink) – C, use an integer data type or an enumeration value for the inputs to Switch Case blocks.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.	For the identified For Iterator blocks, do one of the following: • Set the **Iteration limit source** parameter to `internal`. • If the **Iteration limit source** parameter must be `external`, use a Constant, Probe, or Width block as the source. • Clear the Set next **i (iteration variable) externally** check box. • Consider selecting the **Show iteration variable** check box and observe the iteration value during simulation.
The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code. mo	For the identified While Iterator blocks: • Set the **Maximum number of iterations (-1 for unlimited)** parameter to a positive integer value. • Consider selecting the **Show iteration number port** check box and observe the iteration value during simulation.
The model or subsystem contains an If block with an If expression or Elseif expressions that might cause floating-point equality or inequality comparisons in generated code.	Modify the expressions in the If block to avoid floating-point equality or inequality comparisons in generated code.

Condition	Recommended Action
The model or subsystem contains an If block using Elseif expressions without an Else condition.	In the If block Block Parameters dialog box, select **Show else condition**. Connect the resulting Else output port to an If Action Subsystem block.
The model or subsystem contains an If block with output ports that do not connect to If Action Subsystem blocks.	Verify that output ports of the If block connect to If Action Subsystem blocks.
The model or subsystem contains an Switch Case block without a default case.	In the Switch Case block Block Parameters dialog box, select **Show default case**. Connect the resulting default output port to a Switch Case Action Subsystem block.
The model or subsystem contains a Switch Case block with an output port that does not connect to a Switch Case Action Subsystem block.	Verify that output ports of the Switch Case blocks connect to Switch Case Action Subsystem blocks.

Condition	Recommended Action
The model or subsystem contains one of the following time-dependent blocks in a For Iterator or While Iterator subsystem: • Discrete Filter • Discrete FIR Filter • Discrete State-Space • Discrete Transfer Fcn • Discrete Zero-Pole • Transfer Fcn First Order • Transfer Fcn Lead or Lag • Transfer Fnc Real Zero • Discrete Derivative • Discrete Transfer Fcn (with initial outputs) • Discrete Transfer Fcn (with initial states) • Discrete Zero-Pole (with initial outputs) • Discrete Zero-Pole (with initial states)	In the model or subsystem, consider removing the time-dependent blocks.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (3) - Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques

- EN 50128 - Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming
- MISRA C:2012, Rule 14.2
- MISRA C:2012, Rule 16.4
- MISRA C:2012, Dir 4.1
- “hisl_0006: Usage of While Iterator blocks” (Simulink)
- “hisl_0007: Usage of While Iterator subsystems” (Simulink)
- “hisl_0008: Usage of For Iterator Blocks” (Simulink)
- “hisl_0009: Usage of For Iterator Subsystem blocks” (Simulink)
- “hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks” (Simulink)

Display configuration management data

Check ID: `mathworks.iec61508.MdlVersionInfo`

Display model configuration and checksum information.

Description

This informer check displays the following information for the current model:

- Model version number
- Model author
- Date
- Model checksum

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Could not retrieve model version and checksum information.	This summary is provided for your information. No action is required.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- IEC 61508-3, Table A.8 (5) – Software configuration management
- IEC 62304-8 – Software configuration management process
- ISO 26262-8, Clause 7 – Configuration management
- EN 50128, Table A.9 (5) – Software Configuration Management
- “How Simulink Helps You Manage Model Versions” (Simulink) in the Simulink documentation
- `Model Change Log` in the Simulink Report Generator™ documentation
- `Simulink.BlockDiagram.getChecksum` in the Simulink documentation
- `Simulink.SubSystem.getChecksum` in the Simulink documentation

Check for blocks not recommended for MISRA C:2012

Check ID: `mathworks.misra.BlkSupport`

Identify blocks that are not supported or recommended for MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem.	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystemer. The deprecated Lookup Table blocks are Lookup and Lookup2D.	Consider replacing the deprecated Lookup Table blocks.

Condition	Recommended Action
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- "hisl_0020: Blocks not recommended for MISRA C:2012 compliance" (Simulink)
- "MISRA C Guidelines" (Embedded Coder)
- "MISRA C:2012 Compliance Considerations" (Simulink)
- "What Is a Model Advisor Exclusion?"

Check configuration parameters for MISRA C:2012

Check ID: `mathworks.misra.CodeGenSettings`

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model Verification block enabling is set to `Use local settings` or `Enable All`.	In the Configuration Parameters, set **Model Verification block enabling** to `Disable All`.
System target file is set to a GRT-based target.	In the Configuration Parameters dialog box, on the **Code Generation** pane, set **System target file** to an ERT-based target.
Code Generation > **Interface** parameters are not set to the recommended values.	In the Configuration Parameters dialog box: • Set **Code replacement library** to `None` or `AUTOSAR 4.0` • Set **Shared code placement** to `Shared location` • Clear **Support: non-finite numbers** • Clear **Support: continuous time** (ERT-based target only) • Clear **Support non-inlined S-functions** (ERT-based target only) • Clear **MAT-file logging**
Parenthesis level is not set to `Maximum (Specify precedence with parentheses)`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Parentheses level** to `Maximum (Specify precedence with parentheses)`.
Casting Modes is not set to `Standards Compliant`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Casting Modes** to `Standards Compliant`.
`GenerateSharedConstants` is set to `on`.	Use `get_param` to set `GenerateSharedConstants` to `off`.

Condition	Recommended Action
System-generated identifiers is set to `Classic`.	In the Configuration Parameters dialog box, on the **Code Generation > Symbols** pane, set **System-generated identifiers** to `Shortened`.
Pack Boolean data into bitfields is selected and **Bitfield declarator type specifier** is set to `uchar_T`.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, if **Pack Boolean data into bitfields** is selected, set **Bitfield declarator type specifier** to `uint_T`.
Signed integer division rounds to is not set to `Zero` or `Floor`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, set **Signed integer division rounds to** to `Zero` or `Floor`.
Use division for fixed-point net slope computation is not set to `On` or `Use division for reciprocals of integers only`.	In the Configuration Parameters dialog box, on the **Optimization** pane, set **Use division for fixed-point net slope computation** to `On` or `Use division for reciprocals of integers only`.
Replace multiplications by powers of two with signed bitwise shifts is selected.	In the Configuration Parameters dialog box, on the **Code Generation > Code Style** pane, clear **Replace multiplications by powers of two with signed bitwise shifts**.
Allow right shifts on signed integers is selected.	In the Configuration Parameters dialog box, on the **Code Generation > Code Style** pane, Clear **Allow right shifts on signed integers**.
Use dynamic memory allocation for model initialization is selected.	In the Configuration Parameters dialog box, clear **Use dynamic memory allocation for model initialization**.
Wrap on overflow is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Wrap on overflow** to `warning` or `error`.

Condition	Recommended Action
Inf or NaN block output is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Inf or NaN block output** to `warning` or `error`.
Dynamic momory allocation in MATLAB Function blocks is selected.	In the Configuration Parameters dialog box, clear **Dynamic memory allocation in MATLAB Function blocks**.
`ERTFilePackagingFormat` is set to `Modular`.	Use `get_param` to set `ERTFilePackagingFormat` to `CompactWithDataFile` or `Compact`. If you click **Modify** to automatically fix the parameter setting, the value is set to `Compact`.
`PreserveStaticInFcnDecls` is set to `off`.	Use `get_param` to set `PreserveStaticInFcnDecls` to `on`. To set this value, `ERTFilePackagingFormat` must be set to `CompactWithDataFile` or `Compact`.

Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

This check does not review referenced models.

See Also

- "hisl_0060: Configuration parameters that improve MISRA C:2012 compliance" (Simulink)
- "MISRA C Guidelines" (Embedded Coder)

- “MISRA C:2012 Compliance Considerations” (Simulink)

MathWorks Automotive Advisory Board Checks

In this section...

In this section...

MathWorks Automotive Advisory Board Checks

MathWorks Automotive Advisory Board (MAAB) checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks.

See Also

- "Run Model Checks" (Simulink)
- "Simulink Checks" (Simulink)
- "Simulink Coder Checks" (Simulink Coder)
- "MAAB Control Algorithm Modeling" (Simulink) guidelines
- The MathWorks Automotive Advisory Board on the MathWorks website, which lists downloads for the latest version of *Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow*

Check font formatting

Check ID: `mathworks.maab.db_0043`

Check for difference in font and font sizes.

Description

With the exception of free text annotations within a model, text elements, such as block names, block annotations, and signal labels, must have the same font style and font size. Select a font style and font size that is legible and portable (convertible between platforms), such as Arial or Times New Roman 12 point. To specify font rules for a Simulink session, from the Simulink editor select **Diagram** > **Format** > **Font Styles for Model**.

Available with Simulink Check.

Input Parameters

Font Name

Apply the specified font to all text elements. When you specify `Common` (default), the check identifies different fonts used in your model. Although you can specify other fonts, the fonts available from the drop-down list are `Arial`, `Courier New`, `Georgia`, `Times New Roman`, `Arial Black`, and `Verdana`.

Font Size

Apply the specified font size to all text elements. When you specify `Common` (default), the check identifies different font sizes used in your model. Although you can specify other font sizes, the font sizes available from the drop-down list are `6`, `8`, `9`, `10`, `12`, `14`, `16`.

Font Style

Apply the specified font style to all text elements. When you specify `Common` (default), the check identifies different font styles used in your model. The font styles available from the drop-down list are `normal`, `bold`, `italic`, and `bold italic`.

Results and Recommended Actions

Condition	Recommended Action
The fonts or font sizes for text elements in the model are not consistent or portable.	Specify values for the font parameters and in the right pane of the Model Advisor, click **Modify all Fonts**, or manually change the fonts and font sizes of text elements in the model so they are consistent and portable.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

Action Results

In the right pane of the Model Advisor, clicking **Modify all Fonts** changes the font and font size of all text elements in the model according to the values you specify in the input parameters.

For the input parameters, if you specify `Common`, clicking **Modify all Fonts** changes the font and font sizes of all text elements in the model to the most commonly used fonts, font sizes, or font styles.

See Also

- MAAB guideline, Version 3.0: db_0043: Simulink font and font size in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0043: Simulink font and font size.

Check transition orientations in flow charts

Check ID: `mathworks.maab.db_0132`

Check transition orientations in flow charts.

Description

The following rules apply to transitions in flow charts:

- Draw transition conditions horizontally.
- Draw transitions with a condition action vertically.
- Junctions in flow charts should have a default exit transition.
- Transitions in flow charts should not combine condition and action.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model includes a transition with a condition that is not drawn horizontally or a transition action that is not drawn vertically.	Modify the model.
Junction does not have a default exit transition	Add a default exit transition to the junction.
Transition has condition and action	Split up condition and action into separate transitions

Capabilities and Limitations

- MAAB guideline, Version 3.0 limitation: Although db_0132: Transitions in flow charts has an exception for loop constructs, the check does flag flow charts containing loop constructs if the transition violates the orientation rule.
- JMAAB guideline, Version 4.0 limitation: The check only flags flow charts containing loop constructs if the transition violates the orientation rule.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0132: Transitions in flow charts in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0132: Transitions in Flow Charts.

Check for nondefault block attributes

Check ID: `mathworks.maab.db_0140`

Identify blocks that use nondefault block parameter values that are not displayed in the model diagram.

Description

Model diagrams should display block parameters that have values other than default values. One way of displaying this information is by using the **Block Annotation** tab in the Block Properties dialog box. To automatically fix warnings associated with this check, see “Automatically Fix Display of Nondefault Block Parameters”.

To customize the list of nondefault block parameters that are flagged by the check, see “Customize Model Advisor Check for Nondefault Block Attributes”.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Block parameters that have values other than default values, and the values are not in the model display.	In the Block Properties dialog box, use the **Block Annotation** tab to add block parameter annotations.

Capabilities and Limitations

- Only customizable for block parameters in `IntrinsicDialogParameters`. See “Common Block Properties” (Simulink)
- JMAAB guideline, Version 4.0 limitation: The check flags masked blocks that display parameter information but do not use block annotations. JMAAB 4.0 guidelines allow masked blocks to display parameter information.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialog boxes.
- Allows exclusions of blocks and charts.

Tip

If you use the `add_block` function with `'built-in/`*`blocktype`*`'` as a source block path name for Simulink built-in blocks, some default parameter values of some blocks are different from the defaults that you get if you added those blocks interactively by using Simulink.

See Also

- MAAB guideline, Version 3.0: db_0140: Display of basic block parameters.
- JMAAB guideline, Version 4.0: db_0140: Display of block parameters.
- For a list of block parameter default values, see “Block-Specific Parameters” (Simulink).
- `add_block`.

Check signal line labels

Check ID: `mathworks.maab.na_0008`

Check the labeling on signal lines.

Description

Use a label to identify:

- Signals originating from the following blocks (the block icon exception noted below applies to all blocks listed, except Inport, Bus Selector, Demux, and Selector):
  Bus Selector block (tool forces labeling)
  Chart block (Stateflow)
  Constant block
  Data Store Read block
  Demux block
  From block
  Inport block
  Selector block
  Subsystem block

  Block Icon Exception If a signal label is visible in the display of the icon for the originating block, you do not have to display a label for the connected signal unless the signal label is required elsewhere due to a rule for signal destinations.
- Signals connected to one of the following destination blocks (directly or indirectly with a basic block that performs an operation that is not transformative):
  Bus Creator block
  Chart block (Stateflow)
  Data Store Write block
  Goto block
  Mux block
  Outport block
  Subsystem block
- Any signal of interest.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Signals coming from Bus Selector, Chart, Constant, Data Store Read, Demux, From, Inport, or Selector blocks are not labeled.	Label the signal.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: na_0008: Display of labels on signals in the Simulink documentation.
- JMAAB guideline, Version 4.0: na_0008: Display of labels on signals.
- “Signal Names and Labels” (Simulink) in the Simulink documentation.

Check for propagated signal labels

Check ID: `mathworks.maab.na_0009`

Check for propagated labels on signal lines.

Description

You should propagate a signal label from its source rather than enter the signal label explicitly (manually) if the signal originates from:

- An Inport block in a nested subsystem. However, if the nested subsystem is a library subsystem, you can explicitly label the signal coming from the Inport block to accommodate reuse of the library block.
- A basic block that performs a nontransformative operation.
- A Subsystem or Stateflow Chart block. However, if the connection originates from the output of an instance of the library block, you can explicitly label the signal to accommodate reuse of the library block.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model includes signal labels that were entered explicitly, but should be propagated.	Use the open angle bracket (<) character to mark signal labels that should be propagated and remove the labels that were entered explicitly.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: na_0009: Entry versus propagation of signal labels in the Simulink documentation.
- JMAAB guideline, Version 4.0: na_0009: Entry versus propagation of signal labels.
- “Signal Names and Labels” (Simulink) in the Simulink documentation.

Check default transition placement in Stateflow charts

Check ID: `mathworks.maab.jc_0531`

Check default transition placement in Stateflow charts.

Description

In a Stateflow chart, you should connect the default transition at the top of the state and place the destination state of the default transition above other states in the hierarchy. There should be only one default transition.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
There is no default transition.	Add a default transition.

Condition	Recommended Action
The default transition for a Stateflow chart is not connected at the top of the state.	Move the default transition to the top of the Stateflow chart.
The destination state of a Stateflow chart default transition is lower than other states in the same hierarchy.	Adjust the position of the default transition destination state so that the state is above other states in the same hierarchy.
There is more than one default transition.	Multiple default transitions should be combined into one default transition by using junctions and conditions.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0531: Placement of the default transition in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0531: Placement of the default transition.
- "Syntax for States and Transitions" (Stateflow)

Check return value assignments of graphical functions in Stateflow charts

Check ID: `mathworks.maab.jc_0511`

Identify graphical functions with multiple assignments of return values in Stateflow charts.

Description

The return value from a Stateflow graphical function must be set in only one place.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The return value from a Stateflow graphical function is assigned in multiple places.	Modify the specified graphical function so that its return value is set in one place.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0511: Setting the return value from a graphical function in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0511: Setting the return value from a graphical function.
- "When to Use Reusable Functions in Charts" (Stateflow) in the Stateflow documentation.

Check entry formatting in State blocks in Stateflow charts

Check ID: `mathworks.maab.jc_0501`

Identify missing line breaks between entry action (`en`), during action (`du`), and exit action (`ex`) entries in states. Identify missing line breaks after semicolons (`;`) in statements.

Description

Start a new line after the `entry`, `during`, and `exit` entries, and after the completion of a statement "`;`".

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
An `entry` (`en`) is not on a new line.	Add a new line after the `entry`.
A `during` (`du`) is not on a new line.	Add a new line after the `during`.
An `exit` (`ex`) is not on a new line.	Add a new line after the `exit`.
Multiple statements found on one line.	Add a new line after each statement.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

MAAB guideline, Version 3.0: jc_0501: Format of entries in a State block in the Simulink documentation.

Check usage of return values from a graphical function in Stateflow charts

Check ID: `mathworks.maab.jc_0521`

Identify calls to graphical functions in conditional expressions.

Description

Do not use the return value of a graphical function in a comparison operation.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Conditional expressions contain calls to graphical functions.	Assign return values of graphical functions to intermediate variables. Use these intermediate variables in the specified conditional expressions.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0521: Use of the return value from graphical functions in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0521: Use of the return value from graphical functions.
- "When to Use Reusable Functions in Charts" (Stateflow) in the Stateflow documentation.
- "Reuse Logic Patterns Using Graphical Functions" (Stateflow) in the Stateflow documentation.

Check for pointers in Stateflow charts

Check ID: `mathworks.maab.jm_0011`

Identify pointer operations on custom code variables.

Description

Pointers to custom code variables are not allowed.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Custom code variables use pointer operations.	Modify the specified chart to remove the dependency on pointer operations.

Capabilities and Limitations

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jm_0011: Pointers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 4.0: jm_0011: Pointers in Stateflow.

Check for event broadcasts in Stateflow charts

Check ID: `mathworks.maab.jm_0012`

Identify undirected event broadcasts that might cause recursion during simulation and generate inefficient code.

Description

Event broadcasts in Stateflow charts must be directed.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Event broadcasts are undirected.	Rearchitect the diagram to use directed event broadcasting. Use the send syntax or qualified event names to direct the event to a particular state. Use multiple send statements to direct an event to more than one state.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jm_0012: Event broadcasts in the Simulink documentation.
- JMAAB guideline, Version 4.0: jm_0012: Event broadcasts.
- "Broadcast Events to Synchronize States" (Stateflow) in the Stateflow documentation.

Check transition actions in Stateflow charts

Check ID: `mathworks.maab.db_0151`

Identify missing line breaks between transition actions.

Description

For readability, start each transition action on a new line.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Multiple transition actions are on a single line.	Verify that each transition action begins on a new line.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0151: State machine patterns for transition actions in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0151: State machine patterns for transition actions.
- "Syntax for States and Transitions" (Stateflow)

Check for MATLAB expressions in Stateflow charts

Check ID: `mathworks.maab.db_0127`

Identify Stateflow objects that use MATLAB expressions that are not suitable for code generation.

Description

Do not use MATLAB functions, instructions, and operators in Stateflow objects.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Stateflow objects use MATLAB expressions.	Replace MATLAB expressions in Stateflow objects.

Capabilities and Limitations

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0127: MATLAB commands in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0127: MATLAB commands in Stateflow.
- “Access Built-In MATLAB Functions and Workspace Data” (Stateflow) in the Stateflow documentation.

Check for indexing in blocks

Check ID: `mathworks.maab.db_0112`

Check that blocks use consistent vector indexing.

Description

Check that blocks use consistent vector indexing. When possible, use zero-based indexing to improve code efficiency.

Available with Simulink Check.

The check verifies consistent indexing for the following objects:

Object	Indexing
• Assignment block • For Iterator block • Find block • Multiport Switch block • Selector block	• Zero-based indexing ([0, 1, 2, ...]) • One-based indexing ([1, 2, 3,...])

Object	Indexing
• Stateflow charts with C action language	Zero-based indexing ([0, 1, 2, ...])
• MATLAB Function block • Fcn block • MATLAB System blocks • Truth tables • State transition tables • Stateflow charts with MATLAB action language • MATLAB functions inside Stateflow charts	One-based indexing ([1, 2, 3,...])

Results and Recommended Actions

Condition	Recommended Action
Objects in your model use one-based indexing, but can be configured for zero-based indexing.	Configure objects for zero-based indexing.
Objects in your model use inconsistent indexing.	If possible, configure objects for zero-based indexing. If your model contains objects that cannot be configured for zero-based indexing, configure objects for one-based indexing.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0112: Indexing in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0112: Indexing.

Check file names

Check ID: `mathworks.maab.ar_0001`

Checks the names of all files residing in the same folder as the model

Description

A file name conforms to constraints.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The file name contains illegal characters.	Rename the file. Allowed characters are a–z, A–Z, 0–9. and underscore (_).
The file name starts with a number.	Rename the file.
The file name starts with an underscore ("_").	Rename the file.
The file name ends with an underscore ("_").	Rename the file.
The file extension contains one (or more) underscores.	Change the file extension.
The file name has consecutive underscores.	Rename the file.
The file name contains more than one dot (".").	Rename the file.

Capabilities and Limitations

- MAAB guideline, Version 3.0 limitation: The check does not flag conflicts with C++ keywords.
- Runs on library models.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: ar_0001: Filenames in the Simulink documentation.

- JMAAB guideline, Version 4.0: ar_0001: Usable characters for filenames.

Check folder names

Check ID: `mathworks.maab.ar_0002`

Checks model directory and subdirectory names for invalid characters.

Description

A directory name conforms to constraints.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The directory name contains illegal characters.	Rename the directory. Allowed characters are a–z, A–Z, 0–9. and underscore (_).
The directory name starts with a number.	Rename the directory.
The directory name starts with an underscore ("_").	Rename the directory.
The directory name ends with an underscore ("_").	Rename the directory.
The directory name has consecutive underscores.	Rename the directory.

Capabilities and Limitations

- Runs on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes the full path of the model.
- Analyzes subdirectories in the same directory as the model.

See Also

- MAAB guideline, Version 3.0: ar_0002: Directory names in the Simulink documentation.

- JMAAB guideline, Version 4.0: ar_0002: Usable characters for folder names.

Check for prohibited blocks in discrete controllers

Check ID: `mathworks.maab.jm_0001`

Check for prohibited blocks in discrete controllers.

Description

The check identifies continuous blocks in discrete controller models.

Available with Simulink Check.

Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

1. Open the Model Configuration Editor and navigate to **Check for prohibited blocks in discrete controllers**.
2. In the **Input Parameters** pane, to:
   - Prohibit the blocks as specified in MAAB 3.0, from **Standard**, select `MAAB 3.0`. The **Block type list** table provides the blocks that MAAB 3.0 prohibits inside controllers.
   - To specify blocks to either allow or prohibit, from **Standard**, select `Custom`. In **Treat blocktype list as**, select `Allowed` or `Prohibited`. In the **Block type list** table, you can add or remove blocks.
3. Click **Apply**.
4. Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

Results and Recommended Actions

Condition	Recommended Action
Continuous blocks — Derivative, Integrator, State-Space, Transfer Fcn, Transfer Delay, Variable Time Delay, Variable Transport Delay, and Zero-Pole — are not permitted in models representing discrete controllers.	Replace continuous blocks with the equivalent blocks discretized in the s-domain. Use the Discretizing library, as described in “Discretize Blocks from the Simulink Model” (Simulink) in the Simulink documentation.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jm_0001: Prohibited Simulink standard blocks inside controllers in the Simulink documentation.
- JMAAB guideline, Version 4.0: jm_0001: Prohibited Simulink standard blocks inside controllers.
- “Overview of the Model Advisor Configuration Editor”

Check for prohibited sink blocks

Check ID: `mathworks.maab.hd_0001`

Check for prohibited Simulink sink blocks.

Description

You must design controller models from discrete blocks. Sink blocks, such as the Scope block, are not allowed in controller models.

Available with Simulink Check.

Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

1. Open the Model Configuration Editor and navigate to **Check for prohibited sink blocks**.
2. In the **Input Parameters** pane, to:
   - Prohibit the blocks as specified by MAAB 3.0, from **Standard**, select `MAAB 3.0`. The **Block type list** table provides the sink blocks that MAAB 3.0 prohibits.
   - To specify blocks to either allow or prohibit, from **Standard**, select `Custom`. In **Treat blocktype list as**, select `Allowed` or `Prohibited`. In the **Block type list** table, you can add or remove blocks.
3. Click **Apply**.
4. Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

Results and Recommended Actions

Condition	Recommended Action
Sink blocks are not permitted in discrete controllers.	Remove sink blocks from the model.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: hd_0001: Prohibited Simulink sinks in the Simulink documentation.
- JMAAB guideline, Version 4.0: hd_0001: Prohibited Simulink sinks.
- “Overview of the Model Advisor Configuration Editor”

Check positioning and configuration of ports

Check ID: `mathworks.maab.db_0042`

Check whether the model contains ports with invalid position and configuration.

Description

In models, ports must comply with the following rules:

- Place Inport blocks on the left side of the diagram. It is acceptable to move the Inport block to the right only to prevent signal crossings.
- Place Outport blocks on the right side of the diagram. It is acceptable to move the Outport block to the left only to prevent signal crossings.
- Avoid using duplicate Inport blocks at the subsystem level if possible.
- Do not use duplicate Inport blocks at the root level.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Inport blocks are too far to the right and result in left-flowing signals.	Move the specified Inport blocks to the left.
Outport blocks are too far to the left and result in left-flowing signals.	Move the specified Output blocks to the right.
Ports do not have the default orientation.	Modify the model diagram such that signal lines for output ports enter the side of the block and signal lines for input ports exit the right side of the block.
Ports are duplicate Inport blocks.	• If the duplicate Inport blocks are in a subsystem, remove them where possible. • If the duplicate Inport blocks are at the root level, remove them.

Capabilities and Limitations

- Runs on library models.

- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Does not analyze signal crossings

See Also

- MAAB guideline, Version 3.0: db_0042: Port block in Simulink models in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0042: Port block in Simulink models.

Check for matching port and signal names

Check ID: `mathworks.maab.jm_0010`

Check for mismatches between names of ports and corresponding signals.

Description

Use matching names for ports and their corresponding signals.

Available with Simulink Check.

Prerequisite

Prerequisite MAAB guidelines, Version 3.0, for this check are:

- db_0042: Port block in Simulink models
- na_0005: Port block name visibility in Simulink models

Results and Recommended Actions

Condition	Recommended Action
Ports have names that differ from their corresponding signals.	Change the port name or the signal name to match the name for the signal.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jm_0010: Port block names in Simulink models in the Simulink documentation.

Check whether block names appear below blocks

Check ID: `mathworks.maab.db_0142`

Check whether block names appear below blocks.

Description

If shown, the name of the block should appear below the block.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Blocks have names that do not appear below the blocks.	Set the name of the block to appear below the blocks.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0142: Position of block names in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0142: Position of block names.

Check for mixing basic blocks and subsystems

Check ID: `mathworks.maab.db_0143`

Check for systems that mix primitive blocks and subsystems.

Description

You must design each level of a model with building blocks of the same type, for example, only subsystems or only primitive (basic) blocks. If you mask your subsystem and set MaskType to a nonempty string, the Model Advisor treats the subsystem as a basic block.

Available with Simulink Check.

Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

1. Open the Model Configuration Editor and navigate to **Check for mixing basic blocks and subsystems**.
2. In the **Input Parameters** pane, to:
   - Allow the blocks specified by MAAB 3.0, from **Standard**, select `MAAB 3.0`. The **Block type list** table provides the blocks that MAAB 3.0 allows at any model level.
   - To specify blocks to either allow or prohibit, from **Standard**, select `Custom`. In **Treat blocktype list as**, select `Allowed` or `Prohibited`. In the **Block type list** table, you can add or remove blocks.
3. Click **Apply**.
4. Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

Results and Recommended Actions

Condition	Recommended Action
A level in the model includes subsystem blocks and primitive blocks.	Move nonvirtual blocks into the subsystem.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0143: Similar block types on the model levels in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0143: Similar block types on the model levels.
- “Overview of the Model Advisor Configuration Editor”

Check for unconnected ports and signal lines

Check ID: `mathworks.maab.db_0081`

Check whether model has unconnected input ports, output ports, or signal lines.

Description

Unconnected inputs should be connected to ground blocks. Unconnected outputs should be connected to terminator blocks.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Blocks have unconnected inputs or outputs.	Connect unconnected lines to blocks specified by the design or to Ground or Terminator blocks.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0081: Unconnected signals, block inputs and block outputs in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0081: Unconnected signals, block inputs and block outputs.

Check position of Trigger and Enable blocks

Check ID: `mathworks.maab.db_0146`

Check the position of Trigger and Enable blocks.

Description

Locate blocks that define subsystems as conditional or iterative at the top of the subsystem diagram.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Trigger, Enable, and Action Port blocks are not at the top of the subsystem diagram.	Move the Trigger, Enable, and Action Port blocks to the top of the subsystem diagram.

Capabilities and Limitations

- JMAAB guideline, Version 4.0 limitation: The check does not verify that For Each or For Iterator blocks are uniformly located.
- Runs on library models.
- Analyzes content of library linked blocks.
- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0146: Triggered, enabled, conditional Subsystems in the Simulink documentation.

- JMAAB guideline, Version 4.0: db_0146: Triggered, enabled, conditional Subsystems.

Check usage of tunable parameters in blocks

Check ID: `mathworks.maab.db_0110`

Check whether tunable parameters specify expressions, data type conversions, or indexing operations.

Description

To make a parameter tunable, you must enter the basic block without the use of MATLAB calculations or scripting. For example, omit:

- Expressions
- Data type conversions
- Selections of rows or columns

Supported blocks include:

- Backlash
- Bias
- Combinatorial Logic
- Constant
- Dead Zone
- Derivative
- Discrete-Time Integrator
- Gain
- Hit Crossing
- Initial Condition (IC)
- Integrator
- n-D Lookup Table
- Magnitude-Angle to Complex
- Memory
- Permute Dimensions

- Quantizer
- Rate Limiter
- Rate Transition
- Real-Imag to Complex
- Relay
- Saturation
- Sine
- State-Space
- Switch
- Transport Delay
- Unit Delay
- Variable Transport Delay

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Blocks have a tunable parameter that specifies an expression, data type conversion, or indexing operation.	In each case, move the calculation outside of the block, for example, by performing the calculation with a series of Simulink blocks, or precompute the value as a new variable.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not evaluate mask parameters.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0110: Tunable parameters in basic blocks in the Simulink documentation.

- JMAAB guideline, Version 4.0: db_0110: Tunable parameters in basic blocks.

Check Stateflow data objects with local scope

Check ID: `mathworks.maab.db_0125`

Check whether Stateflow data objects with local scope are defined at the chart level or below.

Description

This check flags Stateflow data whose local scope is not defined at the Chart level or below, regardless of whether the data is used or not.

You must define local data of a Stateflow block on the chart level or below in the object hierarchy. You cannot define local variables on the machine level; however, parameters and constants are allowed at the machine level.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Local data is not defined in the Stateflow hierarchy at the chart level or below.	Define local data at the chart level or below.

Capabilities and Limitations

- JMAAB guideline, Version 4.0 limitation: The check does not detect if local data has the same name within charts or states that have parent-child relationships.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: db_0125: Scope of internal signals and local auxiliary variables in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0125: Scope of internal signals and local auxiliary variables.

Check for Strong Data Typing with Simulink I/O

Check ID: `mathworks.maab.db_0122`

Check whether labeled Stateflow and Simulink input and output signals are strongly typed.

Description

Strong data typing between Stateflow and Simulink input and output signals is required.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
A Stateflow chart does not use strong data typing with Simulink.	Select the **Use Strong Data Typing with Simulink I/O** check box for the specified block.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks and charts.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: db_0122: Stateflow and Simulink interface signals and parameters in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0122: Stateflow and Simulink interface signals and parameters.
- "Syntax for States and Transitions" (Stateflow)

Check usage of exclusive and default states in state machines

Check ID: `mathworks.maab.db_0137`

Check states in state machines.

Description

In state machines:

- There must be at least two exclusive states.
- A state cannot have only one substate.
- The initial state of a hierarchical level with exclusive states is clearly defined by a default transition.

Available with Simulink Check.

Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is db_0149: Flow chart patterns for condition actions.

Results and Recommended Actions

Condition	Recommended Action
A system is underspecified.	Validate that the intended design is represented in the Stateflow diagram.
Chart has only one exclusive (OR) state.	Make the state a parallel state, or add another exclusive (OR) state.
Chart does not have a default state defined.	Define a default state.
Chart has multiple default states defined.	Define only one default state. Make the others nondefault.
State has only one exclusive (OR) substate.	Make the state a parallel state, add another exclusive (OR) state, or replace the state with a flow chart.
State does not have a default substate defined.	Define a default substate.
State has multiple default substates defined.	Define only one default substate, make the others nondefault.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

MAAB guideline, Version 3.0: db_0137: States in state machines in the Simulink documentation.

Check Implement logic signals as Boolean data (vs. double)

Check ID: `mathworks.maab.jc_0011`

Check the optimization parameter for Boolean data types.

Description

Optimization for Boolean data types is required

Available with Simulink Check.

Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is na_0002: Appropriate implementation of fundamental logical and numerical operations.

Results and Recommended Actions

Condition	Recommended Action
Configuration setting for **Implement logic signals as boolean data (vs. double)** is not set.	Select the **Implement logic signals as boolean data (vs. double)** check box in the Configuration Parameters dialog box.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: jc_0011: Optimization parameters for Boolean data types in the Simulink documentation.

- JMAAB guideline, Version 4.0: jc_0011: Optimization parameters for Boolean data types.

Check model diagnostic parameters

Check ID: `mathworks.maab.jc_0021`

Check the model diagnostics configuration parameter settings.

Description

You should enable the following diagnostics:
Algebraic loop
Minimize algebraic loop
Inf or NaN block output
Duplicate data store names
Unconnected block input ports
Unconnected block output ports
Unconnected line
Unspecified bus object at root Outport block
Element name mismatch
Invalid function-call connection

Diagnostics not listed in the Results and Recommended Actions section below can be set to any value.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Algebraic loop is set to `none`.	Set **Algebraic loop** on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box to `error` or `warning`. Otherwise, Simulink might attempt to automatically break the algebraic loops, which can impact the execution order of the blocks.

Condition	Recommended Action
Minimize algebraic loop is set to `none`.	Set **Minimize algebraic loop** on the **Diagnostics** > **Solver** pane in the Configuration Parameters dialog box to `error` or `warning`. Otherwise, Simulink might attempt to automatically break the algebraic loops for reference models and atomic subsystems, which can impact the execution order for those models or subsystems.
Inf or NaN block output is set to `none`, which can result in numerical exceptions in the generated code.	Set **Inf or NaN block output** on the **Diagnostics** > **Data Validity** > **Signals** pane in the Configuration Parameters dialog box to `error` or `warning`.
Duplicate data store names is set to `none`, which can result in nonunique variable naming in the generated code.	Set **Duplicate data store names** on the **Diagnostics** > **Data Validity** > **Signals** pane in the Configuration Parameters dialog box to `error` or `warning`.
Unconnected block input ports is set to `none`, which prevents code generation.	Set **Unconnected block input ports** on the **Diagnostics** > **Data Validity** > **Signals** pane in the Configuration Parameters dialog box to `error` or `warning`.
Unconnected block output ports is set to `none`, which can lead to dead code.	Set **Unconnected block output ports** on the **Diagnostics** > **Data Validity** > **Signals** pane in the Configuration Parameters dialog box to `error` or `warning`.
Unconnected line is set to `none`, which prevents code generation.	Set **Unconnected line** on the **Diagnostics** > **Connectivity** > **Signals** pane in the Configuration Parameters dialog box to `error` or `warning`.
Unspecified bus object at root Outport block is set to `none`, which can lead to an unspecified interface if the model is referenced from another model.	Set **Unspecified bus object at root Outport block** on the **Diagnostics** > **Connectivity** > **Buses** pane in the Configuration Parameters dialog box to `error` or `warning`.

Condition	Recommended Action
Element name mismatch is set to `none`, which can lead to an unintended interface in the generated code.	Set **Element name mismatch** on the **Diagnostics** > **Connectivity** > **Buses** pane in the Configuration Parameters dialog box to `error` or `warning`.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: jc_0021: Model diagnostic settings in the Simulink documentation.

Check the display attributes of block names

Check ID: `mathworks.maab.jc_0061`

Check the display attributes of subsystem and block names.

Description

When the subsystem and block names provide descriptive information, display the names. If the block function is known from its appearance, do not display the name. Blocks with names that are obvious from the block appearance:

- From
- Goto
- Ground
- Logic
- MinMax
- ModelReference
- MultiPortSwitch
- Product
- Relational Operator

- Saturate
- Switch
- Terminator
- Trigonometry
- Unit Delay
- Sum
- Compare To Constant
- Compare To Zero

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Name is displayed and obvious from the block appearance.	Hide name by clearing **Diagram** > **Format** > **Show Block Name**.
Name is not descriptive. Specifically, the block name is: • Not obvious from the block appearance. • The default name appended with an integer.	Modify the name to be more descriptive or hide the name by clearing **Diagram** > **Format** > **Show Block Name**.
Name is descriptive and not displayed. Descriptive names are: • Provided for blocks that are not obvious from the block appearance. • Not a default name appended with an integer.	Display the name by selecting **Diagram** > **Format** > **Show Block Name**

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0061: Display of block names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0061: Display of block names.

Check display for port blocks

Check ID: `mathworks.maab.jc_0081`

Check the **Icon display** setting for Inport and Outport blocks.

Description

The **Icon display** setting is required.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The **Icon display** setting is not set.	Set the **Icon display** to `Port number` for the specified Inport and Outport blocks.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

MAAB guideline, Version 3.0: jc_0081: Icon display for Port block in the Simulink documentation.

Check subsystem names

Check ID: `mathworks.maab.jc_0201`

Check whether subsystem block names include invalid characters.

Description

The names of all subsystem blocks that generate code are checked for invalid characters.

The check does not report invalid characters in subsystem names for:

- Virtual subsystems
- Atomic subsystems with **Function Packaging** set to `Inline`

Available with Simulink Check.

Input Parameters

To control the naming convention for blocks that the check flags, you can use the Model Advisor Configuration Editor.

1. Open the Model Configuration Editor and navigate to **Check port block names**. In the **Input Parameter** pane:

   - Use **Naming standard** to select `MAAB 3.0` or `Custom`. When you select `MAAB 3.0`, the check uses the regular expression `([^a-zA-Z_0-9])|(^\d)|(^ )|(__)|(^_)|(_$)` to verify that names:

     - Use these characters: `a-z`, `A-Z`, `0-9`, and the underscore `(_)`.
     - Do not start with a number.
     - Do not use underscores at the beginning or end of a string.
     - Do not use more than one consecutive underscore.

     When you select `Custom`, you can enter your own **Regular expression for prohibited names**. For example, if you want to allow more than one consecutive underscore, enter `([^a-zA-Z_0-9])|(^\d)|(^ )|(^_)|(_$)`.
2. Click **Apply**.
3. Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

Results and Recommended Actions

Condition	Recommended Action
The subsystem names do not comply with the naming standard specified in the input parameters.	Update the subsystem names to comply with your own guidelines or the MAAB guidelines.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

Tips

Use underscores to separate parts of a subsystem name instead of spaces.

See Also

- MAAB guideline, Version 3.0: jc_0201: Usable characters for Subsystem names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0201: Usable characters for Subsystem names.

Check port block names

Check ID: `mathworks.maab.jc_0211`

Check whether Inport and Outport block names include invalid characters.

Description

The names of all Inport and Outport blocks are checked for invalid characters.

Available with Simulink Check.

Input Parameters

To control the naming convention for blocks that the check flags, you can use the Model Advisor Configuration Editor.

1. Open the Model Configuration Editor and navigate to **Check port block names**. In the **Input Parameter** pane:

   - Use **Naming standard** to select `MAAB 3.0` or `Custom`. When you select `MAAB 3.0`, the check uses the regular expression `([^a-zA-Z_0-9])|(^\d)|(^ )|(__)|(^_)|(_$)` to verify that names:

     - Use these characters: `a-z`, `A-Z`, `0-9`, and the underscore `(_)`.
     - Do not start with a number.
     - Do not use underscores at the beginning or end of a string.
     - Do not use more than one consecutive underscore.

     When you select `Custom`, you can enter your own **Regular expression for prohibited names**. For example, if you want to allow more than one consecutive underscore, enter `([^a-zA-Z_0-9])|(^\d)|(^ )|(^_)|(_$)`.
2. Click **Apply**.
3. Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

Results and Recommended Actions

Condition	Recommended Action
The block names do not comply with the naming standard specified in the input parameters.	Update the block names to comply with your own guidelines or the MAAB guidelines.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

Tips

Use underscores to separate parts of a block name instead of spaces.

See Also

- MAAB guideline, Version 3.0: jc_0211: Usable characters for Inport blocks and Outport blocks in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0211: Usable characters for Inport block and Outport block.

Check character usage in signal labels

Check ID: `mathworks.maab.jc_0221`

Check whether signal line names include invalid characters.

Description

The names of all signal lines are checked for invalid characters.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The signal line name contains illegal characters.	Rename the signal line. Allowed characters include a–z, A–Z, 0–9, and underscore (_).
The signal line name starts with a number.	Rename the signal line.
The signal line name starts with an underscore ("_").	Rename the signal line.
The signal line name ends with an underscore ("_").	Rename the signal line.
The signal line name has consecutive underscores.	Rename the signal line.
The signal line name has blank spaces.	Rename the signal line.
The signal line name has control characters.	Rename the signal line.

Capabilities and Limitations

- Runs on library models.

- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

Tips

Use underscores to separate parts of a signal line name instead of spaces.

See Also

- MAAB guideline, Version 3.0: jc_0221: Usable characters for signal line names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0222: Usable characters for signal line and bus names.

Check character usage in block names

Check ID: `mathworks.maab.jc_0231`

Check whether block names include invalid characters.

Description

The check reports invalid characters in all block names, except:

- Inports and Outports
- Unmasked subsystems

MAAB guideline, Version 3.0, jc_0231: Usable characters for block names does not apply to subsystem blocks.

Available with Simulink Check.

Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is jc_0201: Usable characters for Subsystem names.

Input Parameters

To control the naming convention for blocks that the check flags, you can use the Model Advisor Configuration Editor.

1 Open the Model Configuration Editor and navigate to **Check character usage in block names**. In the **Input Parameter** pane:

   - Use **Naming standard** to select `MAAB 3.0` or `Custom`. When you select `MAAB 3.0`, the check uses the regular expression `([^a-zA-Z_0-9\n\r ])|(^\d)|(^ )` to verify that names:

     - Use these characters: `a-z`, `A-Z`, `0-9`, underscore (_), and blank space.
     - Do not start with a number or blank space.
     - Do not have double byte characters.

   When you select `Custom`, you can enter your own **Regular expression for prohibited names**. For example, if you do not want to allow underscores (_) in a block name, enter `([^a-zA-Z0-9\r])|(^\d)|(^ )`.

2 Click **Apply**.

3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

Results and Recommended Actions

Condition	Recommended Action
The block names do not comply with the naming standard specified in the input parameters.	Update the block names to comply with your own guidelines or the MAAB guidelines.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

Tips

Carriage returns are allowed in block names.

See Also

- MAAB guideline, Version 3.0: jc_0231: Usable characters for block names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0231: Usable characters for block names.

Check Trigger and Enable block names

Check ID: `mathworks.maab.jc_0281`

Check Trigger and Enable block port names.

Description

Block port names should match the name of the signal triggering the subsystem. The check does not flag Trigger or Enable block names if the associated signal does not have a label.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Trigger block does not match the name of the signal to which it is connected.	Match Trigger block names to the connecting signal.
Enable block does not match the name of the signal to which it is connected.	Match Enable block names to the connecting signal.

Capabilities and Limitations

- JMAAB guideline, Version 4.0 limitation: This check only flags Trigger and Enable blocks names.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0281: Naming of Trigger Port block and Enable Port block in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0281: Naming of Trigger Port block and Enable Port block.

Check for Simulink diagrams using nonstandard display attributes

Check ID: `mathworks.maab.na_0004`

Check model appearance setting attributes.

Description

Model appearance settings are required to conform to the guidelines when the model is released.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The toolbar is not visible.	Select **View** > **Toolbar**.
Wide Nonscalar Lines is cleared.	Select **Display** > **Signals & Ports** > **Wide Nonscalar Lines**.
Viewer Indicators is cleared.	Select **Display** > **Signals & Ports** > **Viewer Indicators**.
Testpoint Indicators is cleared.	Select **Display** > **Signals & Ports** > **Testpoint & Logging Indicators**.
Port Data Types is selected.	Clear **Display** > **Signals & Ports** > **Port Data Types**.
Storage Class is selected.	Clear **Display** > **Signals & Ports** > **Storage Class**.
Signal Dimensions is selected.	Clear **Display** > **Signals & Ports** > **Signal Dimensions**.
Model Browser is selected.	Clear **View** > **Model Browser** > **Show Model Browser**.

Condition	Recommended Action
Sorted Execution Order is selected.	Clear **Display** > **Blocks** > **Sorted Execution Order**.
Model Block Version is selected.	Clear **Display** > **Blocks** > **Block Version for Referenced Models**.
Model Block I/O Mismatch is selected.	Clear **Display** > **Blocks** > **Block I/O Mismatch for Referenced Models**.
Library Links is set to **Disabled**, **User Defined** or **All**.	Select **Display** > **Library Links** > **None**.
Linearization Indicators is cleared.	Select **Display** > **Signals & Ports** > **Linearization Indicators**.
Block backgrounds are not white.	Blocks should have black foregrounds with white backgrounds. Click the specified block and select **Format** > **Foreground Color** > **Black** and **Format** > **Background Color** > **White**.
Diagrams do not have white backgrounds.	Select **Diagram** > **Format** > **Canvas Color** > **White**.
Diagrams do not have zoom factor set to 100%.	Select **View** > **Zoom** > **Normal (100%)**.

Action Results

Clicking **Modify** updates the display attributes to conform to the guideline.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

See Also

- MAAB guideline, Version 3.0: na_0004: Simulink model appearance in the Simulink documentation.

- JMAAB guideline, Version 4.0: na_0004: Simulink model appearance.

Check MATLAB code for global variables

Check ID: `mathworks.maab.na_0024`

Check for global variables in MATLAB code.

Description

Verifies that global variables are not used in any of the following:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Global variables are used in one or more of the following: • MATLAB code in MATLAB Function blocks • MATLAB functions defined in Stateflow charts • Called MATLAB functions	Replace global variables with signal lines, function arguments, or persistent data.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

See Also

MAAB guideline, Version 3.0: na_0024: Global Variables in the Simulink documentation.

- MAAB guideline, Version 3.0: na_0024: Global Variables in the Simulink documentation.
- JMAAB guideline, Version 4.0: na_0024: Global variable.

Check visibility of block port names

Check ID: `mathworks.maab.na_0005`

Check the visibility of port block names.

Description

An organization applying the MAAB guideline, Version 3.0, must select one of the following alternatives to enforce:

- The names of port blocks are not hidden.
- The name of port blocks must be hidden.

Available with Simulink Check.

Input Parameters

All Port names should be shown (Format/Show Name)

Select this check box if all ports should show the name, including subsystems.

Results and Recommended Actions

Condition	Recommended Action
Blocks do not show their name and the **All Port names should be shown (Format/Show Name)** check box is selected.	Change the format of the specified blocks to show names according to the input requirement.
Blocks show their name and the **All Port names should be shown (Format/Show Name)** check box is cleared.	Change the format of the specified blocks to hide names according to the input requirement.
Subsystem blocks do not show their port names.	Set the subsystem parameter **Show port labels** to a value other than `none`.
Subsystem blocks show their port names.	Set the subsystem parameter **Show port labels** to `none`.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content in masked subsystems.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

MAAB guideline, Version 3.0: na_0005: Port block name visibility in Simulink models in the Simulink documentation.

Check orientation of Subsystem blocks

Check ID: `mathworks.maab.jc_0111`

Check the orientation of subsystem blocks.

Description

Subsystem inputs must be located on the left side of the block, and outputs must be located on the right side of the block.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Subsystem blocks are not using the right orientation	Rotate the subsystem so that inputs are on the left side of block and outputs are on the right side of the block.

Capabilities and Limitations

- JMAAB guideline, Version 4.0 limitation: The check does not flag the rotation of subsystems.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.

- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0111: Direction of Subsystem in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0111: Direction of Subsystem.

Check usage of Relational Operator blocks

Check ID: `mathworks.maab.jc_0131`

Check the position of Constant blocks used in Relational Operator blocks.

Description

When the relational operator is used to compare a signal to a constant value, the constant input should be the second, lower input.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Relational Operator blocks have a Constant block on the first, upper input.	Move the Constant block to the second, lower input.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0131: Use of Relational Operator block in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0131: Use of Relational Operator block.

Check usage of Switch blocks

Check ID: `mathworks.maab.jc_0141`

Check usage of Switch blocks.

Description

Verifies that the Switch block control input (the second input) is a Boolean value and that the block is configured to pass the first input when the control input is nonzero.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The Switch block control input (second input) is not a Boolean value.	Change the data type of the control input to Boolean.
The Switch block is not configured to pass the first input when the control input is nonzero.	Set the block parameter **Criteria for passing first input** to u2 ~=0.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0141: Use of the Switch block in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0141: Use of the Switch block.
- Switch block

Check usage of buses and Mux blocks

Check ID: `mathworks.maab.na_0010`

Check usage of buses and Mux blocks.

Description

This check verifies the usage of buses and Mux blocks.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The individual scalar input signals for a Mux block do not have common functionality, data types, dimensions, and units.	Modify the scalar input signals such that the specifications match.
The output of a Mux block is not a vector.	Change the output of the Mux block to a vector.
The input for a Bus Selector block is not a bus signal.	Make sure that the input for all Bus Selector blocks is a bus signal.

Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not flag non-scalar inputs as described in MAAB guideline na_0010: Grouping data flows into signals.

See Also

- MAAB guideline, Version 3.0: na_0010: Grouping data flows into signalsin the Simulink documentation.
- “Composite Signals” (Simulink)

Check for bitwise operations in Stateflow charts

Check ID: `mathworks.maab.na_0001`

Identify bitwise operators (`&`, `|`, and `^`) in Stateflow charts. If you select **Enable C-bit operations** for a chart, only bitwise operators in expressions containing Boolean data types are reported. Otherwise, all bitwise operators are reported for the chart.

Description

Do not use bitwise operators in Stateflow charts, unless you enable bitwise operations.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Stateflow charts with **Enable C-bit operations** selected use bitwise operators (`&`, `|`, and `^`) in expressions containing Boolean data types.	Do not use Boolean data types in the specified expressions.
The Model Advisor could not determine the data types in expressions with bitwise operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.
Stateflow charts with **Enable C-bit operations** cleared use bitwise operators (`&`, `|`, and `^`).	To fix this issue, do either of the following: • Modify the expressions to replace bitwise operators. • If not using Boolean data types, consider enabling bitwise operations. In the Chart properties dialog box, select **Enable C-bit operations**.

Capabilities and Limitations

- Applies only to charts that use C as the action language.
- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- “Binary and Bitwise Operations” (Stateflow) in the Stateflow documentation.
- MAAB guideline, Version 3.0: na_0001: Bitwise Stateflow operators in the Simulink documentation.

- JMAAB guideline, Version 4.0: na_0001: Bitwise Stateflow operators.
- “hisf_0003: Usage of bitwise operations” (Simulink) in the Simulink documentation.

Check for comparison operations in Stateflow charts

Check ID: `mathworks.maab.na_0013`

Identify comparison operations with different data types in Stateflow objects.

Description

Comparisons should be made between variables of the same data types.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Comparison operations with different data types were found.	Revisit the specified operations to avoid comparison operations with different data types.
The Model Advisor could not determine the data types in expressions with comparison operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: na_0013: Comparison operation in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 4.0: na_0013: Comparison operation in Stateflow.

Check for unary minus operations on unsigned integers in Stateflow charts

Check ID: `mathworks.maab.jc_0451`

Identify unary minus operations applied to unsigned integers in Stateflow objects.

Description

Do not perform unary minus operations on unsigned integers in Stateflow objects.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Unary minus operations are applied to unsigned integers in Stateflow objects.	Modify the specified objects to remove dependency on unary minus operations.
The Model Advisor could not determine the data types in expressions with unary minus operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: jc_0451: Use of unary minus on unsigned integers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc_0451: Use of unary minus on unsigned integers in Stateflow.

Check for equality operations between floating-point expressions in Stateflow charts

Check ID: `mathworks.maab.jc_0481`

Identify equal to operations (==) in expressions where at least one side of the expression is a floating-point variable or constant.

Description

Do not use equal to operations with floating-point data types. You can use equal to operations with integer data types.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Expressions use equal to operations (==) where at least one side of the expression is a floating-point variable or constant.	Modify the specified expressions to avoid equal to operations between floating-point expressions. If an equal to operation is required, a margin of error should be defined and used in the operation.
The Model Advisor could not determine the data types in expressions with equality operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

MAAB guideline, Version 3.0: jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow in the Simulink documentation.

Check input and output settings of MATLAB Functions

Check ID: `mathworks.maab.na_0034`

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Function identified in the results. If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink): • Change **Complexity** from `Inherited` to `On` or `Off`. • Change **Type** from `Inherit: Same as Simulink` to an explicit type. • Change **Size** from `−1 (Inherited)` to an explicit size.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: na_0034: MATLAB Function block input/output settings in the Simulink documentation.

- JMAAB guideline, Version 4.0: na_0034: MATLAB Function block input/output settings.

Check MATLAB Function metrics

Check ID: `mathworks.maab.himl_0003`

Display complexity and code metrics for MATLAB Functions. Report metric violations.

Description

This check provides complexity and code metrics for MATLAB Functions. The check additionally reports metric violations.

A results table provides links to MATLAB Functions that violate the complexity input parameters.

Available with Simulink Check.

Input Parameters

Maximum effective lines of code per function

Provide the maximum effective lines of code per function. Effective lines do not include empty lines, comment lines, or lines with a function `end` keyword.

Minimum density of comments

Provide minimum density of comments. Density is ratio of comment lines to total lines of code.

Maximum cyclomatic complexity per function

Provide maximum cyclomatic complexity per function. Cyclomatic complexity is the number of linearly independent paths through the source code.

Results and Recommended Actions

Condition	Recommended Action
MATLAB Function violates the complexity input parameters.	For the MATLAB Function: • If effective lines of code is too high, further divide the MATLAB Function. • If comment density is too low, add comment lines. • If cyclomatic complexity per function is too high, further divide the MATLAB Function.

Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: na_0016: Source lines of MATLAB Functions in the Simulink documentation.
- MAAB guideline, Version 3.0: na_0018: Number of nested if/else and case statement in the Simulink documentation.
- JMAAB guideline, Version 4.0: na_0016: Source lines of MATLAB Functions.
- JMAAB guideline, Version 4.0: na_0018: Number of nested if/else and case statement.

Check for mismatches between names of Stateflow ports and associated signals

Check ID: `mathworks.maab.db_0123`

Check for mismatches between Stateflow ports and associated signal names.

Description

The name of Stateflow input and output should be the same as the corresponding signal. The check does not flag:

- Name mismatches for reusable Stateflow charts in libraries.
- Stateflow ports if the corresponding signal does not have a label.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Signals have names that differ from the corresponding Stateflow ports.	Change the names of either the signals or the Stateflow ports.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts. Exclusions will not work for library linked charts.

See Also

- MAAB guideline, Version 3.0: db_0123: Stateflow port names in the Simulink documentation.
- JMAAB guideline, Version 4.0: db_0123: Stateflow port names.

Check scope of From and Goto blocks

Check ID: `mathworks.maab.na_0011`

Check the scope of From and Goto blocks.

Description

You can use global scope for controlling flow. However, From and Goto blocks must use local scope for signal flows.

Available with Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
From and Goto blocks are not configured with local scope.	• Make sure that the ports are connected. • Change the scope of the specified blocks to local.

Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

See Also

- MAAB guideline, Version 3.0: na_0011: Scope of Goto and From blocks in the Simulink documentation.

MISRA C:2012 Checks

In this section...

Check usage of Assignment blocks

Check ID: `mathworks.misra.AssignmentBlocks`

Identify Assignment blocks that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.

Description

This check applies to the Assignment block that is available in the Simulink block library under **Simulink** > **Math Operations**.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.	Set block parameter **Action if any output element is not assigned** to one of the recommended values: • **Error**, if Assignment block is not in an Iterator subsystem. • **Warning**, if Assignment block is in an Iterator subsystem.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

See Also

- MISRA C:2012, Rule 9.1
- ISO/IEC TS 17961: 2013, uninitref
- CERT C, EXP33-C
- CWE, CWE-908
- “hisl_0029: Usage of Assignment blocks” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

Check for blocks not recommended for MISRA C:2012

Check ID: `mathworks.misra.BlkSupport`

Identify blocks that are not supported or recommended for MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem.	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystemer. The deprecated Lookup Table blocks are Lookup and Lookup2D.	Consider replacing the deprecated Lookup Table blocks.
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- “hisl_0020: Blocks not recommended for MISRA C:2012 compliance” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “What Is a Model Advisor Exclusion?”

Check for unsupported block names

Check ID: `mathworks.misra.BlockNames`

Identify block names containing `/`.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Block names containing / were found in the model or subsystem.	Remove / from the block name.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

See Also

- MISRA C:2012, Rule 3.1
- “MISRA C Guidelines” (Embedded Coder).
- “MISRA C:2012 Compliance Considerations” (Simulink)

Check configuration parameters for MISRA C:2012

Check ID: `mathworks.misra.CodeGenSettings`

Identify configuration parameters that might impact MISRA C:2012 compliant code generation.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model Verification block enabling is set to `Use local settings` or `Enable All`.	In the Configuration Parameters, set **Model Verification block enabling** to `Disable All`.
System target file is set to a GRT-based target.	In the Configuration Parameters dialog box, on the **Code Generation** pane, set **System target file** to an ERT-based target.
Code Generation > **Interface** parameters are not set to the recommended values.	In the Configuration Parameters dialog box: • Set **Code replacement library** to `None` or `AUTOSAR 4.0` • Set **Shared code placement** to `Shared location` • Clear **Support: non-finite numbers** • Clear **Support: continuous time** (ERT-based target only) • Clear **Support non-inlined S-functions** (ERT-based target only) • Clear **MAT-file logging**
Parenthesis level is not set to `Maximum (Specify precedence with parentheses)`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Parentheses level** to `Maximum (Specify precedence with parentheses)`.

Condition	Recommended Action
Casting Modes is not set to `Standards Compliant`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set **Casting Modes** to `Standards Compliant`.
`GenerateSharedConstants` is set to `on`.	Use `get_param` to set `GenerateSharedConstants` to `off`.
System-generated identifiers is set to `Classic`.	In the Configuration Parameters dialog box, on the **Code Generation** > **Symbols** pane, set **System-generated identifiers** to `Shortened`.
Pack Boolean data into bitfields is selected and **Bitfield declarator type specifier** is set to `uchar_T`.	In the Configuration Parameters dialog box, on the **Optimization** > **Signals and Parameters** pane, if **Pack Boolean data into bitfields** is selected, set **Bitfield declarator type specifier** to `uint_T`.
Signed integer division rounds to is not set to `Zero` or `Floor`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, set **Signed integer division rounds to** to `Zero` or `Floor`.
Use division for fixed-point net slope computation is not set to `On` or `Use division for reciprocals of integers only`.	In the Configuration Parameters dialog box, on the **Optimization** pane, set **Use division for fixed-point net slope computation** to `On` or `Use division for reciprocals of integers only`.
Replace multiplications by powers of two with signed bitwise shifts is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, clear **Replace multiplications by powers of two with signed bitwise shifts**.
Allow right shifts on signed integers is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, Clear **Allow right shifts on signed integers**.

Condition	Recommended Action
Use dynamic memory allocation for model initialization is selected.	In the Configuration Parameters dialog box, clear **Use dynamic memory allocation for model initialization**.
Wrap on overflow is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Wrap on overflow** to `warning` or `error`.
Inf or NaN block output is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Inf or NaN block output** to `warning` or `error`.
Dynamic momory allocation in MATLAB Function blocks is selected.	In the Configuration Parameters dialog box, clear **Dynamic memory allocation in MATLAB Function blocks**.
`ERTFilePackagingFormat` is set to `Modular`.	Use `get_param` to set `ERTFilePackagingFormat` to `CompactWithDataFile` or `Compact`. If you click **Modify** to automatically fix the parameter setting, the value is set to `Compact`.
`PreserveStaticInFcnDecls` is set to `off`.	Use `get_param` to set `PreserveStaticInFcnDecls` to `on`. To set this value, `ERTFilePackagingFormat` must be set to `CompactWithDataFile` or `Compact`.

Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

Capabilities and Limitations

This check does not review referenced models.

See Also

- “hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)

Check for equality and inequality operations on floating-point values

Check ID: `mathworks.misra.CompareFloatEquality`

Identify equality and inequality operations on floating-point values.

Description

The check flags sources causing equality or inequality operations on floating-point values.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

The check does not flag blocks with equality or inequality operations on floating-point values if they are justified with a Polyspace® annotation. When you run the check, the **Blocks with justification** table lists blocks with equality or inequality operations that have a justification.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object has an equality or inequality operation on a floating-point value.	Consider using non-floating-point values for equality or inequality operations.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Dir 1.1
- CERT C, FLP00-C
- CWE, CWE-697
- “Annotate Blocks for Known Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

Check for bitwise operations on signed integers

Check ID: `mathworks.misra.CompliantCGIRConstructions`

Identify 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.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem has blocks that contain bitwise operations on signed integers.	Consider using unsigned integers for bitwise operations.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- "hisl_0060: Configuration parameters that improve MISRA C:2012 compliance" (Simulink)
- "MISRA C:2012 Compliance Considerations" (Simulink)
- "Secure Coding Standards" (Embedded Coder)

Check for recursive function calls

Check ID: `mathworks.misra.RecursionCompliance`

Identify recursive function calls in Stateflow charts.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags charts that have recursive function calls.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Chart has a recursive function call.	Remove recursive function call.

See Also

- MISRA C:2012, Dir 17.2
- "Guidelines for Avoiding Unwanted Recursion in a Chart" (Stateflow)

Check for switch case expressions without a default case

Check ID: `mathworks.misra.SwitchDefault`

Identify switch case expressions that do not have a default case.

Description

The check flags model objects that have switch case expressions without a default case.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

The check does not flag blocks without default cases if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks without default cases that have a justification.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object has a switch case expression without a default case.	For Switch Case blocks, consider selecting block parameter **Show default case** to explicitly specify a default case.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 16.4
- ISO/IEC TS 17961: 2013, swtchdflt
- CERT C, MSC01-C
- CWE, CWE-478
- “Annotate Blocks for Known Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

Check for blocks not recommended for C/C++ production code deployment

Check ID: `mathworks.codegen.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

Description

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder. If you are using blocks with support notes for code generation, review the information and follow the given advice.

Following the recommendations of this check increases the likelihood of generating code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- "Blocks and Products Supported for C Code Generation" (Simulink Coder)
- "What Is a Model Advisor Exclusion?"
- "Secure Coding Standards" (Embedded Coder)

Check for missing error ports for AUTOSAR receiver interfaces

Check ID: `mathworks.misra.AutosarReceiverInterface`

Identify AUTOSAR receiver interface inports that do not have matching error ports.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags AUTOSAR receiver interfaces inports that are missing error ports.

The check does not flag missing error ports if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists the missing error ports that have a justification.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
AUTOSAR receiver interface inport does not have a matching error port.	Add missing error port and map to the corresponding AUTOSAR receiver interface inport.

Capabilities and Limitations

You can:

- Analyzes top layer/root level models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C: 2012, Directive 4.7
- “MISRA C Guidelines” (Embedded Coder)
- “What Is a Model Advisor Exclusion?”
- “Annotate Blocks for Known Results” (Polyspace Bug Finder)

Check for missing const qualifiers in model functions

Check ID: `mathworks.misra.ModelFunctionInterface`

Identify missing const qualifiers in input data pointers.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags input data pointers that do not have a const qualifier.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
A const qualifier is not defined for the input data pointer.	Consider adding a const qualifier to the input data pointer.

See Also

- MISRA C:2012, Rule 8.13
- "MISRA C Guidelines" (Embedded Coder)

Check integer word length

Check ID: `mathworks.misra.IntegerWordLengths`

Identify integer word lengths that do not comply with hardware implementation settings

Description

The check flags integers whose word lengths exceed the number of bits permitted via the hardware implementation settings.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object contains integer word lengths that are not compliant with hardware implementation settings.	Update the integer so its length does not exceed the permitted number of bits. You can view the permitted number of bits in the Configuration Parameters dialog box, on the **Hardware Implementation > Device details** pane.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “MISRA C Guidelines” (Embedded Coder)
- “What Is a Model Advisor Exclusion?”
- “Secure Coding Standards” (Embedded Coder)

Secure Coding Checks for CERT C, CWE, and ISO/IEC TS 17961 Standards

These checks are used to validate that code generated by Embedded Coder complies with the CERT C, CWE, and ISO/IEC TS 17961 (Embedded Coder) secure coding standards.

Check configuration parameters for secure coding standards

Check ID: `mathworks.security.CodeGenSettings`

Identify configuration parameters that might impact compliance with secure coding standards.

Description

Following the recommendations of this check increases the likelihood of generating code that complies with CERT C, CWE, ISO/IEC TS 17961 secure coding standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model Verification block enabling is set to `Use local settings` or `Enable All`.	In the Configuration Parameters dialog box, set **Model Verification block enabling** to `Disable All`.
System target file is set to a GRT-based target.	In the Configuration Parameters dialog box, on the **Code Generation** > **General** pane, set **System target file** to an ERT-based target.
Code Generation > **Interface** parameters are not set to the recommended values.	In the Configuration Parameters dialog box, on the **Code Generation** > **Interface** pane: • Set **Code replacement library** to `None` or `AUTOSAR 4.0` • Clear **Support: non-finite numbers** • Clear **Support: continuous time** (ERT-based target only) In the Configuration Parameters dialog box: • Clear **Support non-inlined S-functions** (ERT-based target only) • Clear **MAT-file logging**
Signed integer division rounds to is not set to `Zero` or `Floor`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, set **Signed integer division rounds to** to `Zero` or `Floor`.
Replace multiplications by powers of two with signed bitwise shifts is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, clear **Replace multiplications by powers of two with signed bitwise shifts**.

Condition	Recommended Action
Allow right shifts on signed integers is selected.	In the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, clear **Allow right shifts on signed integers**.
Use dynamic memory allocation for model initialization is selected.	In the Configuration Parameters dialog box, clear **Use dynamic memory allocation for model initialization**.
Wrap on overflow is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Wrap on overflow** to `warning` or `error`.
Inf or NaN block output is set to `None`	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set **Inf or NaN block output** to `warning` or `error`.
Dynamic memory allocation in MATLAB Function blocks is selected.	In the Configuration Parameters dialog box, clear **Dynamic memory allocation in MATLAB Function blocks**.

Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

See Also

"Secure Coding Standards" (Embedded Coder)

Check for blocks not recommended for C/C++ production code deployment

Check ID: `mathworks.codegen.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

Description

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder. If you are using blocks with support notes for code generation, review the information and follow the given advice.

Following the recommendations of this check increases the likelihood of generating code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- “Blocks and Products Supported for C Code Generation” (Simulink Coder)
- “What Is a Model Advisor Exclusion?”
- “Secure Coding Standards” (Embedded Coder)

Check for blocks not recommended for secure coding standards

Check ID: `mathworks.security.BlockSupport`

Identify blocks not recommended for compliance with secure coding standards.

Description

Following the recommendations of this check increases the likelihood of generating code that complies with CERT C, CWE, ISO/IEC TS 17961 secure coding standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem.	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystem.	Consider replacing the deprecated Lookup Table blocks.
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- “What Is a Model Advisor Exclusion?”
- “Secure Coding Standards” (Embedded Coder)

Check usage of Assignment blocks

Check ID: `mathworks.misra.AssignmentBlocks`

Identify Assignment blocks that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.

Description

This check applies to the Assignment block that is available in the Simulink block library under **Simulink** > **Math Operations**.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.	Set block parameter **Action if any output element is not assigned** to one of the recommended values: • **Error**, if Assignment block is not in an Iterator subsystem. • **Warning**, if Assignment block is in an Iterator subsystem.

Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

See Also

- MISRA C:2012, Rule 9.1
- ISO/IEC TS 17961: 2013, uninitref
- CERT C, EXP33-C
- CWE, CWE-908
- “hisl_0029: Usage of Assignment blocks” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

Check for switch case expressions without a default case

Check ID: `mathworks.misra.SwitchDefault`

Identify switch case expressions that do not have a default case.

Description

The check flags model objects that have switch case expressions without a default case.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

The check does not flag blocks without default cases if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks without default cases that have a justification.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object has a switch case expression without a default case.	For Switch Case blocks, consider selecting block parameter **Show default case** to explicitly specify a default case.

Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 16.4
- ISO/IEC TS 17961: 2013, swtchdflt
- CERT C, MSC01-C
- CWE, CWE-478

- “Annotate Blocks for Known Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

Check for bitwise operations on signed integers

Check ID: `mathworks.misra.CompliantCGIRConstructions`

Identify 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.

Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
The model or subsystem has blocks that contain bitwise operations on signed integers.	Consider using unsigned integers for bitwise operations.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” (Simulink)
- “MISRA C:2012 Compliance Considerations” (Simulink)

- “Secure Coding Standards” (Embedded Coder)

Check for equality and inequality operations on floating-point values

Check ID: `mathworks.misra.CompareFloatEquality`

Identify equality and inequality operations on floating-point values.

Description

The check flags sources causing equality or inequality operations on floating-point values.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

The check does not flag blocks with equality or inequality operations on floating-point values if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks with equality or inequality operations that have a justification.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object has an equality or inequality operation on a floating-point value.	Consider using non-floating-point values for equality or inequality operations.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Dir 1.1
- CERT C, FLP00-C
- CWE, CWE-697

- “Annotate Blocks for Known Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

Check integer word length

Check ID: `mathworks.misra.IntegerWordLengths`

Identify integer word lengths that do not comply with hardware implementation settings

Description

The check flags integers whose word lengths exceed the number of bits permitted via the hardware implementation settings.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

Results and Recommended Actions

Condition	Recommended Action
Model object contains integer word lengths that are not compliant with hardware implementation settings.	Update the integer so its length does not exceed the permitted number of bits. You can view the permitted number of bits in the Configuration Parameters dialog box, on the **Hardware Implementation > Device details** pane.

Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C

- CWE, CWE-682
- “MISRA C Guidelines” (Embedded Coder)
- “What Is a Model Advisor Exclusion?”
- “Secure Coding Standards” (Embedded Coder)

Detect Dead Logic

Check ID: `mathworks.sldv.deadlogic`

Identify logic that stays inactive during simulation.

Description

This check identifies portions of your model that stay inactive during simulation.

You can run a more detailed analysis that identifies both dead logic and active logic using Simulink Design Verifier™ design error detection. For more information, see “Detect Dead Logic Caused by an Incorrect Value” (Simulink Design Verifier).

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards

Results and Recommended Actions

<table>
<tr><th>Result</th><th>Recommended Action</th></tr>
<tr><td>Failed, model incompatible</td><td>Resolve the model incompatibility. See:
“Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)
“Support Limitations for Model Blocks” (Simulink Design Verifier)
“Support Limitations for Simulink Software Features” (Simulink Design Verifier)
“Support Limitations for Stateflow Software Features” (Simulink Design Verifier)
“Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)
Also see
“Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).</td></tr>
</table>

Result	Recommended Action
Dead logic found in model	Simulink Design Verifier proved that these decision and condition outcomes cannot occur and are dead logic in the model. Dead logic can also be a side effect of specified constraints on parameters or specified minimum and maximum constraints on input ports. In rare cases, dead logic can result from approximations performed by Simulink Design Verifier. It is possible that there are objectives that this analysis did not decide. To extend the results of this analysis, use Simulink Design Verifier design error detection to also identify active logic. From the Simulink Editor, select **Analysis** > **Design Verifier** > **Options**. In the **Design Error Detection** pane, select both **Dead logic** and **Identify active logic**.
Dead logic not found in model	Simulink Design Verifier did not find dead logic in the model. It is possible that there are objectives that this analysis did not decide. To extend the results of this analysis, use Simulink Design Verifier design error detection to also identify active logic. From the Simulink Editor, select **Analysis** > **Design Verifier** > **Options**. In the **Design Error Detection** pane, select both **Dead logic** and **Identify active logic**.

See Also

- MISRA C:2012: Rule 2.1
- CERT C, MSC07-C
- CWE, CWE-561
- “Run Model Checks” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

- “Detect Dead Logic Caused by an Incorrect Value” (Simulink Design Verifier)
- “Design Verifier Pane: Design Error Detection” (Simulink Design Verifier)

Detect Integer Overflow

Check ID: `mathworks.sldv.integeroverflow`

Detects integer or fixed-point data overflow errors in your model

Description

This check identifies operations that exceed the data type range for integer or fixed-point operations.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See • “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier) • “Support Limitations for Model Blocks” (Simulink Design Verifier) • “Support Limitations for Simulink Software Features” (Simulink Design Verifier) • “Support Limitations for Stateflow Software Features” (Simulink Design Verifier) • “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier) Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).
Integer overflow found in model	To view the conditions that cause the integer overflow, create a harness model. When you simulate the harness, the inputs replicate the error. Click **View test case** in the Model Advisor report.

See Also

- MISRA C:2012: Directive 4.1
- ISO/IEC TS 17961: 2013, intoflow
- CERT C, INT30-C and INT32-C
- CWE, CWE-190
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)

Detect Division by Zero

Check ID: `mathworks.sldv.divbyzero`

Detects division-by-zero errors in your model

Description

This check identifies operations in your model that cause division-by-zero errors.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See • “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier) • “Support Limitations for Model Blocks” (Simulink Design Verifier) • “Support Limitations for Simulink Software Features” (Simulink Design Verifier) • “Support Limitations for Stateflow Software Features” (Simulink Design Verifier) • “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier) Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).

Result	Recommended Action
Division by zero found in model	To view the conditions that cause the division by zero, create a harness model. When you simulate the harness, the inputs replicate the error. Click **View test case** in the Model Advisor report.

See Also

- MISRA C:2012: Directive 4.1
- ISO/IEC TS 17961: 2013, diverr
- CERT C, INT33-C and FLP03-C
- CWE, CWE-369
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)

Detect Out Of Bound Array Access

Check ID: `mathworks.sldv.arraybounds`

Detects operations that access outside the bounds of an array index

Description

This check detects instances of out of bound array access in Simulink Design Verifier.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See • “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier) • “Support Limitations for Model Blocks” (Simulink Design Verifier) • “Support Limitations for Simulink Software Features” (Simulink Design Verifier) • “Support Limitations for Stateflow Software Features” (Simulink Design Verifier) • “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier) Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).
Out of bound array access found in model	To view the conditions that cause the out of bound array access, create a harness model. When you simulate the harness, the inputs replicate the error. Click **View test case** in the Model Advisor report.

See Also

- MISRA C:2012: Rule 18.1
- ISO/IEC TS 17961: 2013, invptr
- CERT C, ARR30-C
- CWE, CWE-118
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)

Detect Violation of Specified Minimum and Maximum Values

Check ID: `mathworks.sldv.minmax`

Detect signals which exceed specified minimum and maximum values

Description

This analysis 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.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See • “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier) • “Support Limitations for Model Blocks” (Simulink Design Verifier) • “Support Limitations for Simulink Software Features” (Simulink Design Verifier) • “Support Limitations for Stateflow Software Features” (Simulink Design Verifier) • “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier) Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).

Result	Recommended Action
Violation of minimum and/or maximum found in model	To view the conditions that cause the violation, create a harness model. When you simulate the harness, the inputs replicate the error. Click **View test case** in the Model Advisor report.

See Also

- MISRA C:2012: Directive 4.1
- CERT C, API00-C
- CWE, CWE-628
- “Secure Coding Standards” (Embedded Coder)
- “Design Range Checks” (Simulink Design Verifier)

Check Requirements Consistency in Model Advisor

In this section...

You can check requirements consistency using the Model Advisor.

Identify requirement links with missing documents

Check ID: `mathworks.req.Documents`

Verify that requirements link to existing documents.

Description

You used the Requirements Management Interface (RMI) to associate a design requirements document with a part of your model design and the interface cannot find the specified document.

Available with Simulink Requirements.

Results and Recommended Actions

Condition	Recommended Action
The requirements document associated with a part of your model design is not accessible at the specified location.	Open the Requirements dialog box and fix the path name of the requirements document or move the document to the specified location.

Capabilities and Limitations

You can exclude blocks and charts from this check.

Tips

If your model has links to a DOORS® requirements document, to run this check, the DOORS software must be open and you must be logged in.

See Also

"Maintenance of Requirements Links" (Simulink Requirements)

Identify requirement links that specify invalid locations within documents

Check ID: `mathworks.req.Identifiers`

Verify that requirements link to valid locations (e.g., bookmarks, line numbers, anchors) within documents.

Description

You used the Requirements Management Interface (RMI) to associate a location in a design requirements document (a bookmark, line number, or anchor) with a part of your model design and the interface cannot find the specified location in the specified document.

Available with Simulink Requirements.

Results and Recommended Actions

Condition	Recommended Action
The location in the requirements document associated with a part of your model design is not accessible.	Open the Requirements dialog box and fix the location reference within the requirements document.

Capabilities and Limitations

You can exclude blocks and charts from this check.

Tips

If your model has links to a DOORS requirements document, to run this check, the DOORS software must be open and you must be logged in.

If your model has links to a Microsoft® Word or Microsoft Excel® document, to run this check, those applications must be closed on your computer.

See Also

"Maintenance of Requirements Links" (Simulink Requirements)

Identify selection-based links having descriptions that do not match their requirements document text

Check ID: `mathworks.req.Labels`

Verify that descriptions of selection-based links use the same text found in their requirements documents.

Description

You used selection-based linking of the Requirements Management Interface (RMI) to label requirements in the model's **Requirements** menu with text that appears in the corresponding requirements document. This check helps you manage traceability by identifying requirement descriptions in the menu that are not synchronized with text in the documents.

Available with Simulink Requirements.

Results and Recommended Actions

Condition	Recommended Action
Selection-based links have descriptions that differ from their corresponding selections in the requirements documents.	If the difference reflects a change in the requirements document, click **Update** in the Model Advisor results to replace the current description in the selection-based link with the text from the requirements document (the external description). Alternatively, you can right-click the object in the model window, select **Edit/Add Links** from the **Requirements** menu, and use the Requirements dialog box that appears to synchronize the text.

Capabilities and Limitations

You can exclude blocks and charts from this check.

Tips

If your model has links to a DOORS requirements document, to run this check, the DOORS software must be open and you must be logged in.

If your model has links to a Microsoft Word or Microsoft Excel document, to run this check, those applications must be closed on your computer.

See Also

"Maintenance of Requirements Links" (Simulink Requirements)

Identify requirement links with path type inconsistent with preferences

Check ID: `mathworks.req.Paths`

Check that requirement paths are of the type selected in the preferences.

Description

You are using the Requirements Management Interface (RMI) and the paths specifying the location of your requirements documents differ from the file reference type set as your preference.

Available with Simulink Requirements.

Results and Recommended Actions

Condition	Recommended Action
The paths indicating the location of requirements documents use a file reference type that differs from the preference specified in the Requirements Settings dialog box, on the **Selection Linking** tab.	Change the preferred document file reference type or the specified paths by doing one of the following: • Click **Fix** to change the current path to the valid path. • In the model window, select **Analysis** > **Requirements** > **Settings**, select the **Selection Linking** tab, and change the value for the **Document file reference** option.

Linux Check for Absolute Paths

On Linux® systems, this check is named **Identify requirement links with absolute path type**. The check reports warnings for requirements links that use an absolute path.

The recommended action is:

1 Right-click the model object and select **Requirements** > **Edit/Add Links**.
2 Modify the path in the Document field to use a path relative to the current working folder or the model location.

Capabilities and Limitations

You can exclude blocks and charts from this check.

See Also

"Maintenance of Requirements Links" (Simulink Requirements)

Identify IBM Rational DOORS objects linked from Simulink that do not link to Simulink

Identify IBM® Rational® DOORS objects that are targets of Simulink-to-DOORS requirements traceability links, but that have no corresponding DOORS-to-Simulink requirements traceability links.

Description

You have Simulink-to-DOORS links that do not have a corresponding link from DOORS to Simulink. You must be logged in to the IBM Rational DOORS Client to run this check.

Available with Simulink Requirements.

Results and Recommended Actions

The Requirements Management Interface (RMI) examines Simulink-to-DOORS links to determine the presence of a corresponding return link. The RMI lists DOORS objects that do not have a return link to a Simulink object. For such objects, create corresponding DOORS-to-Simulink links:

1 Click the **FixAll** hyperlink in the RMI report to insert required links into the DOORS client for the list of missing requirements links. You can also create

individual links by navigating to each DOORS item and creating a link to the Simulink object.

2 Re-run the link check.

Model Metrics

Model Metrics

Model metrics analyze your model and help you assess your model with regard to size, architecture, readability, and compliance to standards. Simulink Check provides the metrics for these metric types:

- “Size Metrics” on page 2-309
- “Architecture Metrics” on page 2-310
- “Compliance Metrics” on page 2-311
- “Readability Metrics” on page 2-312

Using the Metrics Dashboard, you can collect and view model metrics to get an assessment of your project quality status. For more information, see “Collect and Explore Metric Data by Using the Metrics Dashboard”.

You can use the model metric API to run the model metrics programmatically and export the results to a file. For more information, see “Collect Model Metrics Programmatically”.

For your company guidelines and standards, you can also use the model metric API to create your own model metrics, compute those metrics, and export the metric data. For more information, see “Create a Custom Model Metric”.

Size Metrics

To collect metric data on a model or subsystem, run these metrics.

Metric	Description
“Simulink block metric” on page 2-312	Calculates the number of blocks in the model.
“Subsystem metric” on page 2-313	Calculates the number of subsystems in the model.
“Library link metric” on page 2-315	Calculates the number of library-linked blocks in the model.
“Effective lines of MATLAB code metric” on page 2-316	Calculates the number of effective lines of MATLAB code.

Metric	Description
"Stateflow chart objects metric" on page 2-317	Calculates the number of Stateflow objects.
"Lines of code for Stateflow blocks metric" on page 2-319	Calculates the number of code lines for the following Stateflow blocks in the model: • States • Transitions • Truth tables
"Subsystem depth metric" on page 2-320	Calculates the subsystem depth of the model.
"Input output metric" on page 2-321	Calculates the number of inports and outports in your model.
"Diagnostic warnings metric" on page 2-323	Calculates the number of diagnostic warnings reported.
"Explicit input output metric" on page 2-323	Calculates the number of inports and outports in your model.
"File metric" on page 2-325	Calculates the number of model and library files.
"Matlab Function metric" on page 2-326	Calculates the number of Matlab Function blocks in your model.
"Model file count" on page 2-327	Calculates the number of model files.
"Parameter metric" on page 2-328	Calculates the number of data objects that parameterize the behavior of a model.
"Stateflow chart metric" on page 2-329	Calculates the number Stateflow charts in your model.

For more information on model metrics, see "Collect Model Metrics".

Architecture Metrics

To learn more about the architecture for a model or subsystem, run these metrics.

Metric	Description
"Cyclomatic complexity metric" on page 2-330	Calculates the cyclomatic complexity of the model.
"Clone content metric" on page 2-331	Calculates the number of components involved in a clone, excluding libraries.
"Clone detection metric" on page 2-332	Calculates the number of clones in components across the model hierarchy.
"Library content metric" on page 2-333	Calculates the number of components involved in a library, excluding clones.

For more information on model metrics, see "Collect Model Metrics".

Compliance Metrics

To determine if your model or subsystem is compliant with standards and guidelines, run one or more of these metrics.

Metric	Description
"MATLAB code analyzer warnings" on page 2-337	Determines warnings for MATLAB code blocks in your model.
"Model Advisor Check Compliance for High-Integrity Systems" on page 2-338	Returns the fraction of checks the model passes from Model Advisor DO-178C/DO-331 Standards.
"Model Advisor Check Compliance for Modeling Standards for MAAB" on page 2-339	Returns the fraction of checks the model passes from Model Advisor MAAB Standard.
"Model Advisor Check Issues for High-Integrity Systems" on page 2-340	Reports the number of issues from Model Advisor DO-178C/DO-331 Standards.
"Model Advisor check issues for MAAB Standards" on page 2-341	Reports the number of issues from Model Advisor MAAB Standard.

For more information on model metrics, see "Collect Model Metrics".

Readability Metrics

Run these metrics to determine readability for a model or subsystem.

Metric	Description
"Nondescriptive block name metric" on page 2-334	Determines nondescriptive Inport, Outport, and Subsystem block names.
"Data and structure layer separation metric" on page 2-336	Calculates the data and structure layer separation.

For more information on model metrics, see "Collect Model Metrics".

Simulink block metric

Metric Type: Size

Metric ID: `mathworks.metrics.SimulinkBlockCount`

Model Advisor Check ID: `mathworks.metricchecks.SimulinkBlockCount`

Calculate the number of Simulink blocks in the model

Description

Use this metric to calculate the number of blocks in the model. The results provide the number of blocks at the model and subsystem level. This metric counts Simulink—based blocks, but does not include underlying blocks used to implement the block. This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Simulink block metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.SimulinkBlockCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of blocks.
- `AggregatedValue`: Number of blocks for component and its subcomponents.
- `Measures`: Not applicable.

Note The results from metric analysis of **Simulink block metric** can differ from calling `sldiagnostics`. The result of the Simulink block metric:

- Includes referenced models.
- Does not include any underlying blocks used to implement a MathWorks block that you used from the Simulink Library Browser.
- Does not include links into MathWorks libraries, which means that MathWorks library blocks that are masked subsystems are counted as one block. The inner content of those blocks is not counted.
- Does not include hidden content under Stateflow Charts or MATLAB Function blocks.
- Does not include requirements blocks.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Subsystem metric

Metric Type: Size

Metric ID: `mathworks.metrics.SubSystemCount`

Model Advisor Check ID: `mathworks.metricchecks.SubSystemCount`

Display number of subsystems in the model

Description

Use this metric to calculate the number of subsystems in the model. The results provide the number of subsystems at the model and subsystem level.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Subsystem metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.SubSystemCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of subsystems.
- `AggregatedValue`: Number of subsystems for a component and its subcomponent.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- Does not count subsystems linked to MathWorks libraries.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Library link metric

Metric Type: Size

Metric ID: `mathworks.metrics.LibraryLinkCount`

Model Advisor Check ID: `mathworks.metricchecks.LibraryLinkCount`

Display number of library links in the model

Description

Use this metric to calculate the number of library-linked blocks in the model. The results provide the number of library-linked blocks at the model and subsystem level.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Library link metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.LibraryLinkCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of library linked blocks.
- `AggregatedValue`: Number of library linked blocks for a component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- Does not count subsystems linked to MathWorks libraries.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Effective lines of MATLAB code metric

Metric Type: Size

Metric ID: `mathworks.metrics.MatlabLOCCount`

Model Advisor Check ID: `mathworks.metricchecks.MatlabLOCCount`

Display number of effective lines of MATLAB code

Description

Run this metric to calculate the number of effective lines of MATLAB code. Effective lines of MATLAB code are lines of executable code. Empty lines, lines that contain only comments, and lines that contain only an end statement are not considered effective lines of code. The results provide the number of effective lines of MATLAB code for each MATLAB Function block and for MATLAB functions in Stateflow charts.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Effective lines of MATLAB code metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.MatlabLOCCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of effective lines of MATLAB code.
- `AggregatedValue`: Number of effective lines of MATLAB code for a component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- Does not analyze the content of MATLAB code in external files.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Stateflow chart objects metric

Metric Type: Size

Metric ID: `mathworks.metrics.StateflowChartObjectCount`

Model Advisor Check ID:
`mathworks.metricchecks.StateflowChartObjectCount`

Display the number of Stateflow objects in each chart

Description

Run this metric to calculate the number of Stateflow objects. For each chart in the model, the results provide the number of the following Stateflow objects:

- Atomic subcharts

- Boxes
- Data objects
- Events
- Graphical functions
- Junctions
- Linked charts
- MATLAB functions
- Notes
- Simulink functions
- States
- Transitions
- Truth tables

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Stateflow chart objects metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.StateflowChartObjectCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of Stateflow objects.
- `AggregatedValue`: Number of Stateflow objects for a component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Lines of code for Stateflow blocks metric

Metric Type: Size

Metric ID: `mathworks.metrics.StateflowLOCCount`

Model Advisor Check ID: `mathworks.metricchecks.StateflowLOCCount`

Display the number of effective lines of code for Stateflow blocks

Description

Use this metric to calculate the number of effective lines of code in Stateflow. Effective lines of MATLAB code are lines of executable code. Empty lines, lines that contain only comments, and lines that contain only an end statement are not considered effective lines of code. This metric calculates the lines of code for the following Stateflow blocks in the model:

- Chart, counting the code on Transitions and inside States
- State Transition Table block
- Truth Table block

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Lines of code for Stateflow blocks metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.StateflowLOCCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`

- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of Stateflow block code lines.
- `AggregatedValue`: Number of Stateflow block code lines for a component and its subcomponents.
- `Measures`: Vector with two entries: number of effective lines of code in MATLAB action language and number of effective lines of code in C action language.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Subsystem depth metric

Metric Type: Size

Metric ID: `mathworks.metrics.SubSystemDepth`

Model Advisor Check ID: `mathworks.metricchecks.SubSystemDepth`

Calculates the maximum depth of all hierarchical children of a subsystem or model

Description

Use this metric to count the maximum depth of all hierarchical children for a given subsystem or model starting from the given component, or root of analysis. The depth is the relative depth of the deepest branch. Depth traversal analysis stops when it reaches a referenced model or a library. Depth and level are restarted with 0 for each of these components.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Subsystem depth metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.SubSystemDepth`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `None`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: subsystem depth for each component in the hierarchy.
- `AggregatedValue`: Not applicable.
- `Measure`: level of component in the hierarchy.
- `AggregatedMeasure`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Input output metric

Metric Type: Size

Metric ID: `mathworks.metrics.IOCount`

Display number of inputs and outputs in the model

Description

Use this metric to calculate the number of inputs and outputs in the model, which include:

- Inputs: Inport blocks, Trigger ports, Enable ports, chart input data and events.
- Outputs: Outport blocks, chart output data and events.
- Implicit inputs: From block, where the matching Goto block is outside of the component.
- Implicit outputs: Goto block, where the matching From block is outside of the component.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Max`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: total interface size or sum of the elements of `Measures`.
- `AggregatedValue`: Number of inputs and outputs for a component and its subcomponents.
- `Measures`: Array consisting of number of inputs, number of outputs, number of implicit inputs, and number of implicit outputs, which are local to the component.
- `AggregatedMeasures`: Maximum number of inputs, outputs, implicit inputs, and implicit outputs for a component and subcomponents.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Diagnostic warnings metric

Metric Type: Size

Metric ID: `mathworks.metrics.DiagnosticWarningsCount`

Calculate the number of diagnostic warnings reported during a model update for simulation.

Description

Use this metric to calculate the number of Simulink diagnostic warnings reported during a model update for simulation. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.DiagnosticWarningsCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of diagnostic warnings reported.
- `AggregatedValue`: Number of diagnostic warnings reported for component and its subcomponents.
- `Measure`: Not applicable.

Capabilities and Limitations

- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Explicit input output metric

Metric Type: Size

Metric ID: `mathworks.metrics.ExplicitIOCount`

Display number of inputs and outputs in the model, excluding From and Goto blocks.

Description

Use this metric to calculate the number of inputs and outputs in the model, which include:

- Inputs: Inport blocks, Trigger ports, Enable ports, chart input data and events.
- Outputs: Outport blocks, chart output data and events.

This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.ExplicitIOCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Max`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Total interface size or sum of the elements of `Measures`.
- `AggregatedValue`: Number of inputs and outputs for a component and its subcomponents.
- `Measures`: Array consisting of number of inputs and number of outputs which are local to the component.
- `AggregatedMeasures`: Maximum number of inputs and outputs for a component and subcomponents.

Capabilities and Limitations

The metric:

- Excludes From and Goto blocks.
- Runs on library models.

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

File metric

Metric Type: Size

Metric ID: `mathworks.metrics.FileCount`

Calculates the number of model and library files used by a specific component and its subcomponents.

Description

Use this metric to count the number of model and library files used by a specific component and its subcomponents. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.FileCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `None`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of model and library files.
- `AggregatedValue`: Not applicable.
- `Measures`: Not applicable.

Capabilities and Limitations

- Runs on library models.

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Matlab Function metric

Metric Type: Size

Metric ID: `mathworks.metrics.MatlabFunctionCount`

Calculates the number of Matlab Function blocks inside a component.

Description

Use this metric to count the number of Matlab Function blocks inside a component. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.MatlabFunctionCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of Matlab Function blocks.
- `AggregatedValue`: Number of Matlab Function blocks for component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

- Runs on library models.

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see "Collect Model Metrics".

Model file count

Metric Type: Size

Metric ID: `mathworks.metrics.ModelFileCount`

Calculate the number of model files.

Description

Use this metric to count the number of model files. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.ModelFileCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `None`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of files reference by a component and its subcomponents.
- `AggregatedValue`: Not applicable.
- `Measures`: Not applicable.

Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Parameter metric

Metric Type: Size

Metric ID: `mathworks.metrics.ParameterCount`

Calculate the number of parameters.

Description

Use this metric to calculate the amount of user-managed parameterization data inside a Simulink system. A parameter is a variable used by a Simulink block or object of a basic type (`single`, `double`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `boolean`, `logical`, `struct`, `char`, `cell`), `Simulink.Parameter`, `Simulink.Variant`, or `enum` value. The parameter can be stored in either the base workspace, the model workspace, or a data dictionary.

This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.ParameterCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of parameters used inside a component.
- `AggregatedValue`: Number of parameters for a component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

This metric:

- Uses the `Simulink.findVars` function and inherits the limitations of this function.
- Counts the parameter instances in a component rather than unique parameters.
- Does not include parameters in masked workspaces.
- Does not include data type and signal objects.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Stateflow chart metric

Metric Type: Size

Metric ID: `mathworks.metrics.StateflowChartCount`

Calculate the number of Stateflow charts at any component level.

Description

Use this metric to count the number of Stateflow charts at any component level. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.StateflowChartCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of Stateflow charts at the model level.
- `AggregatedValue`: Number of charts for component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Cyclomatic complexity metric

Metric Type: Architecture

Metric ID: `mathworks.metrics.CyclomaticComplexity`

Model Advisor Check ID: `mathworks.metricchecks.CyclomaticComplexity`

Display the local and aggregated cyclomatic complexity of the model

Description

Use this metric to calculate the cyclomatic complexity of the model. The results provide the local and aggregated cyclomatic complexity for the:

- Model
- Subsystems
- Charts
- States in charts
- MATLAB functions

Local complexity is the cyclomatic complexity for objects at their hierarchical level. Aggregated cyclomatic complexity is the cyclomatic complexity of an object and its descendants.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Cyclomatic complexity metric** in **By Task > Model Metrics > Complexity Metrics**.

- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.CyclomaticComplexity`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Local cyclomatic complexity.
- `AggregatedValue`: Aggregated cyclomatic complexity.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Does not run on library models.
- Analyzes content in masked subsystems.
- Does not analyze inactive variants.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

- "Collect Model Metrics"
- "Cyclomatic Complexity for Stateflow Charts" (Simulink Coverage)
- "Specify Coverage Options" (Simulink Coverage)

Clone content metric

Metric Type: Architecture

Check ID: `mathworks.metrics.CloneContent`

Calculate the number of components involved in a clone.

Description

Use this metric to calculate the number of components involved in an exact graphical subsystem clone, excluding libraries. Exact graphical subsystem clones must have identical block types, connections, and parameter values. If the software identifies a subsystem as a clone, the subsystem itself and all subcomponents are considered as being involved in a clone. Subcomponents linked to a library and its subcomponents are not considered to be involved in a clone. For more information on clone detection, see “Enable Component Reuse with Clone Detection”.

This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.CloneContent`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `None`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of components involved in a clone.
- `AggregatedValue`: Not applicable.
- `Measures`: Not applicable.

Capabilities and Limitations

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Clone detection metric

Metric Type: Architecture

Check ID: `mathworks.metrics.CloneDetection`

Calculate the number of clones in a model.

Description

Use this metric to count the number of exact graphical subsystem clones in a model. Exact graphical subsystem clones must have identical block types, connections, and parameter values. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.CloneDetection`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of clones.
- `AggregatedValue`: Number of clones for component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Library content metric

Metric Type: Architecture

Check ID: `mathworks.metrics.LibraryContent`

Calculate the number of components involved in a library, excluding clones.

Description

Use this metric to calculate the number of components involved in a library, excluding clones. This metric is available with Simulink Check. To collect data for this metric, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.LibraryContent`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `None`
- `slmetric.metric.AggregateComponentDetails`: `false`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of components involved in a library, excluding clones.
- `AggregatedValue`: Not applicable.
- `Measures`: Not applicable.

Capabilities and Limitations

- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”.

Nondescriptive block name metric

Metric Type: Readability

Check ID: `mathworks.metrics.DescriptiveBlockNames`

Model Advisor Check ID: `mathworks.metricchecks.DescriptiveBlockNames`

Display nondescriptive Inport, Outport, and Subsystem block names

Description

Run this metric to determine nondescriptive Inport, Outport, and Subsystem block names. Default names appended with an integer are nondescriptive block names. The results provide the nondescriptive block names at the model and subsystem levels.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Nondescriptive block name metric** in **By Task > Model Metrics > Readability Metrics**.
- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.DescriptiveBlockNames`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of nondescriptive Inport, Outport, and Subsystem block names.
- `AggregatedValue`: Number of nondescriptive Inport, Outport, and Subsystem block names for a component and its subcomponents.
- `Measures`: 1-D vector containing:
  - Total number of Inport blocks
  - Number of Inport blocks with nondescriptive names
  - Total number of Outport blocks
  - Number of Outport blocks with nondescriptive names
  - Total number of Subsystem blocks
  - Number of Subsystem blocks with nondescriptive names
- `AggregatedMeasures`: 1-D vector containing sum of:
  - Total number of Inport blocks
  - Number of Inport blocks with nondescriptive names

- Total number of Outport blocks
- Number of Outport blocks with nondescriptive names
- Total number of Subsystem blocks
- Number of Subsystem blocks with nondescriptive names

Capabilities and Limitations

The metric:

- Does not run on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”

Data and structure layer separation metric

Metric Type: Readability

Metric ID: `mathworks.metrics.LayerSeparation`

Model Advisor Check ID: `mathworks.metricchecks.LayerSeparation`

Display data and structure layer separation

Description

Run this metric to calculate the data and structure layer separation. The results provide the separation at the model and subsystem level.

Run this metric to calculate the data and structure layer separation. The results provide the separation at the model and subsystem levels.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Data and structure layer separation metric** in **By Task > Model Metrics > Readability Metrics**.

- Programmatically, use `slmetric.Engine.getMetrics` with the metric identifier, `mathworks.metrics.LayerSeparation`.

For guidelines about blocks on model levels, see the MAAB 3.0 guideline db_0143: Similar block types on the model levels.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of basic blocks on a structural level.
- `AggregatedValue`: Number of basic blocks on a structural level for a component and its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Does not run on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

See Also

For more information on model metrics, see “Collect Model Metrics”

MATLAB code analyzer warnings

Metric Type: Compliance

Metric ID: `mathworks.metrics.MatlabCodeAnalyzerWarnings`

Use this metric to calculate the number of MATLAB code analyzer warnings from MATLAB code in the model

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of MATLAB code analyzer warnings
- `AggregatedValue`: Number of MATLAB code analyzer warnings aggregated for a component and subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Analyzes MATLAB code in MATLAB Function blocks
- Analyzes MATLAB functions in Stateflow charts
- Runs on library models
- Analyzes content in masked subsystems
- If specified, analyzes content of library-linked blocks and referenced models
- Does not analyze external MATLAB code files

See Also

- "Collect Model Metrics"
- "Check Code for Errors and Warnings" (MATLAB)

Model Advisor Check Compliance for High-Integrity Systems

Metric Type: Compliance

Metric ID: `mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178`

Use this metric to calculate the fraction of Model Advisor checks that pass for the **High-Integrity Systems** subgroups.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Percentile`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Fraction of total number of checks passed in **High-Integrity Systems** subgroups.
- `AggregatedValue`: Fraction of total number of checks passed in **High-Integrity Systems** subgroups aggregated for a component and all of its subcomponents.
- `Measures`: Vector containing: number of checks passed in subgroups and number of checks in subgroups.
- `AggregatedMeasures`: Vector containing: number of checks passed in subgroups and number of checks in subgroup, for a component and all its subcomponents.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

See Also

- "Collect Model Metrics"
- "Model Checks for DO-178C/DO-331 Standard Compliance"

Model Advisor Check Compliance for Modeling Standards for MAAB

Metric Type: Compliance

Metric ID: `mathworks.metrics.ModelAdvisorCheckCompliance.maab`

Use this metric to calculate the fraction of Model Advisor checks that pass for the group **Modeling Standards for MAAB**

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Percentile`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Fraction of total number of checks passed in MAAB.
- `AggregatedValue`: Fraction of total number of checks passed in MAAB aggregated for a component and all of its subcomponents.
- `Measures`: Vector containing: number of checks passed in group and number of checks in group.
- `AggregatedMeasures`: Vector containing: number of checks passed in group and number of checks in group, for a component and all its subcomponents.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

See Also

- "Collect Model Metrics"
- "Model Checks for MathWorks Automotive Advisory Board (MAAB) Guideline Compliance"

Model Advisor Check Issues for High-Integrity Systems

Metric Type: Compliance

Metric ID: `mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178`

Use this metric to calculate number of issues reported by the subgroups of Model Advisor checks for **High-Integrity Systems**. An issue is a Simulink object that the Model Advisor check flags. You see an issue in the check output as a hyperlink and in the Simulink Editor with Model Advisor highlighting. For configuration parameter checks, we add one issue to each model component that fails the check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of issues reported by the **High-Integrity Systems** checks
- `AggregatedValue`: Number of issues reported by the **High-Integrity Systems** checks aggregated for a component and all of its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

See Also

- “Collect Model Metrics”
- “Model Checks for DO-178C/DO-331 Standard Compliance”

Model Advisor check issues for MAAB Standards

Metric Type: Compliance

Metric ID: `mathworks.metrics.ModelAdvisorCheckIssues.maab`

Use this metric to calculate number of issues reported by the group of Model Advisor checks for **Modeling Standards for MAAB**. An issue is a Simulink object that is flagged by the Model Advisor check. You see an issue in the check output as a hyperlink and in the Simulink Editor with Model Advisor highlighting.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- `Value`: Number of issues reported by the Model Advisor for MAAB checks.
- `AggregatedValue`: Number of issues reported by the Model Advisor for MAAB checks aggregated for a component and all of its subcomponents.
- `Measures`: Not applicable.

Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.
- Adds check issues on the configuration set or issues with data objects to the issue count at the model root level.

See Also

- "Collect Model Metrics"
- "Model Checks for MathWorks Automotive Advisory Board (MAAB) Guideline Compliance"

See Also

Related Examples

- "Collect Model Metrics Using the Model Advisor"
- "Collect Model Metrics Programmatically"
- "Model Metric Data Aggregation"
- "Create a Custom Model Metric"

3

Model Transformer Tasks

Model Transformer Tasks

In this section...

You can use the Model Transformer tool to refactor a model to implement variants. You can perform the steps in the Model Transformer all at once or one step at a time.

Transform the model to variant system

This folder contains the steps to transform a model to a variant system. The following transformations are possible:

- If an If block connects to one or more If Action Subsystems and each If Action Subsystem has one outport, replace this modeling pattern with a subsystem and a Variant Source block.
- If an If block connects to an If Action Subsystem that has no outport or two or more outports, replace this modeling pattern with a Variant Subsystem block.
- If a Switch Case block connects to one or more Switch Case Action Subsystems and each Switch Case Action Subsystem has one outport, replace this modeling pattern with a subsystem and a Variant Source block.
- If a Switch Case block connects to a Switch Case Action Subsystem that has no outport or two or more outports, replace this modeling pattern with a Variant Subsystem block.
- Replace a Switch block with a Variant Source block.
- Replace a Multiport Switch block that has two or more data ports with a Variant Source block.

Note For some model patterns and settings, the Model Transformer cannot perform every one of the preceding transformations.

If you click **Run all**, the Model Transformer performs the three steps in the transformation. The result is a model that contains variant blocks. This model is in the folder that has the prefix `m2m` plus the original model name.

If you want to run every step in the transformation at once, rather than running the steps individually, you can still specify input parameters for those steps that have them.

See Also

- "Transform Model to Variant System"

1. Identify system constants for use in variant transformation

A system constant is the control input or is part of an arithmetic expression that forms the control input to Multiport Switch or Switch blocks and the inputs to If or Switch Case blocks. The control input must be Constant blocks and some combination of blocks that form a supported MATLAB expression. In the Constant block parameters dialog box, the **Constant value** parameters are the system constants. In the transformed model, system constants are part of condition expressions in Variant Source or Variant Subsystem blocks.

When you click **Run This Task**, this step lists system constants that qualify to be part of condition expressions in Variant Source or Variant Subsystem blocks. For a system constant to qualify, it must be a scalar and a `Simulink.Parameter` object with one of these storage classes:

- `Define` with header file specified
- `ImportedDefine` with header file specified
- `CompilerFlag`
- `SystemConstant(AUTOSAR)`
- User-defined custom storage class that defines data as a macro in specified header file

After you run this check, in the results section, you can choose not to use a system constant in the variant transformation by clearing the check box next to it.

See Also

- "Transform Model to Variant System"

2. Identify blocks that qualify for variant transformation

When you click **Run This Task**, in the results section, this step lists modeling patterns that qualify for transformation into Variant Source and Variant Subsystem blocks. Each modeling pattern is a hyperlink to the corresponding location in the model. If you do not want the Model Transformer to perform a transformation, clear the check box next to the qualifying pattern.

See Also

- "Transform Model to Variant System"

3. Convert blocks to variants

When you click **Run This Task**, the Model Transformer creates a model with the blocks that you specified for variant transformation in the preceding step. The transformed model is in the folder that has the prefix `m2m` plus the original model name.

See Also

- "Transform Model to Variant System"

4

Clone Detection Tasks

Clone Detection Checks

Use the Identify Modeling Clones tool to refactor a model by identifying clones and creating models that replace clones with links to subsystem blocks in a library.

In this section...

Identify Exact Clones

This folder contains these checks:

- **Identify library clones and replace them with links to library blocks**
- **Identify graphical clones and replace them with links to library blocks**
- **Identify functional clones and replace them with links to library blocks**

If you click **Run Selected Checks**, the tool exectues these checks and creates models with links to library blocks. The tool identifies clones across referenced model boundaries. The tool identifies library and graphical clones in all model regions including commented-out areas and inactive variants. If you do not want to perform a check, clear the check box next to that check.

Exact clones have identical block types, connections, and parameter values. Exact graphical clones have identical parameter settings and values. Exact functional clones have identical parameter values, but they can have different parameter settings. For example, two Gain blocks can have different `Simulink.Parameters` for the value parameter as long as those parameters evaluate to the same numeric value. Exact clones can have these differences:

- Two clones can have a different sorted order.
- The length of signal lines and the location and size of blocks can be different as long as the block connections are the same.
- Blocks and signals can have different names.

See Also

- “Enable Component Reuse with Clone Detection”

Identify library clones and replace them with links to library blocks

When you click **Run This Check**, the tool lists modeling patterns that are graphical clones of library subsystems. In the **Library file name** field, you specify a library in which to check a model for clones.

In the modeling patterns list, each clone is a hyperlink to the corresponding location in the model. If you do not want to replace a modeling pattern with a link to a library block, you can clear the check box next to the clone.

When you click **Refactor Model**, the tool creates a model with links to the library blocks. By default, the model name is the prefix `gen1_` plus the original model name. In the input parameters, you can specify another prefix.

See Also

- “Enable Component Reuse with Clone Detection”

Identify graphical clones and replace them with library blocks

When you click **Run This Check**, the tool lists subsystems that are graphical clones. In the list, each subsystem clone is a hyperlink to the corresponding location in the model. If you do not want to replace a subsystem with a link to a library block, you can clear the check box next to the subsystem.

When you click **Refactor Model**, the tool creates a library of subsystem clones and a model with links to these library blocks. By default, the library file name is `graphicalCloneLibFile`. The model name is the prefix `gen2_` plus the original model name. In the input parameters, you can specify another prefix.

See Also

- "Enable Component Reuse with Clone Detection"

Identify functional clones and replace them with links to library blocks

When you click **Run This Check**, the tool lists subsystems that are functional clones. In the list, each subsystem clone is a hyperlink to the corresponding location in the model. If you do not want to replace a subsystem with a link to a library block, you can clear the check box next to the subsystem.

When you click **Refactor Model**, the tool creates a library of subsystem clones and a model with links to these library blocks. By default, the library file name is `functionalCloneLibFile`. The model name is the prefix `gen3_` plus the original model name. In the input parameters, you can specify another prefix.

See Also

- "Enable Component Reuse with Clone Detection"

Identify Similar Clones

This folder contains these checks:

- **Identify similar library clones**
- **Identify similar graphical clones**
- **Identify similar functional clones**

If you click **Run Selected Checks**, the tool executes these three checks. The tool identifies clones across referenced model boundaries. The tool identifies graphical clones in all model regions including commented-out regions and inactive variants. If you do not want to perform a check, clear the check box next to that check.

Similar clones have identical block types and connections, but they can have different parameter settings and values. The check **Identify similar graphical clones** lists similar clones across a model hierarchy including inactive variants and commented-out regions. The check **Identify similar functional clones** lists the same clones as the **Identify similar graphical clones** check excluding those clones in inactive variants and commented-out regions. Similar clones can have these differences:

- Two clones can have a different sorted order.
- The signal line length and location and size of blocks can be different as long as the block connections are the same.
- Blocks and signals can have different names.

See Also

- "Enable Component Reuse with Clone Detection"

Identify similar library clones

When you click **Run This Check**, the tool lists modeling patterns that are similar to library subsystems. The tool checks for similar library clones across a model hierarchy including in inactive variants and commented-out regions. In the **Library file name** field, you specify a library in which to check a model for clones.

In the modeling patterns list, each subsystem clone is a hyperlink to the corresponding location in the model.

See Also

- "Enable Component Reuse with Clone Detection"

Identify similar graphical clones

When you click **Run This Check**, the tool lists subsystems that are similar graphical clones. The tool checks for similar graphical clones across a model hierarchy including in inactive variants and commented-out regions.

In the subsystems list, each subsystem clone is a hyperlink to the corresponding location in the model.

See Also

- "Enable Component Reuse with Clone Detection"

Identify similar functional clones

When you click **Run This Check**, the tool lists subsystems that are functional clones. This check lists the same clones as the check **Identify similar graphical clones** lists excluding those clones in inactive variants and commented-out regions. In the list, each subsystem clone is a hyperlink to the corresponding location in the model.

See Also

- “Enable Component Reuse with Clone Detection”