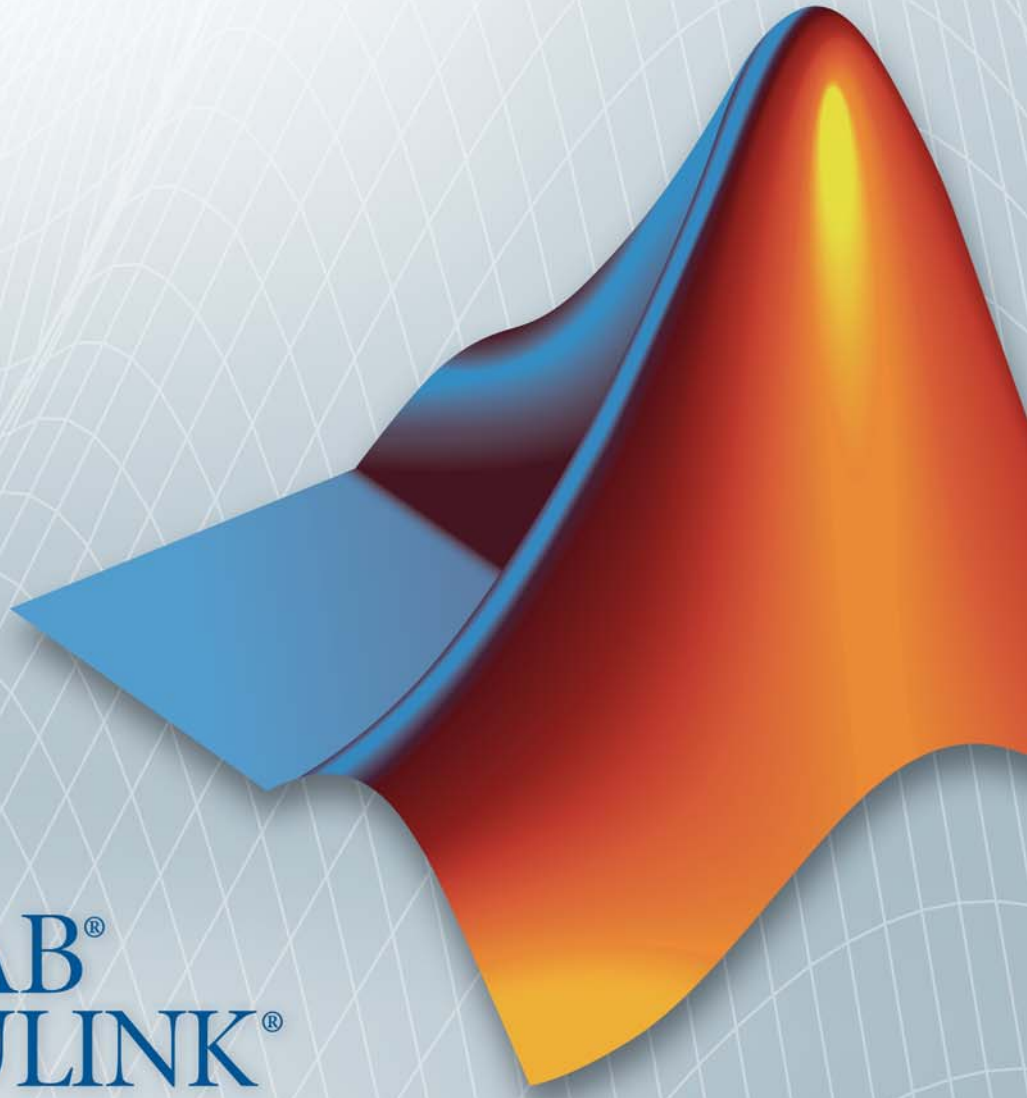
Simulink® Code Inspector™

Reference

R2012b



MATLAB®
&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Simulink® Code Inspector™ Reference

Trademarks

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

Patents

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

Revision History

September 2011	Online only	New for Version 1.0 (Release 2011b)
March 2012	Online only	Revised for Version 1.1 (Release 2012a)
September 2012	Online only	Revised for Version 1.2 (Release 2012b)

Contents

1 Function Reference

2 Class Reference

3 Functions — Alphabetical List

4 Model Configuration Constraints

5 Block Constraints

6 Model Advisor Checks

7 Simulink Code Inspector Dialog Box Parameters

1

Function Reference

Code Inspection

getCodeFolder (slci.Configuration)	Return code folder for code inspection
getCodePlacement (slci.Configuration)	Return code placement for code inspection
getFollowModelLinks (slci.Configuration)	Return model reference handling for model compatibility checking or code inspection
getGenerateCode (slci.Configuration)	Return code generation option for code inspection
getReportFolder (slci.Configuration)	Return report folder for code inspection
getTerminateOnIncompatibility (slci.Configuration)	Return termination option for code inspection
getTopModel (slci.Configuration)	Return top-model attribute for code inspection
inspect (slci.Configuration)	Inspect code generated from model
setCodeFolder (slci.Configuration)	Specify code folder for code inspection
setCodePlacement (slci.Configuration)	Specify code placement for code inspection
setFollowModelLinks (slci.Configuration)	Specify model reference handling for model compatibility checking or code inspection
setGenerateCode (slci.Configuration)	Specify whether to generate code before code inspection
setReportFolder (slci.Configuration)	Specify report folder for code inspection
setTerminateOnIncompatibility (slci.Configuration)	Specify whether to terminate code inspection if model is incompatible
setTopModel (slci.Configuration)	Specify whether model being configured for code inspection is top model

slci.Configuration	Create code inspection object
slci.ExportTraceReport	Generate XLS file that contains traceability matrix

Model Compatibility Checking

checkCompatibility (slci.Configuration)	Check model compatibility with code inspection
getFollowModelLinks (slci.Configuration)	Return model reference handling for model compatibility checking or code inspection
setFollowModelLinks (slci.Configuration)	Specify model reference handling for model compatibility checking or code inspection
slci.Configuration	Create code inspection object
slciadvisor	Open Simulink® Code Inspector™ Advisor

2

Class Reference

Code Inspection

slci.Configuration	Control code inspection and compatibility checking for model

3

Functions — Alphabetical List

slci.Configuration.checkCompatibility

Purpose

Check model compatibility with code inspection

Syntax

```
[results] = checkCompatibility(cfgObj)
[results] = checkCompatibility(cfgObj, Name, Value)
```

Description

`[results] = checkCompatibility(cfgObj)` checks a model for compatibility with the code inspection process and returns objects containing results information.

`[results] = checkCompatibility(cfgObj, Name, Value)` additionally applies the settings specified in name-value pair arguments.

This method runs the Simulink Code Inspector compatibility checker to determine if a model complies with the constrained set of modeling semantics and code optimizations supported by the code inspection process.

You can use the methods `slci.Configuration.getFollowModelLinks` and `slci.Configuration.setFollowModelLinks` to configure whether the scope of the compatibility check encompasses referenced models.

Tips

Before running the Code Inspector on a model, run compatibility checks repeatedly until the model is compatible.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `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

Specify whether to display results of the compatibility checks.

Value	Description
`'Summary'` (default)	Displays a summary of the model results in the Command Window.
`'Details'`	Displays the following in the Command Window: • Which system is being checked while the run is in progress • For each system, the pass and fail results of each check. • A summary of the system results.
`'None'`	Displays no information in the Command Window.

Default: `` `Summary' ``

Output Arguments

results	Cell array of `ModelAdvisor.SystemResult` objects, one for each model checked. Each `ModelAdvisor.SystemResult` object contains an array of `CheckResultObj` objects.
CheckResultObj	Array of `ModelAdvisor.CheckResult` objects, one for each check that runs.

Examples

This example shows how to programmatically run the compatibility checker and report results.

```
fprintf('\nInvoking compatibility checker ...\n');

config = slci.Configuration('slcidemo_roll');
result = config.checkCompatibility('DisplayResults','None');

for i = 1:length(result)
    fprintf('\nModel ''%s'' passed %d checks with %d issues.',...
        result{i}.system,...
        result{i}.numPass, result{i}.numWarn + result{i}.numFail)
end
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run model compatibility checks.

See Also

`slci.Configuration.getFollowModelLinks` | `slci.Configuration.setFollowModelLinks`

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"
- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Return code folder for code inspection

Syntax

```
folder = getCodeFolder(cfgObj)
```

Description

`folder = getCodeFolder(cfgObj)` returns the path to a code folder, as previously specified using `slci.Configuration.setCodeFolder`. Use this method only if you are inspecting previously generated code that has been repackaged to reside in a single, user-defined folder, as specified using `slci.Configuration.setCodePlacement`.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

folder	String specifying a folder path or, if you have not previously set a code folder value, `''` (default).

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setCodePlacement('Single folder')
>> config.setCodeFolder(fullfile('C:','packngo','model1'))
>> pkg = config.getCodePlacement()
pkg =
Single folder
>> folder = config.getCodeFolder()
folder =
C:\packngo\model1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setCodeFolder` | `slci.Configuration.setCodePlacement`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Return code placement for code inspection

Syntax

```
value = getCodePlacement(cfgObj)
```

Description

`value = getCodePlacement(cfgObj)` returns the value of a code inspection option that specifies whether generated code has been repackaged to reside in a single, user-defined folder. The value is meaningful only if you are inspecting previously generated code.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

value	String specifying one of the following values: • `Single folder` if the generated code has been repackaged to reside in a single, user-defined folder. • `Embedded Coder default` (default) if the generated code resides in the default folders created by code generation.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setCodePlacement('Single folder')
>> config.setCodeFolder(fullfile('C:','packngo','model1'))
>> pkg = config.getCodePlacement()
pkg =
Single folder
>> folder = config.getCodeFolder()
folder =
C:\packngo\model1
>>
```

slci.Configuration.getCodePlacement

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setCodePlacement` | `slci.Configuration.setCodeFolder`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Return model reference handling for model compatibility checking or code inspection

Syntax

```
value = getFollowModelLinks(cfgObj)
```

Description

`value = getFollowModelLinks(cfgObj)` returns the value of a code inspection option that specifies whether model compatibility checking and code inspection should be performed for every descendant of this model in the model reference hierarchy.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

value	True if model compatibility checking and code inspection should be performed for every descendant of this model in the model reference hierarchy; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setFollowModelLinks(true)
>> value = config.getFollowModelLinks()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run model compatibility checking and code inspection.

See Also

`slci.Configuration.setFollowModelLinks`

How To

- “Check Model Compatibility Using the Graphical User Interface”
- “Check Model Compatibility Using the Command-Line Interface”
- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Purpose

Return code generation option for code inspection

Syntax

```
value = getGenerateCode(cfgObj)
```

Description

`value = getGenerateCode(cfgObj)` returns the value of a code inspection option that specifies whether to generate model code as part of code inspection.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

value	True if model code should be generated at the beginning of code inspection; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setGenerateCode(true)
>> value = config.getGenerateCode()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setGenerateCode`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

slci.Configuration.getReportFolder

Purpose

Return report folder for code inspection

Syntax

```
folder = getReportFolder(cfgObj)
```

Description

`folder = getReportFolder(cfgObj)` returns the path to a folder in which code inspection places code inspection report artifacts.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

folder	String specifying a folder path. If you have not previously set a report folder value, the default is `slprj/slci`, relative to the location of the model.

Examples

```
>> pwd
ans =
C:\work
>> config = slci.Configuration('mymodel');
>> folder = config.getReportFolder()
folder =
C:\work\slprj\slci
>> config.setReportFolder(fullfile('C:','work','mymodel_report'));
>> folder = config.getReportFolder()
folder =
C:\work\mymodel_report
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setReportFolder`

How To

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

slci.Configuration.getTerminateOnIncompatibility

Purpose

Return termination option for code inspection

Syntax

```
value = getTerminateOnIncompatibility(cfgObj)
```

Description

`value = getTerminateOnIncompatibility(cfgObj)` returns the value of a code inspection option that specifies whether code inspection terminates if a model fails compatibility checking. If termination is selected, model code generation (if requested) also does not occur.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.

Output Arguments

value	True if code inspection should terminate if a model fails code inspection; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setTerminateOnIncompatibility(true)
>> value = config.getTerminateOnIncompatibility()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setTerminateOnIncompatibility` | `slci.Configuration.checkCompatibility`

How To

- "Check Model Compatibility Using the Graphical User Interface"

- “Check Model Compatibility Using the Command-Line Interface”
- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

slci.Configuration.getTopModel

Purpose

Return top-model attribute for code inspection

Syntax

```
value = getTopModel(cfgObj)
```

Description

`value = getTopModel(cfgObj)` returns the value of a code inspection attribute that specifies whether the model being configured for code inspection is the top model in the model reference hierarchy. If the model is not the top model, code inspection (and code generation if requested) uses a model reference target rather than a top model target..

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = slci.Configuration(*modelName*);.

Output Arguments

value	True if the model being configured for code inspection is the top model in the model reference hierarchy; false otherwise. The default is true.

Examples

The following example configures code inspection to use a model reference target.

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setTopModel(false)
>> value = config.getTopModel()
value =
     0
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setTopModel`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

slci.Configuration.inspect

Purpose

Inspect code generated from model

Syntax

```
results = inspect(cfgObj)
results = inspect(cfgObj, Name, Value)
```

Description

`results = inspect(cfgObj)` executes the code inspection process per code inspection configuration parameters and creates and displays a code inspection report.

`results = inspect(cfgObj, Name, Value)` additionally applies the settings specified in name-value pair arguments.

Tips

Before inspecting code generated from a model, run `slci.Configuration.checkCompatibility` repeatedly, modifying the model, until the model is compatible with code inspection.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `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

Specify whether to display inspection results.

Value	Description
`'Summary'` (default)	Displays a summary of the model results in the Command Window.
`'Details'`	Displays the following in the Command Window: • Which system is being inspected while the run is in progress • For each system, the pass and fail results of each inspection. • A summary of the system results.
`'None'`	Displays no information in the Command Window.

Default: `` `Summary' ``

Output Arguments

`results`

Structure containing the following fields:

- `ModelName`: String specifying the name of the model for which code was inspected.
- `Status`: String specifying the status returned by code inspection.
- `ReportFile`: String specifying the folder containing the code inspection report.

Examples

This example shows how to programmatically run the Code Inspector and report results. The model is assumed to have previously passed compatibility checks (see `slci.Configuration.checkCompatibility`).

```
config = slci.Configuration('slcidemo_roll');
config.setReportFolder(fullfile('.','report'));
result = config.inspect();
fprintf('Model %s status: %s\n',result.ModelName, result.Status);
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.checkCompatibility`

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"
- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Specify code folder for code inspection

Syntax

```
setCodeFolder(cfgObj, folder)
```

Description

`setCodeFolder(cfgObj, folder)` specifies the path to a folder containing previously generated code to be inspected. Use this method only if you are inspecting generated code that has been repackaged to reside in a single, user-defined folder, as specified using `slci.Configuration.setCodePlacement`.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.
folder	String specifying a folder path.

Examples

In the following example, you call `slci.Configuration.setCodePlacement` to specify that generated code has been repackaged to reside in a single folder, and then call `slci.Configuration.setCodeFolder` to specify the folder path.

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setCodePlacement('Single folder')
>> config.setCodeFolder(fullfile('C:','packngo','model1'))
>> pkg = config.getCodePlacement()
pkg =
Single folder
>> folder = config.getCodeFolder()
folder =
C:\packngo\model1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setCodePlacement` | `slci.Configuration.getCodeFolder`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Specify code placement for code inspection

Syntax

`setCodePlacement(cfgObj, codePlacement)`

Description

`setCodePlacement(cfgObj, codePlacement)` specifies whether previously generated code retains the default folder structure for generated code, or has been repackaged to reside in a single, user-defined folder.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.
codePlacement	String specifying one of the following values: • `Single folder` if the generated code has been repackaged to reside in a single, user-defined folder. • `Embedded Coder default` (default) if the generated code resides in the default folders created by code generation.

Examples

In the following example, you call `slci.Configuration.setCodePlacement` to specify that generated code has been repackaged to reside in a single folder, and then call `slci.Configuration.setCodeFolder` to specify the folder path.

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setCodePlacement('Single folder')
>> config.setCodeFolder(fullfile('C:','packngo','model1'))
>> pkg = config.getCodePlacement()
pkg =
Single folder
>> folder = config.getCodeFolder()
folder =
```

```
C:\packngo\model1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setCodeFolder` | `slci.Configuration.getCodePlacement`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Purpose

Specify model reference handling for model compatibility checking or code inspection

Syntax

```
setFollowModelLinks(cfgObj, followModelLinks)
```

Description

`setFollowModelLinks(cfgObj, followModelLinks)` specifies whether model compatibility checking and code inspection should be performed for every descendant of this model in the model reference hierarchy.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.
followModelLinks	True if model compatibility checking and code inspection should be performed for every descendant of this model in the model reference hierarchy; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setFollowModelLinks(true)
>> value = config.getFollowModelLinks()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.getFollowModelLinks`

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Purpose

Specify whether to generate code before code inspection

Syntax

```
setGenerateCode(cfgObj, generateCode)
```

Description

`setGenerateCode(cfgObj, generateCode)` specifies whether to generate model code as part of code inspection.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.
generateCode	True if model code should be generated at the beginning of code inspection; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setGenerateCode(true)
>> value = config.getGenerateCode()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.getGenerateCode`

How To

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

slci.Configuration.setReportFolder

Purpose

Specify report folder for code inspection

Syntax

`setReportFolder(cfgObj, folder)`

Description

`setReportFolder(cfgObj, folder)` specifies a folder in which code inspection should place code inspection report artifacts.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.
folder	String specifying a folder path. If you have not previously set a report folder value, the default is `slprj/slci`, relative to the location of the model.

Examples

```
>> pwd
ans =
C:\work
>> config = slci.Configuration('mymodel');
>> folder = config.getReportFolder()
folder =
C:\work\slprj\slci
>> config.setReportFolder(fullfile('C:','work','mymodel_report'))
>> folder = config.getReportFolder()
folder =
C:\work\mymodel_report
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.getReportFolder`

How To

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

slci.Configuration.setTerminateOnIncompatibility

Purpose

Specify whether to terminate code inspection if model is incompatible

Syntax

```
setTerminateOnIncompatibility(cfgObj, terminate)
```

Description

`setTerminateOnIncompatibility(cfgObj, terminate)` specifies whether code inspection terminates if a model fails compatibility checking. If termination is selected, model code generation (if requested) also does not occur.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by `cfgObj = slci.Configuration(modelName);`.
terminate	True if code inspection should terminate if a model fails code inspection; false otherwise. The default is false.

Examples

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setTerminateOnIncompatibility(true)
>> value = config.getTerminateOnIncompatibility()
value =
     1
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.getTerminateOnIncompatibility` | `slci.Configuration.checkCompatibility`

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

slci.Configuration.setTopModel

Purpose

Specify whether model being configured for code inspection is top model

Syntax

`setTopModel(cfgObj, top)`

Description

`setTopModel(cfgObj, top)` specifies whether the model being configured for code inspection is the top model in the model reference hierarchy. If the model is not the top model, code inspection (and code generation if requested) uses a model reference target rather than a top model target.

Input Arguments

cfgObj	Handle to a Simulink Code Inspector configuration object previously returned by *cfgObj* = `slci.Configuration(`*modelName*`);`.
top	True if the model being configured for code inspection is the top model in the model reference hierarchy; false otherwise. The default is true.

Examples

The following example configures code inspection to use a model reference target.

```
>> config = slci.Configuration('slcidemo_roll');
>> config.setTopModel(false)
>> value = config.getTopModel()
value =
     0
>>
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.getTopModel`

How To

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

slci.Configuration

Purpose

Control code inspection and compatibility checking for model

Description

An `slci.Configuration` object configures code inspection and compatibility checking for a model.

Construction

slci.Configuration	Create code inspection object

Methods

checkCompatibility	Check model compatibility with code inspection
getCodeFolder	Return code folder for code inspection
getCodePlacement	Return code placement for code inspection
getFollowModelLinks	Return model reference handling for model compatibility checking or code inspection
getGenerateCode	Return code generation option for code inspection
getReportFolder	Return report folder for code inspection
getTerminateOnIncompatibility	Return termination option for code inspection
getTopModel	Return top-model attribute for code inspection
inspect	Inspect code generated from model
setCodeFolder	Specify code folder for code inspection

setCodePlacement	Specify code placement for code inspection
setFollowModelLinks	Specify model reference handling for model compatibility checking or code inspection
setGenerateCode	Specify whether to generate code before code inspection
setReportFolder	Specify report folder for code inspection
setTerminateOnIncompatibility	Specify whether to terminate code inspection if model is incompatible
setTopModel	Specify whether model being configured for code inspection is top model

Copy Semantics

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

Examples

The Simulink Code Inspector example `slcidemo_intro` shows how to programmatically run the compatibility checker and the Code Inspector and report results. The example also illustrates reporting of an error that is purposely introduced into the generated code.

See also the reference pages for `slci.Configuration.checkCompatibility`, `slci.Configuration.inspect`, and other `slci.Configuration` methods for individual call examples.

Alternatives

Open the Simulink Code Inspector dialog box from **Tools** menu of the model window and use the dialog box to configure and run model compatibility checks and code inspection.

How To

- "Check Model Compatibility Using the Graphical User Interface"

- “Check Model Compatibility Using the Command-Line Interface”
- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Purpose

Create code inspection object

Syntax

```
cfgObj = slci.Configuration(modelName)
```

Description

`cfgObj = slci.Configuration(modelName)` creates an object of class `slci.Configuration` and returns a handle to it.

Input Arguments

modelName	Name of the model for which you are configuring code inspection and compatibility checking.

Output Arguments

cfgObj	Handle to code inspection object.

Examples

This example creates a code inspection object, `config`, and uses it to check the specified model for compatibility with code inspection.

```
config = slci.Configuration('slcidemo_roll');
result = config.checkCompatibility('DisplayResults','None');

for i = 1:length(result)
    fprintf('\nModel ''%s'' passed %d checks with %d issues.',...
        result{i}.system,...
        result{i}.numPass, result{i}.numWarn + result{i}.numFail)
end
```

Alternatives

Open the Simulink Code Inspector dialog box from **Code** menu of the model window and use the dialog box to configure and run model compatibility checks and code inspection.

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"
- "Inspect Code Using the Graphical User Interface"

- “Inspect Code Using the Command-Line Interface”

Purpose

Generate XLS file that contains traceability matrix

Syntax

```
slci.ExportTraceReport('model_name')
slci.ExportTraceReport('model_name', 'file_name')
slci.ExportTraceReport('model_name', 'file_name', 'path')
```

Description

`slci.ExportTraceReport('model_name')` generates an XLS file that contains a "Traceability Matrix" on page 3-40. *`model_name`* is the name of the model.

`slci.ExportTraceReport('model_name', 'file_name')` generates an XLS file that contains a "Traceability Matrix" on page 3-40. *`file_name`* is a string that specifies the name of the XLS file. The first time that you call `slci.ExportTraceReport`, *`file_name`* is optional. If you do not provide *`file_name`*, the function names the file using the following convention. *`modelUpdate`* is the date and time that you last updated the model:

`model_name``_Trace_`*`modelUpdate`*`.xls`

To regenerate the traceability matrix, you must specify *`file_name`*.

`slci.ExportTraceReport('model_name', 'file_name', 'path')` generates an XLS file that contains a "Traceability Matrix" on page 3-40. *`path`* is an optional string that specifies the full path to the location where you want the software to save the file.

Tips

- The `slci.ExportTraceReport` function works in Microsoft® Windows® platforms only.
- To include requirements documentation in the traceability matrix, attach requirements documents to the model before using `slci.ExportTraceReport`.
- You must generate and inspect model code, with traceability report options selected, and without reported failures, before using `slci.ExportTraceReport`.
- The `slci.ExportTraceReport` function does not support generating a traceability matrix for referenced models. When you generate a

traceability matrix for a model that contains referenced models, the traceability matrix contains information about the Model block only. The traceability matrix does not contain information about the contents of the referenced model. If your model contains referenced models, generate a traceability matrix for the top-level model and each referenced model separately.

- In most cases, the `slci.ExportTraceReport` function identifies comments that you add to the traceability matrix. When the function cannot identify comments, the traceability matrix includes the text:

  Row is not unique: *`comment`*

For more information, see Prerequisites for Generating a Traceability Matrix.

Definitions

Traceability Matrix

A traceability matrix provides traceability among model objects, generated code, and model requirements. You can add comments to the generated traceability matrix. If you change the model and regenerate the traceability matrix, the software retains your comments.

Examples

Generate a traceability matrix with traceability between model objects and generated code for the `slcidemo_roll` model.

1. Open the example model `slcidemo_roll_orig` and save it to a work folder as `slcidemo_roll`.
2. Open the Configuration Parameters dialog box, and on the **Code Generation > Report** pane, verify that at least one traceability report option is selected.
3. Optionally, run model compatibility checks to verify that the model is ready for code inspection. For example, open the SLCI Advisor using the MATLAB command `slciadvisor('slcidemo_roll')`, select all checks, and run the checks.
4. Generate and inspect the model code.

5 Create a traceability matrix using a command similar to the following:

```
slci.ExportTraceReport('slcidemo_roll','slcidemo_roll_tracereport')
```

6 Open the file `slcidemo_roll_tracereport.xls` and examine the contents of the generated worksheets.

How To

- Traceability Matrices
- Prerequisites for Generating a Traceability Matrix
- Generate a Traceability Matrix

slciadvisor

Purpose

Open Simulink Code Inspector Advisor

Syntax

```
slciadvisor('model_name')
```

Description

`slciadvisor('model_name')` opens an SLCI Advisor session (equivalent to Model Advisor preloaded with Simulink Code Inspector checks) for the specified open model. This function provides direct access to SLCI model compatibility checking that can streamline iterative checking of a model.

Example

Open an interactive SLCI model compatibility checking session for the example model `slcidemo_roll_orig`.

1. Open the example model `slcidemo_roll_orig` and save it to a work folder as `slcidemo_roll`.
2. Open the SLCI Advisor for the model using the following command:

   ```
   >> slciadvisor('slcidemo_roll')
   ```

3. Select all SLCI checks, and run the checks.

How To

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"

4

Model Configuration Constraints

About Model Configuration Constraints

Simulink Code Inspector requires that you set a subset of Simulink configuration parameters and other model attributes to specific values. “Simulink Configuration Parameter Constraints” on page 4-4 presents required settings for Configuration Parameters Dialog Box parameters and their equivalent command-line parameters. “Other Modelwide Attribute Constraints” on page 4-18 presents required settings for other model attributes.

For each Configuration Parameters dialog pane or other model attributes category, a table provides:

- The category name; dialog pane names link to the complete dialog pane description
- Constraints that apply to each listed model configuration parameter or model attribute

A sample table is shown below. For each entry:

- The **Parameter** column lists the dialog box name of the parameter, with the command-line name of the parameter in parentheses. (For model attribute entries, the first column identifies the attribute.)
- The **Constraint** column lists the Simulink Code Inspector constraint on the model parameter or attribute.
- The **FATAL / Nonfatal** column identifies whether violation of the constraint terminates code inspection. You can also configure code inspection so that a constraint violation (FATAL or Nonfatal) terminates code inspection.
- The **Compatibility Check** column lists the compatibility check that checks for violation of the constraint, and links to a description of the check.

Solver Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Type (`SolverType`)	Must be set to `Fixed-step`.	Nonfatal	**Check solver settings > Verify 'Type' setting**
Solver (`Solver`)	Must be set to `Discrete (no continuous states)` (equivalent to `FixedStepDiscrete` specified at the command line).	Nonfatal	**Check solver settings > Verify 'Solver' setting**

Simulink Configuration Parameter Constraints

In this section...

Solver

Solver Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Type (`SolverType`)	Must be set to `Fixed-step`.	Nonfatal	**Check solver settings > Verify 'Type' setting**
Solver (`Solver`)	Must be set to `discrete (no continuous states)` (equivalent to `FixedStepDiscrete` specified at the command line).	Nonfatal	**Check solver settings > Verify 'Solver' setting**

Data Import/Export

Data Import/Export Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Initial state (`LoadInitialState`)	Must be cleared (set to `off`).	Nonfatal	**Check solver settings > Verify 'Initial state' setting**

Optimization

Optimization Pane: General			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Implement logic signals as Boolean	Must be selected (set to `on`).	Nonfatal	**Check optimization settings > Verify**

Optimization Pane: General			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
data (vs. double) (`BooleanDataType`)			**'Implement logic signals as Boolean data (vs. double)' setting**
Optimize initialization code for model reference (`OptimizeModelRef-InitCode`)	Must be selected (set to `on`).	Nonfatal	**Check optimization settings > Verify 'Optimize initialization code for model reference' setting**
Remove code from floating-point to integer conversions that wraps out-of-range values (`EfficientFloat2Int-Cast`)	Must be selected (set to `on`).	Nonfatal	**Check optimization settings > Verify 'Remove code from floating-point to integer conversions that wraps out-of-range values' setting**
Remove code from floating-point to integer conversions with saturation that maps NaN to zero (`EfficientMapNaN2Int-Zero`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Remove code from floating-point to integer conversions with saturation that maps NaN to zero' setting**
Remove code that protects against division arithmetic exceptions (`NoFixptDivByZero-Protection`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Remove code that protects against division arithmetic exceptions' setting**

Optimization: Signals and Parameters

Optimization Pane: Signals and Parameters			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Inline invariant signals (`InlineInvariant-Signals`)	Must be selected (set to `on`).	Nonfatal	**Check optimization settings > Verify 'Inline invariant signals' setting**
Simplify array indexing (`StrengthReduction`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Simplify array indexing' setting**
Pack Boolean data into bitfields (`BooleansAsBitfields`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Pack Boolean data into bitfields' setting**
Maximum stack size (bytes) (`MaxStackSize`)	Must be set to `inf`.	Nonfatal	**Check optimization settings>Verify 'Maximum stack size (bytes)' setting**
Loop unrolling threshold (`RollThreshold`)	Must be set to a value that does not result in partially unrolled loops in the generated code.	Nonfatal	**Check loop unrolling threshold setting>Verify loop unrolling threshold setting**
Pass reusable subsystem outputs as: (`PassReuseOutputArgsAs`)	Must be set to `Structure reference` if referenced model has root outports with non-auto storage class.	Nonfatal	**Check for root Outport blocks that have non-auto storage class >Verify that the storage class of root outports is supported**

Diagnostics: Data Validity

Diagnostics Pane: Data Validity			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Detect downcast (ParameterDowncastMsg)	Must be set to error.	Nonfatal	**Check diagnostic settings > Verify 'Detect downcast' setting**
Detect overflow (ParameterOverflowMsg)	Must be set to error.	Nonfatal	**Check diagnostic settings > Verify 'Detect overflow' setting**
Detect underflow (ParameterUnderflow-Msg)	Must be set to error.	Nonfatal	**Check diagnostic settings > Verify 'Detect underflow' setting**
Detect precision loss (ParameterPrecision-LossMsg)	Must be set to error.	Nonfatal	**Check diagnostic settings > Verify 'Detect precision loss' setting**
Detect loss of tunability (ParameterTunability-LossMsg)	Must be set to error.	Nonfatal	**Check diagnostic settings > Verify 'Detect loss of tunability' setting**
Underspecified initialization detection (Underspecified-Initialization-Detection)	Must be set to Simplified. Configuring the model to initialize block initial conditions using simplified behavior can improve the consistency of model results.	Nonfatal	**Check diagnostic settings > Verify 'Underspecified initialization detection' setting**

Diagnostics: Connectivity

Diagnostics Pane: Connectivity			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Bus signal treated as vector (`StrictBusMsg`)	Must be set to `error` (equivalent to `ErrorOnBusTreatedAsVector` specified at the command line).	FATAL	**Check diagnostic settings > Verify Bus signal treated as vector setting**
Non-bus signals treated as bus signals (`NonBusSignalsTreatedAsBus`)	Must be set to `error`.	FATAL	**Check diagnostic settings > Verify 'Non-bus signals treated as bus signals' setting**

Diagnostics: Model Referencing

Diagnostics Pane: Model Referencing			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Invalid root Inport/Outport block connection (`ModelReferenceIOMsg`)	Must be set to `error`. This setting disallows automatic insertion of hidden signal copy blocks at the model inports and outports. If an error is generated, it identifies the locations at which you can manually insert Signal Conversion blocks to avoid the error and maintain traceability.	Nonfatal	**Check diagnostic settings > Verify 'Invalid root Inport/Outport block connection' setting**

Hardware Implementation

Hardware Implementation Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Number of bits: char (ProdBitPerChar)	Must be set to 8.	Nonfatal	**Check hardware implementation settings > Verify 'char' setting**
Number of bits: short (ProdBitPerShort)	Must be set to 16.	Nonfatal	**Check hardware implementation settings > Verify 'short' setting**
Number of bits: int (ProdBitPerInt)	Must be set to 32.	Nonfatal	**Check hardware implementation settings > Verify 'int' setting**
Number of bits: long (ProdBitPerLong)	Must be set to 32.	Nonfatal	**Check hardware implementation settings > Verify 'long' setting**
Number of bits: float (ProdBitPerFloat)	Must be set to 32.	Nonfatal	**Check hardware implementation settings > Verify 'float' setting**
Number of bits: double (ProdBitPerDouble)	Must be set to 64.	Nonfatal	**Check hardware implementation settings > Verify 'double' setting**
Number of bits: native (ProdWordSize)	Must be set to 32.	Nonfatal	**Check hardware implementation settings > Verify 'native' setting**
Number of bits: pointer (ProdBitPerPointer)	Must be set to 32.	Nonfatal	**Check hardware implementation settings > Verify 'pointer' setting**
Signed integer division rounds to (ProdIntDivRoundTo)	Must be set to Zero.	Nonfatal	**Check hardware implementation settings > Verify 'Signed integer division rounds to' setting**

Hardware Implementation Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Shift right on a signed integer as arithmetic shift (`ProdShiftRightInt-Arith`)	Must be selected (set to `on`).	Nonfatal	**Check hardware implementation settings > Verify 'Shift right on a signed integer as arithmetic shift' setting**
None (`ProdEqTarget`)	Must be selected (set to `on`).	Nonfatal	**Check hardware implementation settings > Verify 'None' setting**
• **Device vendor** • **Device type** (`ProdHWDeviceType`)	Must not be set to `ASIC/FPGA`.	Nonfatal	**Check hardware implementation settings>Verify 'Device vendor->Device type' setting**

Code Generation: General

Code Generation Pane: General			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
System target file (`SystemTargetFile`)	Must be set to `ert.tlc` or the system target file for an ERT-derived target.	FATAL	**Check system target file setting**
Language (`TargetLang`)	Must be set to `C` or `C++`.	FATAL	**Check code generation settings > Verify 'Language' setting**
TLC options (`TLCOptions`)	Must be unspecified (set to `''`).	Nonfatal	**Check code generation settings > Verify 'TLC options' setting**

Code Generation: Comments

Code Generation Pane: Comments			
Parameter	Constraint	FATAL / Nonfatal	Compatibility Check
Include comments (`GenerateComments`)	Must be selected (set to `on`). The Code Inspector parses autogenerated comments to obtain traceability information about model data.	FATAL	**Check code generation settings > Verify 'Include comments' setting**

Code Generation: Symbols

Code Generation Pane: Symbols			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Generate scalar inlined parameter as (`InlinedPrmAccess`)	Must be set to `Literals`.	Nonfatal	**Check code generation settings > Verify 'Generate scalar inlined parameter as' setting**
Signal naming (`SignalNamingRule`)	Must be set to `None`.	Nonfatal	**Check code generation settings>Verify 'Signal naming' setting**
Parameter naming (`ParamNamingRule`)	Must be set to `None`.	Nonfatal	**Check code generation settings>Verify 'Parameter naming' setting**

Code Generation: Custom Code

Code Generation Pane: Custom Code			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Source file (`CustomSourceCode`)	Must be unspecified (set to `''`).	FATAL	**Check code generation settings > Verify 'Source file' setting**
Header file (`CustomHeaderCode`)	Must be unspecified (set to `''`).	Nonfatal	**Check code generation settings>Verify 'Header file' setting**
Initialize function (`CustomInitializer`)	Must be unspecified (set to `''`).	Nonfatal	**Check code generation settings > Verify 'Initialize function' setting**
Terminate function (`CustomTerminator`)	Must be unspecified (set to `''`).	Nonfatal	**Check code generation settings > Verify 'Terminate function' setting**

Code Generation: Interface

Code Generation Pane: Interface			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Code replacement library (`CodeReplacement-Library`)	Must be set to `C89/C90 (ANSI)`, `ANSI_C`, `C99 (ISO)` , or `ISO_C` in the Configuration Parameters dialog box. You can also use "Supported Functions and Operations in Code	Nonfatal	**Check code generation settings > Verify 'Code replacement library' setting**

Code Generation Pane: Interface			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
	Replacement Libraries" on page 4-22.		
Shared code placement (`UtilityFuncGeneration`)	Must be set to `Shared location`. Using a shared location for utility functions, macros, and user-defined data types promotes debugging and traceability of generated code.	Nonfatal	**Check code generation settings>Verify 'Shared code placement' setting**
Support: non-finite numbers (`SupportNonFinite`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'non-finite numbers' setting**
Support: absolute time (`SupportAbsoluteTime`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'absolute time' setting**
Classic call interface (`GRTInterface`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'Classic call interface' setting**
Single output/update function (`CombineOutputUpdate-Fcns`)	Must be selected (set to `on`).	Nonfatal	**Check code generation settings > Verify 'Single output/update function' setting**
Terminate function required (`IncludeMdlTerminate-Fcn`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'Terminate function required' setting**
Generate reusable code (`MultiInstanceERTCode`)	Must be selected (set to `on`). This check applies only to the top model in a model hierarchy.	Nonfatal	**Check code generation settings > Verify 'Generate reusable code' setting**

Code Generation Pane: Interface			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Pass root-level I/O as (`RootIOFormat`)	Must be set to `Individual arguments`. This check applies only to the top model in a model hierarchy.	Nonfatal	**Check code generation settings > Verify 'Pass root-level I/O as' setting**
Suppress error status in real-time model data structure (`SuppressErrorStatus`)	Must be selected (set to `on`). This helps prevent generation of the `rtModel` data structure, which is not supported for code inspection.	Nonfatal	**Check code generation settings > Verify 'Suppress error status in real-time model data structure' setting**
Combine signal/state structures (`CombineSignalStateStructs`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'Combine signal/state structures' setting**
MAT-file logging (`MatFileLogging`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'MAT-file logging' setting**
Interface (`RTWCAPIParams`, `RTWCAPISignals`, `RTWCAPIStates`, `RTWCAPIRootIO`, `ExtMode`, and `GenerateASAP2`)	Must be cleared (`RTWCAPIParams`, `RTWCAPISignals`, `RTWCAPIStates`, `RTWCAPIRootIO`, `ExtMode`, and `GenerateASAP2` must be set to `off`).	FATAL	**Check code generation settings > Verify Code Generation > Interface > Interface setting**

Code Generation: Verification

"Code Generation Pane: Verification"			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Create block (`CreateSILPILBlock`)	Must be set to `None`.	Nonfatal	**Check code generation settings > Verify 'Create block' setting**
Measure function execution times (`CodeProfiling-Instrumentation`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify 'Measure function execution times' setting**

Code Generation: Code Style

Code Generation Pane: Code Style			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Preserve condition expression in if statement (`PreserveIfCondition`)	Must be selected (set to `on`).	Nonfatal	**Check code generation settings > Verify 'Preserve condition expression in if statement' setting**

Code Generation: Data Type Replacement

Code Generation Pane: Data Type Replacement			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Replace data type names in the generated code (`EnableUser-ReplacementTypes`)	Must be cleared (set to `off`). Data type replacement is not supported for code inspection.	Nonfatal	**Check code generation settings > Verify 'Replace data type names in the generated code' setting**

Code Generation: Not in GUI

Parameter Command-Line Information Summary			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
`AdvancedOptControl`	Should be set to `-SLCI`. This setting disables optimizations that are incompatible with Simulink Code Inspector.	Nonfatal	**Check optimization settings > Verify 'AdvancedOptControl' setting**
`IncludeERTFirstTime`	Must be set to `off`.	Nonfatal	**Check code generation settings > Verify 'IncludeERTFirstTime' setting**

Other Modelwide Attribute Constraints

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Unconnected objects	There must not be unconnected lines, input ports, or output ports in the model or subsystem. This helps prevent dead code and hidden ground blocks.	Nonfatal	**Check for unconnected objects in the model**
Function specifications	The model cannot specify custom model entry function prototypes. **Function specification** in the Model Interface dialog box must be set to `Default model initialize and step functions`.	Nonfatal	**Check function specification setting**
Conditional input branch execution	The model must enable signal storage reuse and local block outputs when using conditional input branch execution.	Nonfatal	**Check conditional input branch execution setting**
Unsupported blocks	There must not be blocks in the model that are not supported by Simulink Code Inspector.	Nonfatal	**Check for unsupported blocks**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Storage classes for workspace variables	The model cannot reference workspace variables that are not supported for one or both of these reasons: • The "Custom Storage Classes" **Type** is not set to `Unstructured`. • Workspace variable is tunable, with **data type** set to `struct`. .	Nonfatal	**Check storage class for workspace variables**
Usage of sample times	The model cannot use multiple, variable, continuous, or asynchronous sample times.	FATAL	**Check for sample times in the model**
Automatic insertion of Signal Conversion blocks on signals entering block inports	Automatic insertion of a Signal Conversion block on a signal entering a block inport is not supported for code inspection. It creates a hidden Signal Conversion block, which is not supported for code inspection.	Nonfatal	**Check for Signal Conversion blocks automatically inserted on signals entering block input ports > Verify no Signal Conversion blocks are automatically inserted on signals entering block inports**
Fixed-point instrumentation and block reduction both selected	Simultaneous use of fixed-point instrumentation and block reduction is not supported for code inspection.	Nonfatal	**Check for usage of fixed-point instrumentation > Verify usage of fixed-point instrumentation**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Conditional assignment of root Outport blocks	If the root outport storage class is set to `Auto`, when it used in a referenced model, it cannot be directly connected to conditionally executed subsystems.	Nonfatal	**Check for root Outport blocks being conditionally assigned**
Root Outport block sample times	Root Outport blocks cannot specify a constant (`Inf`) sample time. This constraint prevents the root outport assignment from being moved to the model initialize function, which would cause the model functions to fail validation.	Nonfatal	**Check usage of root Outport blocks > Verify sample times**
Root Output block bus passing method	A root Outport block that passes a bus to a parent model must pass the bus as a structure. Otherwise, Simulink software might insert a hidden Signal Conversion block in the parent model, which is not supported for code inspection.	Nonfatal	**Check usage of root Outport blocks > Verify root Outports pass buses to parent models as structures**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Automatic virtual to nonvirtual bus conversion	Automatic conversion between virtual and nonvirtual buses is not supported for code inspection. It creates a hidden Signal Conversion block, which is not supported for code inspection.	FATAL	**Check usage of buses > Check for automatic conversion between virtual to non-virtual buses**
Block operations on a bus	A nonvirtual block cannot operate on a virtual bus, and a Unit Delay block cannot operate on a bus (virtual or nonvirtual). This constraint simplifies bus processing to promote traceability and readability of generated code.	FATAL	**Check usage of buses > Verify that no blocks in the model operate on a virtual bus**

Supported Functions and Operations in Code Replacement Libraries

Simulink Code Inspector inspects code that uses these functions and operations in the code replacement libraries (CRLs). For more information about CRLs, see "Code Replacement".

Math Functions			
abs	acos	acosh	asin
asinh	atan	atan2	atanh
ceil	cos	cosh	exp
fix	floor	hypot	log
log10	max	min	mod/fmod
pow	rem	round	saturate
sin	sincos	sinh	sqrt
tan	tanh		

Operator	Key	Scalar Inputs	Nonscalar Inputs
Multiplication (*)	RTW_OP_MUL	—	Yes
Matrix right division (/)	RTW_OP_RDIV[1]	—	Yes
Matrix left division (\)	RTW_OP_LDIV[1]	—	Yes
Matrix inversion (inv)	RTW_OP_INV[1]	—	Yes
Transposition (.')	RTW_OP_TRANS	—	Yes

Notes:
[1] Matrix division and inversion are supported for Simulink code generation (not for Stateflow® or MATLAB Coder™ code generation).

5

Block Constraints

About Block Constraints

Simulink Code Inspector supports a subset of Simulink blocks for code inspection. For the supported blocks, some block-specific constraints on data types and block parameters may apply. Additionally, a few constraints apply to all supported blocks. Before code inspection, when you check the compatibility of your model with code inspection rules, the compatibility checker detects and reports violations of block constraints.

"Block Constraints — Alphabetical List" on page 5-5 presents the supported blocks in alphabetical order. For each supported block, a table provides:

- The block name, which links to the complete block description
- Data type constraints that apply to the block
- Block parameter constraints that apply to the block

A sample table is shown below. For each entry:

- The **Constraint** column lists the Simulink Code Inspector constraint on block data types or a block parameter. For block parameters, the entry lists the dialog box name of the parameter, with the command-line name of the parameter in parentheses.
- The **FATAL / Nonfatal** column identifies whether violation of the constraint terminates code inspection. You can also configure code inspection so that constraint violation (FATAL or Nonfatal) terminates code inspection.
- The **Compatibility Check** column lists the compatibility check that checks for violation of the constraint, and links to a description of the check.

Saturation			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Discontinuities blocks > Check Saturate blocks**
	Input and output ports should have the same data type.	Nonfatal	
Block Parameters	**Upper limit** (`UpperLimit`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Lower limit (`LowerLimit`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	The source of the upper limit value must be block parameter **Upper limit** rather than input ports (`UpperLimitSource` must be set to `dialog`).	Nonfatal	
	The source of the lower limit value must be block parameter **Lower limit** rather than input ports (`LowerLimitSource` must be set to `dialog`).	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

"All Blocks" on page 5-7 lists constraints that apply to supported blocks.

"Supported Blocks — By Category" on page 5-49 presents the supported blocks by category and provides links to the block-specific constraints.

Note Blocks that are supported for code inspection are available in the block library `slcilib`, which you can open by entering `slcilib` in the MATLAB Command Window.

Block Constraints — Alphabetical List

In this section...

In this section...

In this section...

All Blocks

Constraints that apply to all blocks			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Input and output ports must be of data types among the following: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`, or `Enumerated` with default value `0`. If the block supports buses: • Ports can be buses for which the elements (potentially including other buses) meet the data type constraint. • Ports must not have arrays of buses. • Ports must not have buses with elements that are arrays of buses.	Nonfatal	All block compatibility checks

Constraints that apply to all blocks			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Other	Block names must not contain character strings `*/` or `/*`. Additionally, block names must not end with the character `*`.	Nonfatal	
	Input and output ports must be noncomplex. Complex values are not supported for code inspection.	Nonfatal	
	Input and output ports must be scalars, vectors, or 2D matrices.	Nonfatal	
	Input and output ports must not use frame-based signals.	Nonfatal	
	Output custom signal storage class must be set to `Unstructured`.	Nonfatal	
	Output port must not be testpointed when the block has constant (`Inf`) sample time.	Nonfatal	
	Output signal storage class must be set to `Auto` when the block has constant (`Inf`) sample time.	Nonfatal	

Abs

Abs			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Absolute blocks**
	Input and output ports should have the same data type.	Nonfatal	

Abs			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Block Parameters	**Integer rounding mode** (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Action Port

Action Port			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks >Check Action Port blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Bitwise Operator

Bitwise Operator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Logical and Bit Operations blocks > Check Bitwise Operator blocks**
	If **Number of input ports** (`NumInputPorts`) is `1` and **Operator** (`logicop`) is set to `AND`, `OR`, `NAND`, `NOR`, or `XOR`, the inport data type must be scalar. If the	Nonfatal	

Bitwise Operator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Use bit mask (`Usebitmask`) check box is selected, you cannot specify the **Number of input ports**.		
Block Parameters	No block-specific constraints		

Bus Assignment

Bus Assignment			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Bus Assignment blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Bus Creator

Bus Creator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Bus Creator blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Bus Selector

Bus Selector			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Bus Selector blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Constant

Constant			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sources blocks > Check Constant blocks**
	No block-specific constraints		
Block Parameters	**Constant value** (`Value`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	

Data Store Memory

Data Store Memory			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to allblocks.		**Check usage of Signal Routing blocks > Check Data Store Memory blocks**
	State must have storage class `Auto`. Values other than `Auto` require use of storage classes, which are not supported for code inspection.	Nonfatal	
Block Parameters	**Initial value** (`InitialValue`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Signal type (`SignalType`) must be set to `auto` or `real`. Complex values are not supported for code inspection.	Nonfatal	
Other	For global data store memory, configuration parameter **Optimization > Signals and Parameters > Inline parameters** (`InlineParams`) must be selected (set to `on`).	Nonfatal	**Check usage of global data stores >Verify global data store usage**
	For global data store memory, **Initial value** (`InitialValue`) must not be tunable.	Nonfatal	

Data Store Read

Data Store Read			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Data Store Read blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify elements. **Specify element(s) to select** (`DataStoreElements`) must be `''`.	Nonfatal	
	The block cannot reference signal objects as synthesized local data stores.	Nonfatal	**Check for usage of synthesized local data stores >Verify synthesized local data store usage**
Other	For global data store memory, configuration parameter **Optimization > Signals and Parameters >Inline parameters** (`InlineParams`) must be selected (set to `on`).	Nonfatal	**Check usage of global data stores >Verify global data store usage**
	For global data store memory, **Initial value** (`InitialValue`) must not be tunable.	Nonfatal	

Data Store Write

Data Store Write			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Data Store Write blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify elements. **Specify element(s) to select** (`DataStoreElements`) must be `''`.	Nonfatal	
	The block cannot reference signal objects as synthesized local data stores.	Nonfatal	**Check for usage of synthesized local data stores >Verify synthesized local data store usage**
Other	For global data store memory, configuration parameter **Optimization > Signals and Parameters >Inline parameters** (`InlineParams`) must be selected (set to `on`).	Nonfatal	**Check usage of global data stores >Verify global data store usage**
	For global data store memory, **Initial value** (`InitialValue`) must not be tunable.	Nonfatal	

Data Type Conversion

Data Type Conversion			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Data Type Conversion blocks**
	No block-specific constraints		
Block Parameters	**Input and output to have equal** (`ConvertRealWorld`) must be `Real World Value (RWV)`.	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Data Type Duplicate

Data Type Duplicate			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Data Type Duplicate blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Data Type Propagation

Data Type Propagation			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Data Propogation blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Discrete-Time Integrator

Discrete-Time Integrator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Discrete blocks > Check Discrete Integrator blocks**
	Input ports data types must be: • `single` or `double` for non-reset	Nonfatal	
	Inports and outports must be scalars	Nonfatal	
	Output ports data types must `single` or `double`	Nonfatal	
	Except for the reset port, input and output ports should have the same	Nonfatal	
Block Parameters	Block parameter **Integrator method** (`IntegratorMethod`) must be set to one of the following: • `Integration: Forward Euler` • `Integration: Backward Euler`	Nonfatal	

Discrete-Time Integrator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	• `Integration: Trapezoidal`		
	Block parameter **Show state port** (`ShowStatePort`) must not be selected (must be set to `off`).	Nonfatal	
	If **External reset** (`ExternalReset`) is not set to `none`, the source of Inport 2 must not: • Be a Constant block. • Have a constant sample time.	Nonfatal	
	Block parameters **Upper saturation limit** (`UpperSaturationLimit`) and **Lower saturation limit** (`LowerSaturationLimit`) must not: • Be empty, non-finite, or complex. • Use MATLAB structures. • Have two or more dimensions. • Specify the `:` operator.	FATAL	
Other	Block must not be inside a conditional subsystem.	Nonfatal	

Demux

Demux			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Demux blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

DocBlock

DocBlock			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	No block-specific constraints		Not applicable
Block Parameters	No block-specific constraints		

Enable Port

Enable Port			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks > Check Enable Port blocks**
	The signal entering an enable port of a subsystem must be of data type `boolean`.	Nonfatal	
Block Parameters	**Show output port** (`ShowOutputPort`) must not be selected (must be set to `off`).	Nonfatal	

Enable Port			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Enable Port blocks are not supported at the root level of the model.	FATAL	
	The signal entering the Enable Port of the parent subsystem must not: • Be from a Constant block. • Have a constant sample time.	Nonfatal	

From

From			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check From blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Function-Call Generator

Function-Call Generator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks>Check Function Call Generator blocks**
	No block-specific constraints		
Block Parameters	The **Number of iterations** (`numberOfIterations`) must be set to `1`.		

Gain

Gain			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Gain blocks**
	Input and output ports should have the same data type.	Nonfatal	
Block Parameters	**Gain** (`Gain`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Parameter data type (`ParamDataTypeStr`) must use the same data type as the Gain block input.	Nonfatal	
	Multiplication (`Multiplication`) must be set to `Element-wise(K.*u)`, `Matrix(K*u)`, `Matrix(u*K)`, or `Matrix(K*u)(u vector)`. **Note** Only `single` or `double` data types are supported for `Matrix(K*u)`, `Matrix(u*K)`, or `Matrix(K*u)(u vector)` multiplications methods.	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Goto

Goto			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Goto blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Ground

Ground			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sources blocks > Check Ground blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

If

If			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks >Check If blocks**
	Block destination must not be a terminator block or an empty action subsystem.	Nonfatal	

If			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Block Parameters	Source of Inport 1 must not: • Be a Constant block. • Have a constant sample time.	Nonfatal	

Inport

Inport			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sources blocks > Check Inport blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify variable-dimension signals. **Variable-size signal** (`VarSizeSig`) must *not* be set to `Yes`.	Nonfatal	
	Signal Type (`NumberOfTableDimensions`) must not be set to `complex`.	Nonfatal	
	Sampling Mode (`SamplingMode`) must not be set to `Frame based`.	Nonfatal	
	For inports in triggered subsystems, **Latch input be delaying outside signal** (`LatchByDelayingOutsideSignal`) must not selected (must be set to `off`).	Nonfatal	

Note Shadowed inports are supported for code inspection.

Logical Operator

Logical Operator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Logical and Bit Operations blocks > Check Logic blocks**
	Output ports must be of the data type `boolean` or `uint8`.	Nonfatal	
	Block must have at least two inports, except in the case of the NOT operator.	FATAL	
	The block input ports should have the same data type.	Nonfatal	
Block Parameters	No block-specific constraints		

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Lookup Tables blocks > Check Lookup Table blocks**
	Input and output ports should have the same data type.	Nonfatal	
	Input and output ports must be scalars.	Nonfatal	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Block Parameters	**Number of table dimensions** (`NumberOfTableDimensions`) must be set to 1 or 2.	Nonfatal	
	Interpolation method (`InterpMethod`) must be set to `Linear`.	Nonfatal	
	Extrapolation method (`ExtrapMethod`) must be set to `Clip` or `Linear`.	Nonfatal	
	Index search method (`IndexSearchMethod`) must be set to `Binary search`.	Nonfatal	
	Begin index search using previous index result (`BeginIndexSearchUsingPreviousIndexResult`) must not be selected (must be set to `off`).	Nonfatal	
	Support tunable table size in code generation (`SupportTunableTableSize`) must not be selected (must be set to `off`).	Nonfatal	
	Remove protection against out-of-range input in generated code (`RemoveProtectionInput`) must be selected (must be set to `on`).	Nonfatal	
	Saturate on integer overflow (`SaturateOnIntegerOverflow`) must not be selected (must be set to `off`).	Nonfatal	
	Fraction > Data Type (`FractionDataTypeStr`) must be set to `double` or `single`.	Nonfatal	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Table data (`Table`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Breakpoints 1 (`BreakpointsForDimension1`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Breakpoints 2 (`BreakpointsForDimension2`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Breakpoints 1 (`BreakpointsForDimension1DataTypeStr`) must use the same data type as the block input.	Nonfatal	
	Breakpoints 2 (`BreakpointsForDimension2DataTypeStr`) must use the same data type as the block input.	Nonfatal	
	Table data (`TableDataTypeStr`) must use the same data type as the block output.	Nonfatal	
	Intermediate Results (`IntermediateResultsDataTypeStr`)	Nonfatal	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	must use the same data type as the block output.		
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Math Function

Math Function			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Math blocks**
	Input and output ports should have the same data type.	Nonfatal	
Block Parameters	**Function** (`Operator`) must be set to one of the following values: `exp`, `log`, `10^u`, `log10`, `magnitude^2`, `square`, `transpose`, `pow`, `reciprocal`, `hypot`, `rem`, or `mod`. You cannot select `conj` or `hermitian`.	FATAL	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Merge

Merge			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks >Check Merge blocks**
Block Parameters	**Initial output** (`InitialOutput`) must be `0`.	Nonfatal	
	Allow unequal port widths (`AllowUnequalInputPortWidths`) must not be selected (must be set	Nonfatal	
	Input port offsets (`InputPortOffsets`) must be `[]`.	Nonfatal	

MinMax

MinMax			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Minmax blocks**
	Input and output ports must be of a data type among the following: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.	FATAL	
	Input and output ports should have the same data type.	Nonfatal	
	Block must have at least two inports.	FATAL	
Block Parameters	**Integer rounding mode** (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Model

Model			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks > Check Model Reference blocks**
	No block-specific constraints		
Block Parameters	The block cannot have variants. **Enable variants** (`Variant`) must not be selected (must be set to `off`).	Nonfatal	

Model Info

Model Info			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	No block-specific constraints		Not applicable
Block Parameters	No block-specific constraints		

Multiport Switch

Multiport Switch			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Multiport Switch blocks**
	Data input and output ports must have the same data type.	Nonfatal	
	Block must have at least three inports.	FATAL	

Multiport Switch			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Block Parameters	If data port indices are specified for a Multiport Switch block, there can be only one value specified per input.	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	
	Allow different data input sizes (`AllowDiffInputSizes`) must not be selected (must be set to `off`).	Nonfatal	
	Data port for default case (`DataPortForDefault`) must be set to `Last data port`.	Nonfatal	

Mux

Mux			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Mux blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Outport

Outport			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sinks blocks > Check Outport blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify variable-dimension signals. **Variable-size signal** (`VarSizeSig`) must *not* be set to `Yes`.	Nonfatal	
	Signal type (`NumberOfTableDimensions`) must *not* be set to `complex`.	Nonfatal	
	Sampling mode (`SamplingMode`) must *not* be set to `Frame based`.	Nonfatal	
	Root level outport **Initial output** (`InitialOutput`) must be `[]`.	Nonfatal	
	Source of initial output value (`SourceOfInitialOutputValue`) must be set to `Dialog`.	Nonfatal	

Probe

Probe			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Probe blocks**
	No block-specific constraints		
Block Parameters	The block parameter **Data type for width** (`ProbeWidthDataType`) must not be `boolean`.	Nonfatal	

Probe			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	The block parameter **Data type for signal dimensions** (`ProbeDimensionsDataType`) must not be `boolean`.	Nonfatal	
	The block parameter **Data type for sample time** (`ProbeSampleTimeDataType`) must be `single` or `double`.	Nonfatal	

Product

Product			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Product blocks**
	Input and output ports should have the same data type.	Nonfatal	
Block Parameters	**Multiplication** (`Multiplication`) must be set to `Element-wise(.*)` or `Matrix (*)`. **Note** Only `single` or `double` data types are supported for `Matrix (*)` multiplication.	Nonfatal	
	Block parameter **Number of inputs** (`inputs`) must be set to `2`, `**`, `/*`, `*/`, `//`, or `/` when both of the following are true: • Inport **Signal type** is a matrix.	Nonfatal	

Product			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	• Product block parameter **Multiplication** is set to `Matrix (*)`.		
	Block parameter **Number of inputs** (`inputs`) must be set to `2`, `**`, `/*`, `*/`, or `//` when both of the following are true: • Inport **Signal type** is a scalar or vector. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.	Nonfatal	
	Block parameter **Number of inputs** (`inputs`) must be set to `/` when both of the following are true: • Inport **Signal type** is a scalar. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Relational Operator

Relational Operator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Logical and Bit Operations blocks > Check Relational Operator blocks**
	Block output port data type must be either an enumerated type with default value 0, or `boolean`.	FATAL	
	Block input ports should have the same data type.	Nonfatal	
Block Parameters	**Relational operator** (`Operator`) must be set to `<=`, `==`, `>=`, `~=`, `<`, or `>` (not `isInf`, `isNaN`, or `isFinite`).	FATAL	

Reshape

Reshape			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks> Check Reshape blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Rounding Function

Rounding Function			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to allblocks.		**Check usage of Math Operations blocks > Check Rounding Function blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Saturation

Saturation			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Discontinuities blocks > Check Saturate blocks**
	Input and output ports should have the same data type.	Nonfatal	
Block Parameters	**Upper limit** (`UpperLimit`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	Lower limit (`LowerLimit`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	
	The source of the upper limit value must be block parameter **Upper limit** rather than input ports	Nonfatal	

Saturation			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	(`UpperLimitSource` must be set to `dialog`).		
	The source of the lower limit value must be block parameter **Lower limit** rather than input ports (`LowerLimitSource` must be set to `dialog`).	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Selector

Selector			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Selector blocks**
	Inports and outports must be scalars or vectors.	Nonfatal	
Block Parameters	Must use one-dimensional inputs and must specify indices using the block dialog (not using port-based indexing).	Nonfatal	

S-Function

Note Simulink Code Inspector supports S-functions created using the Legacy Code Tool.

S-Function			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of User-Defined Function blocks > Check S-Function blocks**
	Arguments must be scalars, or vectors of fixed dimension.	Nonfatal	
Block Parameters	S-functions: • Must be created using the Legacy Code Tool. • Can only specify an `OutputFcnSpec` (not `InitializeConditionsFcnSpec`, `StartFcnSpec`, or `TerminateFcnSpec`). • Can not have more than one `dwork`. **Note** When you use the Legacy Code Tool to define an S-Function prototype, the: • Data must be a scalar or a one-dimensional vector. Do not use a two-dimensional vector. For example, use `u[6]`, not `u[2][3]`. • Dimension must be explicitly set. For example, use `u[6]`, not `u[]`.	Nonfatal	

Shift Arithmetic

Shift Arithmetic			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Logical and Bit Operations blocks > Check ArithShift blocks**
	No block-specific constraints		
Block Parameters	**Diagnostic for out of range shift value** (`DiagnosticForOORShift`) must be set to `Error`.	Nonfatal	
	Binary points to shift (`BinPtShiftNumber`) must be set to `0`.	Nonfatal	
	Bits to shift: Number (`BitShiftNumber`) must be within the allowable range of the inport data type.	Nonfatal	
	If **Bits to shift: Source** (`BitShiftNumberSource`) is set to `Input port` and **Bits to shift: Direction** (`BitShiftDirection`) is set to `Bidirectional`, the source of Inport 2 must not: • Be a Constant block. • Have a constant sample time.	Nonfatal	

Sign

Sign			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks>Check Sign blocks**
	No block-specific constraints		
Block Parameters	No block-specific constraints		

Signal Conversion

Signal Conversion			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Signal Conversion blocks**
	No block-specific constraints		
Block Parameters	**Output** (`ConversionOutput`) must be set to `Signal copy`.	Nonfatal	

Signal Specification

Signal Specification			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Attributes blocks > Check Signal Specification blocks**
	No block-specific constraints		
Block Parameters	**Variable-size signal** (`VariableSizeSignal`) must be `No`.	Nonfatal	

Signal Specification			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Signal type (`SignalType`) must not be `complex`.	Nonfatal	
	Sampling mode (`SamplingMode`) must not be `Frame based`.	Nonfatal	

Sqrt

Sqrt			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Sqrt blocks**
	Block inputs and outports must have the same data type.	Nonfatal	
	Block inputs and outports data types must be `single` or `double`.	FATAL	
Block Parameters	**Function** (`Operator`) must be `sqrt` or `signedSqrt`.	Nonfatal	
	Output signal type (`OutputSignalType`) must not be set to `complex`.	Nonfatal	

Stateflow

Stateflow			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Stateflow blocks	Constraints that apply to all blocks.		**Check usage of Stateflow blocks**
	Function packaging (`RTWSystemCode`) must be set to `Inline`.		
Stateflow Data and Event Types	Stateflow data must not be of machine scope.	Nonfatal	**Check for Stateflow machine data >All Stateflow data must be parented by a Stateflow chart**
	Stateflow events must not be of machine scope.	Nonfatal	**Check for Stateflow machine events >All Stateflow events must be parented by a Stateflow chart**
Stateflow Charts	The chart must not contain control flow cycles.	FATAL	**Check usage of Stateflow charts**
	The chart must not contain any of the following objects: • States • Subcharts • Graphical functions • MATLAB functions • Truth Tables • Simulink functions	FATAL	
	Chart property **Action Language** must be set to `C`.	Nonfatal	
	Chart property **Update method** must be set to `Inherited`.	Nonfatal	

Stateflow			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Chart property **Execute (enter) Chart at Initialization** must not be selected.	FATAL	
	Chart property **Saturate on integer overflow** must not be selected.	Nonfatal	
	Chart property **Support variable-size arrays** must not be selected.	Nonfatal	
	The chart must not contain unstructured control flow.	FATAL	
	The control flow must not have more than 1 default transition.	Nonfatal	
Stateflow transitions	Transition must be for one of these operations: • `:=` or `=` • `+` , `+=` , `-`, or `-=` • `*` , `*=`, `/` or `/=` • `&`, `&&` or `&=` • `|`, `||` or `|=` • `<<`, `>>`, `++` or `--` • `cast()` • `^` or `^=` • `%%` or `<` • `<=` or `==` • `~=` or `!=` • `<>` or `>`	Nonfatal	**Check usage of Stateflow transitions**

Stateflow			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	• >= or ~		
	Transition must not access context-sensitive constants.	Nonfatal	
	Transition must not access custom data.	Nonfatal	
	Transitions must not have an action.	Nonfatal	
	Transition must not contain a binary operator with mixed data type operands.	Nonfatal	
	Transition must not access time.	Nonfatal	
Stateflow Junctions	Non-terminating junctions must have exactly one unconditional transition exiting them.	FATAL	**Check usage of Stateflow junctions**
	Chart must not contain a history junction.	Nonfatal	
	Unconditional transition must be last in order of execution.	Nonfatal	
Stateflow Data	Chart must not use constants.	Nonfatal	**Check usage of Stateflow data**
	Chart must not use data stores	Nonfatal	
	Chart data types must be `builtin`, `enumerated`, or `bus`. If the chart data type is a bus, the data must not be arrays of buses or have elements that are arrays of buses.	Nonfatal	
	Chart must not use data with initial values.	Nonfatal	
	Chart must not use local data.	Nonfatal	

Stateflow			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Chart must not use parameters.	Nonfatal	
	Chart must not use complex data.	Nonfatal	
	Chart must not use non-scalar data.	Nonfatal	
Stateflow Events	Event scope must be an `Output`.	Nonfatal	**Check usage of Stateflow events**
	Event trigger must be a `function-call`.	Nonfatal	

Subsystems

Subsystem, Atomic Subsystem, Enabled Subsystem, Function-Call Subsystem, If Action Subsystem, Triggered Subsystem			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks > Check Subsystem blocks**
	No block-specific constraints		
Block Parameters	Subsystems must be one of the following: • Virtual • Enabled • Function-Call • If Action • Inlined Atomic • Triggered	FATAL	
	For nonvirtual subsystems, **Function packaging** (`RTWSystemCode`) must be set to `Inline`.	FATAL	

Subsystem, Atomic Subsystem, Enabled Subsystem, Function-Call Subsystem, If Action Subsystem, Triggered Subsystem			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Other	Action subsystems must not contain model reference blocks and/or conditional subsystems.	Nonfatal	**Check usage of Ports and Subsystems blocks >Check Action Subsystem blocks**
	Actions subsystems connected to the same If or Switch Case blocks must do one of the following: • All combine their output and code updates. • All separate their output and code updates.	Nonfatal	**Check destinations of If and Switchcase blocks >Check destination Action subsystem of If and Switchcase blocks**

Sum, Add, Subtract

Sum			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types and Ports	Constraints that apply to all blocks.		**Check usage of Math Operations blocks>Check Sum blocks**
	Input and output ports should have the same data type.	Nonfatal	
	Blocks must have at lease 2 inports.	FATAL	
Block Parameters	**Accumulator data type** (`AccumDataTypeStr`) must use the same data type as the block inputs.	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	

Switch

Switch			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Switch blocks**
	The first and third input ports and the output port must have the same data type.	Nonfatal	
Block Parameters	**Integer rounding mode** (`RndMeth`) must be set to `Zero` , `Floor`, or `Ceiling`.	Nonfatal	
	Allow different data input sizes (`AllowDiffInputSizes`) must not be selected (must be set to `off`).	Nonfatal	

Switch Case

Switch Case			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks >Check SwitchCase blocks**
Block Parameters	**Case conditions** (`CaseConditions`) must not have a range of values for the input.	Nonfatal	
	Block destination must not be a terminator block or an empty action subsystem.	Nonfatal	
	Source of Inport 1 must not: • Be a Constant block. • Have a constant sample time.	Nonfatal	

Terminator

Terminator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sinks blocks > Check Terminator blocks**
	No block-specific constraints		
Block Parameters	Block must not be connected to a model reference block.	Nonfatal	

Trigger

Trigger			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Ports and Subsystems blocks >Check Trigger Port blocks**
	In the parent subsystem, the signal entering the trigger port must be a scalar.	Nonfatal	
	In the parent subsystem, the signal entering the trigger port must be `boolean` when the **Trigger type** (`TriggerType`) is set to `rising`, `falling`, or `either`.	Nonfatal	
Block Parameters	**Show output port** (`ShowOutputPort`) must not be selected (must be set to `off`).	Nonfatal	
	Block must be not be in the root diagram of the model when **Trigger type** (`TriggerType`) is set to `rising`, `falling`, or `either`.	FATAL	

Trigger			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	States when enabling (`StatesWhenEnabling`) must not be set to `inherit`.	Nonfatal	
	The signal entering the Trigger Port of the parent subsystem must not: • Be from a constant block. • Have a constant sample time.	Nonfatal	

Trigonometric Function

Trigonometric Function			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Math Operations blocks > Check Trigonometry blocks**
	No block-specific constraints		
Block Parameters	**Function** (`Operator`) must *not* be set to `cos + jsin` (complex exponential of the input).	Nonfatal	
	Approximation method (`ApproximationMethod`) must be set to `None`.	Nonfatal	

Unit Delay

Unit Delay			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Discrete blocks > Check Unit Delay blocks**
	State must have storage class `Auto`. Values other than `Auto` require use of storage classes, which are not supported for code inspection.	Nonfatal	
Block Parameters	**Initial conditions** (`X0`) must not: be empty, be nonfinite, have a MATLAB structure as a value, be complex, have two or more dimensions, or specify the range (`:`) operator.	FATAL	

Vector Concatenate

Vector Concatenate			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Signal Routing blocks > Check Vector Concatenate blocks**
	Block inports and outports must be scalars or vectors.	Nonfatal	
Block Parameters	**Mode** (`Mode`) must be set to `Vector`.	Nonfatal	

Supported Blocks — By Category

In this section...

Commonly Used Blocks

- "Mux" on page 5-29
- "Outport" on page 5-30
- "Product" on page 5-31
- "Relational Operator" on page 5-33
- "Saturation" on page 5-34
- "Subsystems" on page 5-43
- "Sum, Add, Subtract" on page 5-44
- "Switch" on page 5-45
- "Terminator" on page 5-46
- "Unit Delay" on page 5-48

Discontinuity Blocks

- "Saturation" on page 5-34

Discrete Blocks

- "Unit Delay" on page 5-48
- "Discrete-Time Integrator" on page 5-16

Logic and Bit Operation Blocks

- "Logical Operator" on page 5-23
- "Relational Operator" on page 5-33
- "Shift Arithmetic" on page 5-37

Lookup Tables

- "1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)" on page 5-23

Math Operation Blocks

Model-Wide Utilities

Port & Subsystem Blocks

- "Trigger" on page 5-46

Signal Attribute Blocks

- "Data Type Conversion" on page 5-15
- "Data Type Duplicate" on page 5-15
- "Data Type Propagation" on page 5-16
- "Probe" on page 5-30
- "Signal Conversion" on page 5-38
- "Signal Specification" on page 5-38

Signal Routing Blocks

- "Bus Assignment" on page 5-10
- "Bus Creator" on page 5-10
- "Bus Selector" on page 5-11
- "Data Store Memory" on page 5-12
- "Data Store Read" on page 5-13
- "Data Store Write" on page 5-14
- "Demux" on page 5-18
- "From" on page 5-19
- "Goto" on page 5-21
- "Merge" on page 5-27
- "Multiport Switch" on page 5-28
- "Mux" on page 5-29
- "Selector" on page 5-35
- "Switch" on page 5-45
- "Vector Concatenate" on page 5-48

Sink Blocks

Source Blocks

User-Defined Functions

6

Model Advisor Checks

Simulink Code Inspector Checks

In this section...

In this section...

Simulink Code Inspector Checks Overview

Use Simulink Code Inspector Model Advisor checks to configure your model for code inspection.

See Also

- “Consult the Model Advisor”
- “Simulink Checks”
- “Embedded Coder™ Checks”
- “Simulink Verification and Validation™ Checks”

Check code generation settings

Check code generation settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that code generation settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Language' setting	The model is configured to generate `C++ (Encapsulated)` files.	In the Configuration Parameters dialog box, on the **Code Generation** pane, set **Language** to `C` or `C++`.
Verify 'Shared code placement' setting	The model is not configured to generated shared utility code to a shared location. If shared utility code is generated into *`model`*`.c`, the Code Inspector reports the code as unverified.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set **Shared code placement** to `Shared location`. Using a shared location for utility functions, macros, and user-defined data types promotes debugging and traceability of generated code.
Verify 'Source file' setting	Custom code is configured to appear near the top of the generated model source file.	In the Configuration Parameters dialog box, on the **Code Generation > Custom Code** pane, clear the **Source file** field.
Verify 'Header file' setting	Custom code is configured to appear near the top of the generated model header file.	In the Configuration Parameters dialog box, on the **Code Generation > Custom Code** pane, clear the **Header file** field.
Verify 'Initialize function' setting	Custom code is configured to appear in the generated model initialize function.	In the Configuration Parameters dialog box, on the **Code Generation > Custom Code** pane, clear the **Initialize function** field.

Subcheck	Condition	Recommended Action
Verify 'Terminate function' setting	Custom code is configured to appear in the generated model terminate function.	In the Configuration Parameters dialog box, on the **Code Generation > Custom Code** and clear the **Terminate function** field.
Verify 'Combine signal/state structures' setting	The model is configured to combine global block signals and global state data into one data structure in the generated code. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **Combine signal/state structures** parameter.
Verify 'Include comments' setting	The model is configured to omit autogenerated comments from generated code files. The Code Inspector parses autogenerated comments to obtain traceability information about model data.	In the Configuration Parameters dialog box, on the **Code Generation > Comments** pane, select **Include comments**.
Verify 'Generate scalar inlined parameter as' setting	The model is configured to generate scalar inlined parameters as variables with `#define` macros, rather than as numeric constants.	In the Configuration Parameters dialog box, on the **Code Generation > Symbols** pane, set **Generate scalar inlined parameter as** to `Literals`.
Verify 'Preserve condition expression in if statement' setting	The model is configured to optimize empty primary condition expressions in `if` statements by negating them, rather than preserving the empty primary condition expressions.	In the Configuration Parameters dialog box, on the **Code Generation > Code Style** pane, select **Preserve condition expression in if statement**.
Verify 'Replace data type names in the generated code' setting	The model is configured to replace built-in data type names with user-defined data type names in the generated code. Data type replacement is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Data Type Replacement** pane, clear the **Replace data type names in the generated code** parameter.

Subcheck	Condition	Recommended Action
Verify 'Code replacement library' setting	A code replacement library other than `C89/C90 (ANSI)`, `ANSI_C`, `C99 (ISO)`, or `ISO_C` is selected for the model.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set **Code replacement library** to `C89/C90 (ANSI)`, `ANSI_C`, `C99 (ISO)`, or `ISO_C`. The check fails if you do not select `C89/C90 (ANSI)`, `ANSI_C`, `C99 (ISO)`, or `ISO_C`. However, if you create your library using "Supported Functions and Operations in Code Replacement Libraries" on page 4-22, Simulink Code Inspector does inspect the generated code.
Verify 'Classic call interface' setting	The model is configured to generate model function calls compatible with the main program module of a pre-R2011a GRT target. The classic call interface is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **Classic call interface** parameter.
Verify 'Single output/update function' setting	The model is configured to generate code in separate *`model`*`_output` and *`model`*`_update` functions, rather than a *`model`*`_step` function that combines the two.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, select **Single output/update function**.
Verify 'Terminate function required' setting	The model is configured to generate a *`model`*`_terminate` function, potentially containing model termination code to be executed during system shutdown. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **Terminate function required** parameter.
Verify 'Generate reusable code' setting	The model is not configured to generate reusable, multi-instance code that is reentrant. This parameter is applicable only to the top model in a model hierarchy.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, select **Generate reusable code**.

Subcheck	Condition	Recommended Action
Verify 'MAT-file logging' setting	The model is configured to log execution data to a MAT-file. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **MAT-file logging** parameter.
Verify 'non-finite numbers' setting	The model is configured to generate nonfinite data (for example, `NaN` and `Inf`) and related operations. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **Support: non-finite numbers** parameter.
Verify 'absolute time' setting	The model is configured to generate and maintain integer counters for absolute and elapsed time values. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, clear the **Support: absolute time** parameter.
Verify 'Suppress error status in real-time model data structure' setting	The model is configured to include an error status field in a generated `rtModel` data structure. The `rtModel` data structure is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, select **Suppress error status in real-time model data structure**.
Verify 'IncludeERT-FirstTime' setting	The model is configured to include the *`firstTime`* argument in the generated *`model`*`_initialize` function. This is not supported for code inspection.	In the MATLAB Command Window, set the model parameter `IncludeERTFirstTime` to `off`. For example, `set_param(gcs, 'IncludeERTFirstTime', 'off')`.
Verify 'Pass root-level I/O as' setting	The model is configured to use packed structures, rather than individual arguments, to pass root-level model input and output values to the *`model`*`_step` function. This is not supported for code inspection. This parameter is applicable only to the top model in a model hierarchy.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set **Pass root-level I/O as** to `Individual arguments`.

Subcheck	Condition	Recommended Action
Verify 'Create block' setting	The model is configured to generate a SIL or PIL block during code generation. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Verification** pane, set **Create block** to `None`.
Verify 'Measure function execution times' setting	The model is configured to generate code with instrumentation to collect execution times for functions inside the generated code. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Verification** pane, clear the **Measure function execution times** parameter.
Verify 'Signal naming' setting	The model is configured to change signal names when creating corresponding identifiers in the generated code.	In the Configuration Parameters dialog box, on the **Code Generation > Symbols** pane, set **Signal naming** to `None`.
Verify 'Parameter naming' setting	The model is configured to change parameter names when creating corresponding identifiers in the generated code.	In the Configuration Parameters dialog box, on the **Code Generation > Symbols** pane, set **Parameter naming** to `None`.
Verify 'TLC options' setting	The model is configured with TLC options.	In the Configuration Parameters dialog box, on the **Code Generation** pane, clear the **TLC options** field.
Verify Code Generation > Interface > Interface setting	The model is configured to generate code for C API, external mode, or ASAP2 data interfaces. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set **Interface** to `None`.

See Also

Simulink Configuration Parameter Constraints

Check data import/export settings

Check data import/export settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that data import/export settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Initial state' setting	The model is configured to load initial states from a workspace, which is not compatible with code inspection.	In the Configuration Parameters dialog box, on the **Data Import/Export** pane, clear the **Initial state** parameter.

See Also

Simulink Configuration Parameter Constraints

Check diagnostic settings

Check diagnostic settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that diagnostic settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Invalid root Inport/Outport block connection' setting	The model is not configured to generate an error if Simulink software detects invalid internal connections to the root-level Inport or Outport blocks. This potentially allows automatic insertion of hidden signal copy blocks at the model inports and outports, which is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Diagnostics > Model Referencing** pane, set **Invalid root Inport/Outport block connection** to `error`. If an error is generated, it identifies the locations at which you can manually insert Signal Conversion blocks to avoid the error and maintain traceability.
Verify 'Underspecified initialization detection' setting	The model is not configured to initialize block initial conditions using simplified behavior. The simplified behavior can improve the consistency of model results.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Underspecified initialization detection** to `Simplified`.
Verify 'Non-bus signals treated as bus signals' setting	The model is not configured to generate an error when Simulink software implicitly converts a non-bus signal to a bus signal to support connecting the signal to a Bus Assignment or Bus Selector block.	In the Configuration Parameters dialog box, on the **Diagnostics > Connectivity** pane, set **Non-bus signals treated as bus signals** to `error`.
Verify 'Detect downcast' setting	The model is not configured to generate an error when a parameter downcast occurs during simulation.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Detect downcast** to `error`.

Subcheck	Condition	Recommended Action
Verify 'Detect overflow' setting	The model is not configured to generate an error when a parameter overflow occurs during simulation.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Detect overflow** to `error`.
Verify 'Detect underflow' setting	The model is not configured to generate an error when a parameter underflow occurs during simulation.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Detect underflow** to `error`.
Verify 'Detect precision loss' setting	The model is not configured to generate an error when parameter precision loss occurs during simulation.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Detect precision loss** to `error`.
Verify 'Detect loss of tunability' setting	The model is not configured to generate an error when an expression with tunable variables is reduced to its numerical equivalent.	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set **Detect loss of tunability** to `error`.
Verify Bus signal treated as vector setting	The model is not configured to generate an error when Simulink software detects a virtual bus signal that is used as a mux signal. Strict bus behavior is not enforced.	In the Configuration Parameters dialog box, on the **Diagnostics > Connectivity** pane, set **Bus signal treated as vector** to `error`.

See Also

Simulink Configuration Parameter Constraints

Check hardware implementation settings

Check hardware implementation settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that hardware implementation settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'char' setting	The bit length of character data for the production hardware does not equal `8`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, select a production hardware **Device type** that is compatible with the settings in this table.
Verify 'short' setting	The bit length of `short` data for the production hardware does not equal `16`.	
Verify 'int' setting	The bit length of `int` data for the production hardware does not equal `32`.	
Verify 'long' setting	The bit length of `long` data for the production hardware does not equal `32`.	
Verify 'float' setting	The bit length of floating-point data for the production hardware does not equal `32`.	
Verify 'double' setting	The bit length of `double` data for the production hardware does not equal `64`.	
Verify 'pointer' setting	The bit length of pointer data for the production hardware does not equal `32`.	

Subcheck	Condition	Recommended Action
Verify 'native' setting	The microprocessor native word size for the production hardware does not equal `32` bits.	
Verify 'Signed integer division rounds to' setting	The method of producing a signed integer quotient for the production hardware is not to choose the integer that is closer to zero (`Zero` method).	
Verify 'Shift right on a signed integer as arithmetic shift' setting	The method by which the compiler implements signed integer right shift for the production hardware is not an arithmetic right shift.	
Verify 'None' setting	The test hardware differs from the deployment hardware.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, under **Emulation hardware (code generation only)**, select **None**.
Verify 'Device vendor->Device type' setting	The device vendor and device type are `ASIC/FPGA`.	In the Configuration Parameters dialog box, on the **Hardware Implementation** pane, under **Embedded hardware (simulation and code generation)**, do not select **Device vendor** `ASIC/FPGA`.

See Also

Simulink Configuration Parameter Constraints

Check optimization settings

Check optimization settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that optimization settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'AdvancedOptControl' setting	The model is not configured to disable optimizations that are incompatible with Simulink Code Inspector.	In the MATLAB Command Window, set the model parameter `AdvancedOptControl` to `-SLCI`. For example, `set_param(gcs, 'AdvancedOptControl', '-SLCI')`.
Verify 'Implement logic signals as Boolean data (vs. double)' setting	The model is configured to implement logic signals with the `double` data type, rather than with the more memory-efficient `boolean` data type.	In the Configuration Parameters dialog box, on the **Optimization** pane, select **Implement logic signals as Boolean data (vs. double)**.
Verify 'Optimize initialization code for model reference' setting	The model is configured to generate initialization code for blocks that have states, without an optimization that can produce more efficient code for referenced models.	In the Configuration Parameters dialog box, on the **Optimization** pane, select **Optimize initialization code for model reference**.
Verify 'Inline invariant signals' setting	The model is configured to use symbolic names (instead of inline numerical values) for invariant signals in generated code.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, select **Inline invariant signals**.

Subcheck	Condition	Recommended Action
Verify 'Remove code from floating-point to integer conversions that wraps out-of-range values' setting	The model is configured not to remove wrapping code that handles out-of-range floating-point to integer conversion results when out-of-range conversions occur.	In the Configuration Parameters dialog box, on the **Optimization** pane, select **Remove code from floating-point to integer conversions that wraps out-of-range values**.
Verify 'Remove code from floating-point to integer conversions with saturation that maps NaN to zero' setting	The model is configured to remove code that handles floating-point to integer conversion results for `NaN` values when mapping from `NaN` to integer zero occurs.	In the Configuration Parameters dialog box, on the **Optimization** pane, clear the **Remove code from floating-point to integer conversions with saturation that maps NaN to zero** parameter.
Verify 'Remove code that protects against division arithmetic exceptions' setting	The model is configured to remove code that guards against division by zero for fixed-point data.	In the Configuration Parameters dialog box, on the **Optimization** pane, clear the **Remove code that protects against division arithmetic exceptions** parameter.
Verify 'Maximum stack size (bytes)' setting	The model is configured with a maximum stack size.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, set the **Maximum stack size (bytes)** to `inf`.

Subcheck	Condition	Recommended Action
Verify 'Pack Boolean data into bitfields' setting	The model is configured to store Boolean signals as one-bit bitfields.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, clear the **Pack Boolean data into bitfields** parameter.
Verify 'Simplify array indexing' setting	The model is configured to generate code that replaces multiply operations with add operations in array indices when accessing arrays in a loop.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, clear the **Simplify array indexing** parameter.

See Also

Simulink Configuration Parameter Constraints

Check solver settings

Check solver settings in the model configuration that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that solver settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Type' setting	The model is configured with a variable-step solver.	In the Configuration Parameters dialog box, on the **Solver** pane, set **Type** to `Fixed-step`.
Verify 'Solver' setting	The model is configured with a solver other than a fixed-step discrete solver.	In the Configuration Parameters dialog box, on the **Solver** pane, set **Solver** to `discrete (no continuous states)` (equivalent to `FixedStepDiscrete` specified at the command line).

See Also

Simulink Configuration Parameter Constraints

Check for unconnected objects in the model

Check for unconnected ports and lines in the model.

Description

This check reports unconnected lines, input ports, and output ports in the model or subsystem.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check for unconnected objects	One or more lines, input ports, or output ports are not properly connected in the model or subsystem. This can result in dead code or hidden ground blocks.	Connect or remove the affected blocks.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check system target file setting

Check whether a compatible system target file is selected for the model.

Description

This check verifies that the **System target file** selected for the model is `ert.tlc` or is derived from `ert.tlc`.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify system target file setting	The system target file selected for the model is not `ert.tlc` or an ERT-derived target.	In the Configuration Parameters dialog box, on the **Code Generation** pane, set **System target file** to `ert.tlc` or an ERT-derived target.

See Also

Simulink Configuration Parameter Constraints

Check function specification setting

Check for function specification settings that might impact compatibility with Simulink Code Inspector.

Description

This check verifies that function prototype control settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check model interface settings	The model specifies custom function prototypes for model entry functions. This is not supported for code inspection.	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, click **Configure Model Functions** to open the Model Interface dialog box, and set **Function specification** to `Default model initialize and step functions`.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for Stateflow machine data

Check the model for Stateflow data of machine scope. Data of machine scope is incompatible with Simulink Code Inspector

Description

This check verifies that the model does not contain Stateflow data of machine scope.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
All Stateflow data must be parented by a Stateflow chart	The model contains Stateflow data of machine scope.	Modify model so that it does not contain Stateflow data of machine scope.

See Also

"Data Specification"

Check for Stateflow machine events

Check the model for Stateflow events of machine scope. Events of machine scope are incompatible with Simulink Code Inspector

Description

This check verifies that the model does not contain Stateflow events of machine scope.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
All Stateflow events must be parented by a Stateflow chart	The model contains Stateflow events of machine scope.	Modify model so that it does not contain Stateflow events of machine scope.

See Also

"Input and Output Events"

Check conditional input branch execution setting

If the model is using conditional input branch execution, check that local block outputs are enabled.

Description

This check verifies that the model configuration parameter **Enable local block outputs** is selected when **Conditional input branch execution** is selected.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify conditional input branch execution setting	The model configuration parameter **Conditional input branch execution** is selected, but **Enable local block outputs** is not selected. The model must enable local block outputs when using conditional input branch execution.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, select **Enable local block outputs**.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for unsupported blocks

Check for blocks that are not supported by Simulink Code Inspector.

Description

This check updates the model diagram and reports blocks that are not supported by Simulink Code Inspector.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check for blocks not supported by Simulink Code Inspector	One or more blocks in the model are not supported for code inspection. **Note** Supported blocks are listed in "Supported Blocks — By Category" on page 5-49, and also can be viewed in the `slcilib` block library.	Possible actions include: • Replace an unsupported block with a supported block. • Replace an unsupported block with an equivalent combination of supported blocks. • Replace an unsupported block with an S-Function block created using the Legacy Code Tool. • If one or more unsupported blocks cannot be removed, use referenced models to isolate the unsupported block(s), and/or use a partial verification work flow that omits the unsupported block(s).

See Also

- "Fix or Work Around Unsupported Blocks"
- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check storage class for workspace variables

Check for workspace variables referenced by the model.

Description

This check reports workspace variables that use unsupported storage classes.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check storage class for workspace variables referenced by the model	Workspace variables referenced by the model are not supported for one or both of these reasons: • The "Custom Storage Classes" **Type** is not set to `Unstructured`. • Workspace variable is tunable, with **data type** set to `struct`.	Modify the model so that the model does not reference workspace variables or set the workspace variable **Type** to `Unstructured`.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for sample times in the model

Check for sample time characteristics that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports instances of multiple, variable, continuous, or asynchronous sample times.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check sample times	The model is using multiple, variable, continuous, or asynchronous sample times. This is not supported for code inspection.	Modify the model such that multiple, variable, continuous, or asynchronous sample times are not being used.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for Signal Conversion blocks automatically inserted on signals entering block input ports

Check for hidden Signal Conversion blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports hidden Signal Conversion blocks that have been automatically inserted on signals entering block input ports.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify no Signal Conversion blocks are automatically inserted on signals entering block inports	A hidden Signal Conversion block has been automatically inserted on a signal entering a block inport. Hidden Signal Conversion blocks are not supported for code inspection.	Manually insert a Signal Conversion block on the signal entering the block inport, and configure the Signal Conversion block to be excluded from block reduction.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for usage of fixed-point instrumentation

Check for usage of fixed-point instrumentation that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports fixed-point instrumentation incompatibilities.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify usage of fixed-point instrumentation	The model configuration parameter **Block reduction** (`BlockReduction`) is selected, and the fixed point parameter **Fixed-point instrumentation mode** (`MinMaxOverflowLogging`) is set to a value other than `Force off`. Simultaneous use of block reduction and fixed-point instrumentation is not supported for code inspection.	Open the Fixed-Point Tool and turn off fixed-point instrumentation for the model.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for root Outport blocks being conditionally assigned

Check that root outports of submodels are not connected to conditionally executed subsystems.

Description

This check updates the model diagram and verifies that root outports of referenced models are not connected to conditionally executed subsystems.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify that root Outports are not assigned conditionally	A root outport of a referenced model is directly connected to a conditionally executed subsystem and the root outport storage class is set to `Auto`. Code inspection is not supported for submodels for which root outports are assigned by blocks inside conditionally executed subsystems.	This check only applies to referenced models. You can do one of the following: • If this model will be the top model in the hierarchy, in the Configuration Parameters dialog box, on the **Model Referencing** pane, set **Total number of instances allowed per top model** to `Zero`, which will suppress the check. • Modify the model so that the root outports are not directly connected to conditionally executed subsystems. • Use a root outport with a storage class that is not set to `Auto`.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for usage of synthesized local data stores

Check for signal objects in the model workspace that are referenced as synthesized local data stores by Data Store Read or Data Store Write blocks.

Description

This check updates the model diagram and verifies synthesized local data store usage. If your model has signal objects that are referenced as synthesized local data stores by Data Store Read or Data Store Write blocks, Simulink creates a hidden Data Store Memory block at the root level of the model. This model is incompatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify synthesized local data store usage	Signal objects are referenced as synthesized local data stores by Data Store Read or Data Store Write blocks.	Avoid using signal objects that are referenced as synthesized local data stores by Data Store Read or Data Store Write block. As a possible work around, create graphical Data Store Memory blocks to specify the data stores.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check loop unrolling threshold setting

Checks that the model does not have a loop unrolling threshold that might result in partially unrolled loops in the generated code.

Description

This check updates the model diagram and verifies that the model does not have a loop unrolling threshold that might result in partially unrolled loops in the generated code.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify loop unrolling threshold setting	The model is configured with a **Loop unrolling threshold** that might result in partially unrolled loops in the generated code.	In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane, set the **Loop unrolling threshold** to the value in the **Recommended Action** section of the Model Advisor window.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check usage of global data stores

Checks that global Data Store Memory blocks use inlined parameters with non-tunable initial values.

Description

This check updates the model diagram and verifies global data store usage. If your model has Data Store blocks with parameters that are not inlined or have tunable initial values, it is incompatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify global data store usage	Configuration parameter **Optimization > Signals and Parameters >Inline parameters** (`InlineParams`) is not selected.	Select **Optimization > Signals and Parameters >Inline parameters**.
	Initial value (`InitialValue`) must not be tunable.	Fix the **Initial value** setting.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check destinations of If and Switchcase blocks

Check usage of If and Switch Case blocks connected to Action subsystems that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and verifies the usage of If and Switch Case blocks connected to Action subsystems.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check destination Action subsystem of If and Switchcase blocks	Action subsystems connected to the same If or Switch Case blocks do not do one of the following: • All combine their output and code updates. • All separate their output and code updates.	Modify the listed Action subsystems so that they all combine their output and code updates. Place a Signal Conversion block on signals leaving the inports within the Action subsystems. Select the Signal Conversion block parameter **Exclude this block from 'Block reduction' optimization** to exclude it from block reduction.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check for root Outport blocks that have non-auto storage class

Check usage of root outport blocks in referenced model that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and verifies the usage of root outport blocks in referenced models.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify that the storage class of root outports is supported	**Pass reusable subsystem outputs as:** is not set to `Structure reference` when root outports in referenced models have non-auto storage class.	Set **Pass reusable subsystem outputs as:** to `Structure reference`.

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check usage of Sources blocks

Check for usage of Sources blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Sources blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Inport blocks **Note** This will check shadowed inports if you have any in your model.	The block cannot specify variable-dimension signals. Block parameter **Variable-size signal** (`VarSizeSig`) is set to `Yes`.	Set **Variable-size signal** to `No`.
	Block parameter **Signal Type** (`SignalType`) is set to `complex`.	Set **Signal Type** to `real` or `auto`.
	Block parameter **Sampling Mode** (`SamplingMode`) is set to `Frame based`.	Set **Signal Type** to `Sample based` or `auto`.
	For inports in triggered subsystems, **Latch input be delaying outside signal** (`LatchByDelayingOutsideSignal`) is selected (set to `on`).	Clear **Latch input be delaying outside signal**. To retain the latching behavior, restructure the model by placing a Unit Delay block before the input block in the parent diagram.

	Violates a constraint that applies to all blocks:
	
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Constant blocks	Block parameter **Constant value** (`Value`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Constant value** setting.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Ground blocks	Violates a constraint that applies to all blocks: • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Signal Routing blocks

Check for usage of Signal Routing blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Signal Routing blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action

Check Bus Creator blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Bus Selector blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Check Bus Assignment blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Data Store Memory blocks	The block state does not have storage class `Auto`. Values other than `Auto` require use of storage classes, which are not supported for code inspection.	Modify the block such that its code generation storage class is set to `Auto`. If the block state name does not resolve to a signal object, set **Storage Class** in the **State Attributes** tab of the block parameter dialog box to `Auto`. If the block state name does resolve to a signal object, set the `CoderInfo.StorageClass` property of the signal object to `Auto`.
	Block parameter **Initial value** (`InitialValue`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Initial value** setting.
	Block parameter **Signal type** (`SignalType`) is set to `complex`. Complex values are not supported for code inspection.	Set **Signal type** to `auto` or `real`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`,	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	`int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Data Store Read blocks	The block cannot specify elements. Block parameter **Specify element(s) to select** (`DataStoreElements`) is set to a nonempty string.	Clear element selections from the **Element Selection** tab of the block dialog box.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Data Store Write blocks	The block cannot specify elements. Block parameter **Specify element(s) to select** (`DataStoreElements`) is set to a nonempty string.	Clear element selections from the **Element Selection** tab of the block dialog box.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Check From blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Goto blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Merge blocks	**Initial output** is set to an unsupported value.	Set **Initial output** to `0`.
	Allow unequal port widths (`AllowUnequalInputPortWidths`) is selected.	Clear the **Allow unequal port widths** parameter.
	Input port offsets is set to an unsupported value.	Set **Input port offsets** to `[]`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Switch blocks	The first and third input ports and the output port do not have the same data type.	Modify the data ports to have the same data type. Consider selecting the block parameter **Require all data port inputs to have the same data type**.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Block parameter **Allow different data input sizes** (`AllowDiffInputSizes`) is selected.	Clear the **Allow different data input sizes** parameter.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`,	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	`int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Multiport Switch blocks	Data input and output ports do not have the same data type.	Modify the data ports to have the same data type. Consider selecting the block parameter **Require alldata port inputs to have the same data type**.
	Multiport Switch blocks must have at least three inports.	Reconfigure the block to have at least three inports.
	Data port indices are specified and an input has more than one value.	Modify the data port configuration so that only one value is specified per input.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Block parameter **Allow different data input sizes** (`AllowDiffInputSizes`) is selected.	Clear the **Allow different data input sizes** parameter.
	Block parameter **Data port for default case** (`DataPortForDefault`) is not set to `Last data port`.	Set **Data port for default case** to `Last data port`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Mux blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Demux blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Selector blocks	Uses multidimensional input, or uses port-based indexing instead of specifying indices using the block dialog.	Configure the block to use one-dimensional inputs, and specify indices using the block dialog. Set block parameter **Index Option** to `Select all`, `Index vector (dialog)`, or `Starting index (dialog)`.
	Block inport or outport is not a scalar or vector.	Configure the listed block to use scalar or vector inports and outports.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`,	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	`int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Vector Concatenate blocks	Block parameter **Mode** (`Mode`) is not set to `Vector`.	Set **Mode** to `Vector`.
	Block inports and outports are not scalars or vectors.	Configure the inports and outports to be scalars or vectors.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Math Operations blocks

Check for usage of Math Operations blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Math Operations blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Absolute blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Gain blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Block parameter **Gain** (`Gain`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Gain** setting.
	Block parameter **Parameter data type** (`ParamDataTypeStr`) does not use the same data type as the Gain block input.	Modify the Gain block to use the same data type for its input and parameter. Consider setting **Parameter data type** to `Inherit: Same as input`.
	Block parameter **Multiplication** (`Multiplication`) is not set to `Element-wise(K.*u)`, `Matrix(K*u)`, `Matrix(u*K)`, or `Matrix(K*u)(u vector)`.	Set **Multiplication** to `Element-wise(K.*u)`, `Matrix(K*u)`, `Matrix(u*K)`, or `Matrix(K*u)(u vector)`. Only `single` or `double` data types are supported for `Matrix(K*u)`, `Matrix(u*K)`, or `Matrix(K*u)(u vector)` multiplications methods.

Subcheck	Condition	Recommended Action
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
	• Block has constant (`Inf`) sample time and an outport has been testpointed.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Math blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Function (`Operator`) is set to an unsupported value: `conj` or `hermitian`.	Set **Function** to one of the following values: `exp`, `log`, `10^u`, `log10`, `magnitude^2`, `square`, `transposepow`, `reciprocal`, `hypot`, `rem`, or `mod`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Product blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Block parameter **Multiplication** (`Multiplication`) is not set to `Element-wise(.*)` or `Matrix (*)`.	Set **Multiplication** to `Element-wise(.*)` or `Matrix (*)`. Only `single` or `double` data types are supported for `Matrix (*)` multiplication.
	Block parameter **Number of inputs** (`inputs`) is not set to `2`, `**`, `/*`, `*/`, `//`, or `/` when both of the following are true: • Inport **Signal type** is a matrix. • Product block parameter **Multiplication** is set to `Matrix (*)`.	Set **Number of inputs** to `2`, `**`, `/*`, `*/`, `//`, or `/` if both of the following are true: • Inport **Signal type** is a matrix. • Product block parameter **Multiplication** is set to `Matrix (*)`.
	Block parameter **Number of inputs** (`inputs`) is not set to `2`, `**`, `/*`, `*/`, or `//` when both of the following are true:	Set **Number of inputs** to `2`, `**`, `/*`, `*/`, or `//` if both of the following are true:

Subcheck	Condition	Recommended Action
	• Inport **Signal type** is a scalar or vector. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.	• Inport **Signal type** is a scalar or vector. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.
	Block parameter **Number of inputs** (`inputs`) is not set to `/` when both of the following are true: • Inport **Signal type** is a scalar. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.	Set **Number of inputs** to `/` if both of the following are true: • Inport **Signal type** is a scalar. • Product block parameter **Multiplication** is set to `Element-wise(.*)`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Sum blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Block parameter **Accumulator data type** (`AccumDataTypeStr`) does not use the same data type as the block inputs.	Modify the block to use the same data type for its inputs and accumulator. Consider setting **Accumulator data type** to `Inherit: Same as first input`.
	Block does not have at least 2 inports.	Configure the model so that there are 2 inports to the block.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`,	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	`int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Trigonometry blocks	Block parameter **Function** (`Operator`) is set to `cos + jsin` (complex exponential of the input).	Set **Function** to a value other than `cos + jsin`.
	Block parameter **Approximation method** (`ApproximationMethod`) is not set to `None`.	Set **Approximation method** to `None`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check MinMax blocks	An unsupported data type is specified for an input or output port.	Modify the port data type to be one of the following: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.
	Input and output ports do not have the same data type.	Modify the port data types to match.
	MinMax blocks must have at least two inports.	Reconfigure the block to have at least two inports.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Check Rounding Function blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double` or `single`, .If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Reshape blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Check Sign blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Sqrt blocks	Block inputs and outports do not have the same data type.	Fix the listed block inport or outport.
	Block parameter **Function** (`Operator`) is not set to `sqrt` or `signedSqrt`.	Set block parameter **Function** to `sqrt` or `signedSqrt`.
	Block parameter **Output signal type** (`OutputSignalType`) is set to `complex`.	Set block parameter **Output signal type** (`OutputSignalType`) to `auto` or `real`.
	Block inputs and outports data types are not `single` or `double`.	Fix the listed block inport or outport.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- “Block Constraints — Alphabetical List” on page 5-5
- “Supported Blocks — By Category” on page 5-49

Check usage of Signal Attributes blocks

Check for usage of Signal Attributes blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Signal Attributes blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Data Type Conversion blocks	Block parameter **Input and output to have equal** (`ConvertRealWorld`) is not set to `Real World Value (RWV)`.	Set **Input and output to have equal** to `Real World Value (RWV)`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Check Data Type Duplicate blocks	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Signal Conversion blocks	Block parameter **Output** (`ConversionOutput`) is not set to `Signal copy`.	Set **Output** to `Signal copy`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Probe blocks	Block parameter **Data type for width** (`ProbeWidthDataType`) is set to `boolean`.	Set block parameter **Data type for width** to any data type except `boolean`.
	Block parameter **Data type for signal dimensions** (`ProbeDimensionsDataType`) is set to `boolean`.	Set block parameter **Data type for signal dimensions** to any data type except `boolean`.
	Block parameter **Data type for sample time** (`ProbeSampleTimeDataType`) is not set to `single` or `double`.	Set block parameter **Data type for sample time** to `single` or `double`.

Subcheck	Condition	Recommended Action

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
	• Block has constant (`Inf`) sample time and an outport has been testpointed.
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.
		Fix the listed block inport or outport.

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Logical and Bit Operations blocks

Check for usage of Logical and Bit Operations blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Logical and Bit Operations blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Relational Operator blocks	The data type of a block outport is not either an enumerated type with default value 0, or `boolean`.	Modify the outport data type to be either an enumerated type with default value 0, or `boolean`.
	Block input ports do not have the same data type.	Modify the input ports to have the same data type.
	Block parameter **Relational operator** (`Operator`) is set to an unsupported value: `isInf`, `isNaN`, or `isFinite`.	Set **Relational operator** to a supported value: `<=`, `==`, `>=`, `~=`, `<`, or `>`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Logic blocks	Logical Operator block outport is not `boolean` or `uint8`.	Modify the data type of the outport to `boolean` or `uint8`.
	Logical Operator blocks must have at least two inports, except in the case of the NOT operator.	Reconfigure the block to have at least two inports.
	Block input ports do not have the same data type.	Configure the input ports to have the same data type.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	`Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Bitwise Operator blocks	With **Number of input ports**(`NumInputPorts`) set to `1` and **Operator**(`logicop`) set to `AND`, `OR`, `NAND`, `NOR`, or `XOR`, the inport data type is not a scalar.	Configure the inport data type to be a scalar.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
	• Block has constant (`Inf`) sample time and an outport has been testpointed.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check ArithShift blocks	**Diagnostic for out of range shift value** (`DiagnosticForOORShift`) is not set to `Error`.	Set **Diagnostic for out of range shift value** to `Error`.
	Binary points to shift (`BinPtShiftNumber`) is not set to `0`.	Set **Bits points to shift** to `0`.
	Bits to shift: Number (`BitShiftNumber`) is not within the allowable range of the inport data type.	Enter a **Bits to shift: Number** that is within the allowable range of the inport data type.
	Bits to shift: Source (`BitShiftNumberSource`) is set to `Input port` and **Bits to shift: Direction** (`BitShiftDirection`) is set to `Bidirectional` when the source of Inport 2 either: • Is a Constant block. • Has a constant sample time.	Modify the model so that the source of Input 2 is not a Constant block or have a constant sample time.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Lookup Tables blocks

Check for usage of Lookup Table blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Lookup Table blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Lookup Table blocks	Input and output ports do not have the same data type.	Modify the input and output ports to have the same data type.
	Input or output port is not a scalar.	Configure the listed input and output ports to be scalars.
	Block parameter **Number of table dimensions** (`NumberOfTableDimensions`) is not set to 1 or 2.	Set **Number of table dimensions** to 1 or 2.
	Block parameter **Interpolation method** (`InterpMethod`) is not set to `Linear`.	Set **Interpolation method** to `Linear`.
	Block parameter **Extrapolation method** (`ExtrapMethod`) is not set to `Clip` or `Linear`.	Set **Extrapolation method** to `Clip` or `Linear`.
	Block parameter **Index search method** (`IndexSearchMethod`) is not set to `Binary search`.	Set **Index search method** to `Binary search`.
	Block parameter **Begin index search using previous index result** (`BeginIndexSearchUsingPreviousIndexResult`) is selected (set to `on`).	Clear the **Begin index search using previous index result** parameter.

Subcheck	Condition	Recommended Action
	Block parameter **Support tunable table size in code generation** (`SupportTunableTableSize`) is selected (set to `on`).	Clear the **Support tunable table size in code generation** parameter.
	Block parameter **Remove protection against out-of-range input in generated code** (`RemoveProtectionInput`) is cleared (set to `off`).	Select the **Remove protection against out-of-range input in generated code** parameter.
	Block parameter **Saturate on integer overflow** (`SaturateOnIntegerOverflow`) is selected (set to `on`).	Clear the **Saturate on integer overflow** parameter.
	Block parameter **Fraction > Data Type** (`FractionDataTypeStr`) is not set to `double` or `single`.	Set **Fraction > Data Type** to `double` or `single`.
	Block parameter **Table data** (`Table`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Table data** setting.
	Block parameter **Breakpoints 1** (`sForDimension1`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Breakpoints 1** setting.
	Block parameter **Breakpoints 2** (`BreakpointsForDimension2`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **s 2** setting.

Subcheck	Condition	Recommended Action
	Block parameter **s 1** (`sForDimension1DataTypeStr`) is not using the same data type as the block input.	Modify the data types to match.
	Block parameter **s 2** (`sForDimension2DataTypeStr`) is not using the same data type as the block input.	Modify the data types to match.
	Block parameter **Table data** (`TableDataTypeStr`) is not using the same data type as the block output.	Modify the data types to match.
	Block parameter **Intermediate Results** (`IntermediateResultsDataTypeStr`) is not using the same data type as the block output.	Modify the data types to match.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. IIf the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- “Block Constraints — Alphabetical List” on page 5-5
- “Supported Blocks — By Category” on page 5-49

Check usage of User-Defined Function blocks

Check for usage of User-Defined Function blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in User-Defined Function blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check S-Function blocks	The S-function was not created using the Legacy Code Tool.	If possible, create the S-function using the Legacy Code Tool, or explore alternatives for including the code in the model.
	An S-function argument is neither a scalar nor a vector of fixed dimension.	Modify the S-function such that arguments are scalars or vectors of fixed dimension.
	The Legacy Code Tool S-function specifies a `InitializeConditionsFcnSpec`, `StartFcnSpec`, or `TerminateFcnSpec`, rather than an `OutputFcnSpec`.	Modify the S-function configuration to specify an `OutputFcnSpec`.
	The S-function has more than one `dwork`.	Modify the S-function configuration to specify one `dwork`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Ports and Subsystems blocks

Check for usage of Ports and Subsystems blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Ports and Subsystems blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Enable Port blocks	The signal entering the enable port is not of data type `boolean`.	Fix the signal data type.
	Block parameter **Show output port** (`ShowOutputPort`) is selected.	Clear the parameter **Show output port**.
	The Enable Port block is located at the root level of the model.	Remove or relocate the Enable Port block.
	The signal entering the Enable Port of the parent subsystem: • Is from a Constant block. • Has a constant sample time.	Modify the model so that the signal entering the Enable Port of the parent subsystem: • Is not from a Constant block. • Does not have a constant sample time.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Model Reference blocks	The Model block cannot have variants. Block parameter **Enable variants** (`Variant`) is selected (set to `on`).	Clear the **Enable variants** parameter.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Subsystem blocks	The subsystem is not one of the following: • Virtual • Enabled • Function-Call • If Action • Inlined Atomic • Triggered	If possible, reconfigure the subsystem to be either virtual (clear the Subsystem block parameter **Treat as atomic unit**), or an inlined atomic, enabled, function-call, if action, or triggered subsystem. Alternatively, wrap the subsystem in a Model block, or explore other implementation options.
	For nonvirtual subsystems, **Function packaging** (`RTWSystemCode`) is not set to `Inline`.	Set **Function packaging** to `Inline`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Action Subsystem blocks	Action subsystem contains model reference blocks and/or conditional subsystems.	Reconfigure the subsystem so that it does not contain model reference blocks and/or a conditional subsystems.
Check Trigger Port blocks	In the parent subsystem, the signal entering the trigger port is not a scalar.	Configure the signal entering the trigger port of the parent subsystem to be scalar.
	In the parent subsystem, the signal entering the trigger port is not a `boolean` data type when **Trigger type** (`TriggerType`) is `rising`, `falling`, or `either`.	Configure the signal entering the trigger port of the parent subsystem to be `boolean`.
	Show output port (`ShowOutputPort`) is selected.	Clear **Show output port**.
	Block is at the root diagram of the model with **Trigger type** (`TriggerType`) set to `rising`, `falling`, or `either`.	Do one of the following: • Configure the model so that the trigger block is not at the root of the model. • Configure the model so that **Trigger type** is `function-call`.

Subcheck	Condition	Recommended Action
	States when enabling (`StatesWhenEnabling`) is set to `inherit`.	Set **States when enabling** to `held` or `reset`.
	The signal entering the Trigger Port of the parent subsystem: • Is from a Constant block. • Has a constant sample time.	Modify the model so that the signal entering the Trigger Port of the parent subsystem: • Is not from a Constant block. • Does not have a constant sample time.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals.	Fix the listed block inport or outport

Subcheck	Condition	Recommended Action
	• Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Action Port blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses: - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals.	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check If blocks	Block destination is a terminator block or an empty action subsystem.	Modify the model so that the block destination is not a terminator block or an empty action subsystem.
	Source of Inport 1 either: • Is a Constant block. • Has a constant sample time.	Modify the model so that the source of Input 1 is not a Constant block or have a constant sample time.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check Function-Call Generator blocks	The **Number of iterations** (`numberOfIterations`) is not set to `1`.	Set the **Number of iterations** to `1`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	
Check SwitchCase blocks	**Case conditions** (`CaseConditions`) has a range of values for the input.	Configure **Case conditions** so that the input does not have a range of values.
	Block destination is a terminator block or an empty action subsystem.	Modify the model so that the block destination is not a terminator block or an empty action subsystem.
	Source of Inport 1 either: • Is a Constant block. • Has a constant sample time.	Modify the model so that the source of Input 1 is not a Constant block or have a constant sample time.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Discontinuities blocks

Check for usage of Discontinuities blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Discontinuities blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Saturate blocks	Input and output ports do not have the same data type.	Modify the port data types to match.
	Block parameter **Upper limit** (`UpperLimit`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Upper limit** setting.
	Block parameter **Lower limit** (`LowerLimit`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Lower limit** setting.
	Block parameter `UpperLimitSource` is not set to `dialog`.	Use the block parameter **Upper limit** rather than input ports to specify the upper limit.
	Block parameter `LowerLimitSource` is not set to `dialog`.	Use the block parameter **Lower limit** rather than input ports to specify the lower limit.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to an unsupported value.	Set **Integer rounding mode** to `Zero`, `Floor`, or `Ceiling`.

Subcheck	Condition	Recommended Action

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix..
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
	• Block has constant (`Inf`) sample time and an outport has been testpointed.
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.
		Fix the listed block inport or outport.

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Sinks blocks

Check for usage of Sinks blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Sinks blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Outport blocks	The block cannot specify variable-dimension signals. Block parameter **Variable-size signal** (`VarSizeSig`) is set to `Yes`.	Set **Variable-size signal** to `No`.
	Signal type (`NumberOfTableDimensions`) is set to `complex`.	.Set **Signal type** to `real` or `auto`.
	Sampling mode (`SamplingMode`) is set to `Frame based`.	Set **Sampling mode** to `Sample based` or `auto`.
	Root level outport **Initial output** (`InitialOutput`) is not `[]`.	Set root level outport **Initial output** to `[]`.
	Source of initial output value (`SourceOfInitialOutputValue`) is not set to `Dialog`.	Set **Source of initial output value** to `Dialog`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Terminator blocks	Block is connected to a model reference block.	Modify the model so that the model reference block is not connected to a terminator block.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
	• Block has constant (`Inf`) sample time and an outport has been testpointed.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Discrete blocks

Check for usage of Discrete blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Discrete blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Unit Delay blocks	The block state does not have storage class `Auto`. Values other than `Auto` require use of storage classes, which are not supported for code inspection.	Modify the block such that its code generation storage class is set to `Auto`. If the block state name does not resolve to a signal object, set **Storage Class** in the **State Attributes** tab of the block parameter dialog box to `Auto`. If the block state name does resolve to a signal object, set the `CoderInfo.StorageClass` property of the signal object to `Auto`.
	Block parameter **Initial conditions** (`X0`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Fix the **Initial conditions** setting.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:	Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	- Port is not a bus for which the elements (potentially including other buses) meet the data type constraint. - Port has arrays of buses. - Port has buses with elements that are arrays of buses. • Block name contains character string `*/` or `/*`, or ends with the character `*`. • Block inport or outport is complex. • Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

Subcheck	Condition	Recommended Action
Check Discrete Integrator blocks	Input ports data types are not: • `single` or `double` for non-reset ports • `boolean` for external reset ports	Modify the input ports data types to be: • `single` or `double` for non-reset ports • `boolean` for external reset ports
	Inports and outports are not scalars.	Modify the inport or outports to be scalars.
	Output ports data types are not `single` or `double`.	Modify the output ports data types to be `single` or `double`.
	The input and output ports do not have the same data type.	Modify the port data types to match. The reset port data type does not need to match the other input and output data types.
	Block parameter **Integrator method** (`IntegratorMethod`) is not set to one of the following: • `Integration: Forward Euler` • `Integration: Backward Euler` • `Integration: Trapezoidal`	Set **Integrator method** to one of the following: • `Integration: Forward Euler` • `Integration: Backward Euler` • `Integration: Trapezoidal`
	Block parameter **Show state port** (`ShowStatePort`) is selected.	Clear **Show state port**.
	Block parameter **External reset** (`ExternalReset`) is set to `none` when the source of Inport 2 either: • Is a Constant block. • Has a constant sample time.	Modify the model so that the source of Input 2 is not a Constant block or have a constant sample time.
	Either or both block parameters **Upper saturation limit** (`UpperSaturationLimit`) and **Lower saturation limit** (`LowerSaturationLimit`):	Set both the **Upper saturation limit** and the **Lower saturation limit** to a one dimensional, non-complex, finite value.

Subcheck	Condition	Recommended Action
	• Are empty, non-finite, or complex. • Use MATLAB structures. • Have two or more dimensions. • Specify the `:` operator.	
	Block is inside a conditional subsystem.	Modify the model so that the Discrete Integrator block is not inside a conditional subsystem.

	Violates a constraint that applies to all blocks:
	• If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
	• Block inport or outport is not a scalar, vector, or 2D matrix.
	• Block inport or outport uses frame-based signals.
	• Block output custom signal storage class is not set to `Unstructured`.
		Fix the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-49

Check usage of Stateflow blocks

Check for usage of Stateflow blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow blocks.

Results and Recommended Actions

Check	Condition	Recommended Action
Check Stateflow blocks	**Function packaging** (`RTWSystemCode`) is not set to `Inline`.	Set **Function packaging** to `Inline`.

	Violates a constraint that applies to all blocks:
	• Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or `Enumerated` with default value `0`. If the block supports buses:
	  - Port is not a bus for which the elements (potentially including other buses) meet the data type constraint.
	  - Port has arrays of buses.
	  - Port has buses with elements that are arrays of buses.
	• Block name contains character string `*/` or `/*`, or ends with the character `*`.
	• Block inport or outport is complex.
		Fix the listed block inport or outport.

Check	Condition	Recommended Action
	• Block inport or outport is not a scalar, vector, or 2D matrix. • Block inport or outport uses frame-based signals. • Block output custom signal storage class is not set to `Unstructured`. • Block has constant (`Inf`) sample time and an outport has been testpointed. • Block output signal storage class is not set to `Auto` when the block has constant (`Inf`) sample time.	

See Also

- MATLAB Chart
- State Transition Table
- Truth Table

Check usage of Stateflow charts

Check for usage of Stateflow charts that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow charts.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check that control flows do not have cycles	Chart contains control flow cycles, which are not supported for code inspection.	Configure the chart so that it does not contain control flow cycles.
Check usage of Stateflow object palette	The chart contains one or more of the following objects: • States • Subcharts • Graphical functions • MATLAB functions • Truth Tables • Simulink functions	Configure the chart so that it does not contain the unsupported objects.
Check that all charts specify 'C' as their action language	Chart property **Action Language** is not set to `C`.	Set **Action Language** to `C`.
Check that all charts specify 'Inherited' as their update method	Chart property **Update method** is not set to `Inherited`.	Set **Update method** to `Inherited`.

Subcheck	Condition	Recommended Action
Check that no charts execute at initialization	Chart property **Execute (enter) Chart at Initialization** is selected (set to on).	Clear the chart property **Execute (enter) Chart at Initialization** parameter.
Check that no charts specify saturation on overflow for integer operations	Chart property **Saturate on integer overflow** is selected (set to on).	Clear the chart property **Saturate on integer overflow** parameter.
Check that no charts support variable-size arrays	Chart property **Support variable-size arrays** is selected (set to on).	Clear the chart property **Support variable-size arrays** parameter.
Check that control flows are structured	Chart contains unstructured control flows, which are not supported for code inspection.	Configure the chart so that it does not contain unstructured control flows.
Check that all control flows have unique default transitions	Control flow has more than 1 default transition.	Configure the chart so that it has 1 default transition.

Check usage of Stateflow transitions

Check for usage of Stateflow transitions that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow transitions.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check that transitions do not have unsupported operations	Transition uses an operation that is not: • `:=` or `=` • `+` , `+=` , `-`, or `-=` • `*` , `*=`, `/` or `/=` • `&`, `&&` or `&=` • `|`, `||` or `|=` • `<<`, `>>`, `++` or `--` • `cast()` • `^` or `^=` • `%%` or `<` • `<=` or `==` • `~=` or `!=` • `<>` or `>` • `>=` or `~`	Modify the chart so that transition uses only supported operations.
Check that no transitions access context-sensitive constants	Transition uses context-sensitive constants, which is not supported for code inspection.	Modify the transition to avoid using context-sensitive constants.

Subcheck	Condition	Recommended Action
Check that no transitions access custom data	Transition accesses custom data, which is not supported for code inspection.	Modify the transition to avoid accessing custom data.
Check that no transitions have event triggers	Transition has an event trigger, which is not supported for code inspection.	Modify the transition to avoid using an event trigger.
Check that transitions do not have transition actions	Transitions has a transition action, which is not supported for code inspection.	Modify the transition to avoid using a transition action.
Check that no transitions contain a binary operator whose operands are of mixed data type	Transition contains a binary operator of mixed data type operands, which is not supported for code inspection.	Modify the chart to avoid using binary operators with operands of mixed data type.
Check that no transitions access time (t)	Transitions accesses time, which is not supported for code inspection.	Modify the transition to avoid accessing time.

See Also

- "Graphical Expression of Modal Logic"
- "Transitions"
- "Transition Connections"
- "Default Transitions"

Check usage of Stateflow junctions

Check for usage of Stateflow junctions that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow junctions.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check that non-terminating junctions have	Non-terminating junction does not have exactly one unconditional exiting transition. A single	Modify junction so that it has one unconditional exiting transition.
Check that the chart uses no history junctions	Chart contains a history junction.	Modify chart so that it does not contain a history junction.
Check that unconditional transitions execute last in execution order	Unconditional transition is not last in order of execution.	Modify chart so that the unconditional transition is the last in order of execution. This prevents transition shadowing.

See Also

- "Connective Junctions"
- "History Junctions"

Check usage of Stateflow data

Check for usage of Stateflow data that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow data.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check that the chart uses no constants	Chart uses constants.	Modify chart so that it does not access constants.
Check that the chart uses no data stores	Chart uses data stores.	Modify chart so that it does not access data stores.
Check that Stateflow data is of a supported data type	Chart data types are not `builtin`, `enumerated`, or `bus`. If the chart data type is a `bus`, the data is an array of buses or has elements that are arrays of buses.	Modify chart data types to be `builtin`, `enumerated`, or `bus`. If the chart data type is `bus`, update the chart so that the data is not an array of buses or have elements that are arrays of buses.
Check that the chart uses only data with no initial values	Chart use data with initial values.	Modify chart sot that it does not use data with initial values.
Check that the chart uses no local data	Chart uses local data.	Modify chart so that it does not use local data.
Check that the chart uses no parameters	Chart uses parameters.	Modify chart so that it does not use parameters.

Subcheck	Condition	Recommended Action
Check that the chart uses only non-complex data	Chart uses complex data.	Modify chart so that it does not use complex data.
Check that the chart uses only scalar data	Chart uses non-scalar data.	Modify chart so that it does not use non-scalar data.

See Also

"Data Specification"

Check usage of Stateflow events

Check for usage of Stateflow events that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports incompatibilities it finds in Stateflow events.

Results and Recommended Actions

Check	Condition	Recommended Action
Check Stateflow events	Event scope is not `Output`.	Modify model so that event scope is `Output`.
	Event trigger is not `function-call`.	Modify model so that event trigger is `function-call`

See Also

"Input and Output Events"

Check usage of root Outport blocks

Check for usage of root Outport blocks that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports root Outport block usage incompatibilities.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify sample times	One or more root Outport blocks specify a constant (`Inf`) sample time. This will cause the model functions to fail validation, because the root outport assignment is moved to the model initialize function.	Set the sample times of the root Outport blocks to explicit, nonconstant sample times.
Verify root Outports pass buses to parent models as structures	One or more root Outport blocks pass a bus to the parent model without passing the bus as a structure. This might cause Simulink software to insert a hidden Signal Conversion block in the parent model, which is not supported for code inspection.	For each instance, open the Outport block dialog box and select the parameter **Output as nonvirtual bus in parent model** (`BusOutputAsStruct`).

See Also

"Other Modelwide Attribute Constraints" on page 4-18

Check usage of buses

Check for usage of buses that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports bus usage incompatibilities.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check for automatic conversion between virtual to non-virtual buses	Simulink software performed an automatic conversion from a virtual to a nonvirtual bus at the interface of one or more listed blocks. This creates a hidden Signal Conversion block, which is not supported for code inspection.	Modify the model to use nonvirtual buses at the interfaces of the listed blocks.
Verify that no blocks in the model perform an unsupported operation on a bus	In the model, a nonvirtual block operates on a virtual bus, or a Unit Delay block operates on a bus (virtual or nonvirtual).	Modify the model so that nonvirtual blocks operate on a virtual buses, and Unit Delay blocks operate onbuses. This action simplifies bus processing to promote traceability and readability of generated code.

See Also

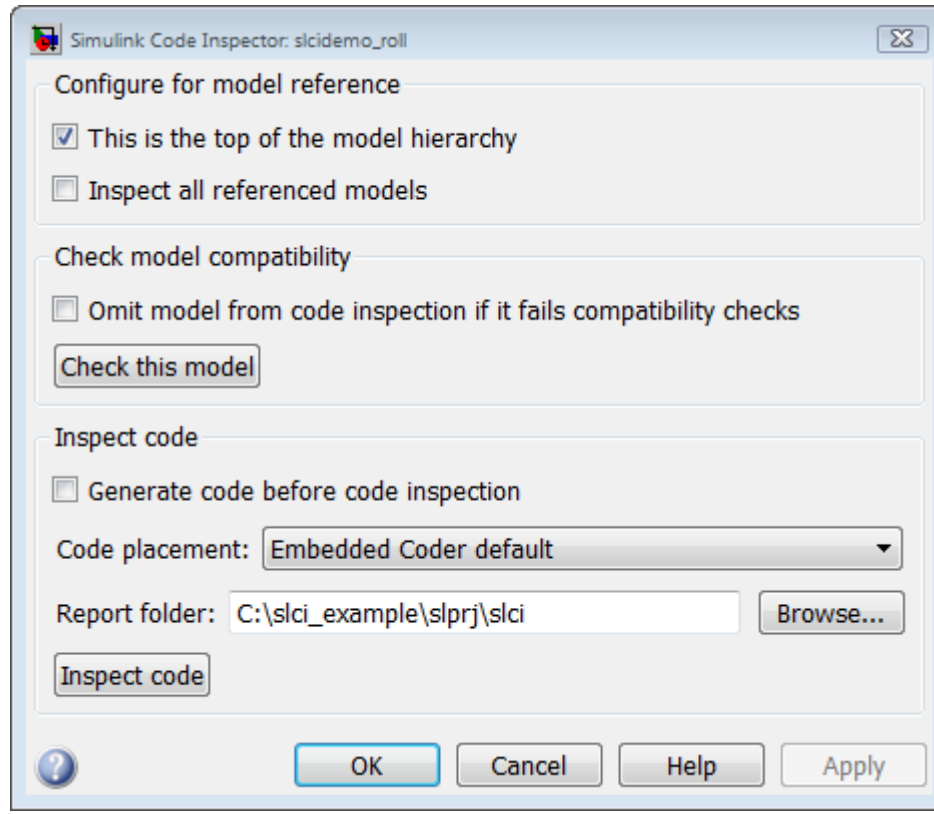
"Other Modelwide Attribute Constraints" on page 4-18

7

Simulink Code Inspector Dialog Box Parameters

Simulink Code Inspector Dialog Box

The Simulink Code Inspector dialog box with parameters at their initial default settings appears as follows.



In this section...

Simulink Code Inspector Dialog Box Overview

Control code inspection and compatibility checking for a model.

To get help on an option

1. Right-click the option's text label.
2. Select **What's This** from the popup menu.

See Also

- "Check Model Compatibility Using the Graphical User Interface"
- "Check Model Compatibility Using the Command-Line Interface"
- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

This is the top of the model hierarchy

Specify whether the model being configured for code inspection is the top model in the model reference hierarchy.

Settings

Default: on

 On

Code inspection (and code generation if requested) uses a top model target.

 Off

Code inspection (and code generation if requested) uses a model reference target.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setTopModel`.

See Also

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Inspect all referenced models

Specify whether model compatibility checking and code inspection should be performed for descendants of this model in the model reference hierarchy.

Settings

Default: off

 On

Model compatibility checking and code inspection are performed for descendants of this model in the model reference hierarchy.

 Off

Model compatibility checking and code inspection are performed only for this model.

Dependencies

Selecting **Inspect all referenced models** changes the displayed name for the option **Omit model from code inspection if it fails compatibility check** to **Omit models from code inspection if they fail compatibility checks**, and changes the displayed name of the button **Check this model** to **Check all models**.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setFollowModelLinks`.

See Also

- “Check Model Compatibility Using the Graphical User Interface”
- “Check Model Compatibility Using the Command-Line Interface”
- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Omit model from code inspection if it fails compatibility check

Specify whether code inspection terminates if a model fails compatibility checking.

Settings

Default: off

 On

Code inspection terminates if a model fails compatibility checking. Code generation (if requested) also does not occur.

 Off

Code inspection does not terminate if a model fails compatibility checking.

Dependencies

Selecting the option **Inspect all referenced models** changes the displayed name for this option from **Omit model from code inspection if it fails compatibility check** to **Omit models from code inspection if they fail compatibility checks**.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setTerminateOnIncompatibility`.

See Also

- “Check Model Compatibility Using the Graphical User Interface”
- “Check Model Compatibility Using the Command-Line Interface”
- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”

Generate code before code inspection

Specify whether to generate code before code inspection.

Settings

Default: off

 On

Generates model code at the beginning of code inspection.

 Off

Uses previously generated model code for code inspection.

Dependencies

Selecting **Generate code before code inspection** disables the **Code placement** and **Code folder** options, and changes the displayed name of the button **Inspect code** to **Generate and inspect code**.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setGenerateCode`.

See Also

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Code placement

Specify code placement for code inspection.

Settings

Default: `Embedded Coder default`

`Embedded Coder default`
: Specifies that previously generated code resides in the default folders created by code generation.

`Single folder`
: Specifies that previously generated code has been repackaged to reside in a single, user-defined folder.

Dependencies

- Clearing the option **Generate code before code inspection** enables the **Code placement** option.
- Selecting the value `Single folder` for **Code placement** enables the **Code folder** parameter.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setCodePlacement`.

See Also

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Code folder

Specify a folder containing previously generated code for code inspection.

Settings

Default: `''`

Specifies the path to a folder containing previously generated code to be inspected. Use this parameter only if you are inspecting generated code that has been repackaged to reside in a single, user-defined folder.

Dependencies

This parameter is enabled by setting the value of the **Code placement** parameter to `Single folder`.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setCodeFolder`.

See Also

- "Inspect Code Using the Graphical User Interface"
- "Inspect Code Using the Command-Line Interface"

Report folder

Specify a report folder for code inspection.

Settings

Default: Subfolder `slprj/slci` relative to the location of the model.

Specifies the path to a folder in which code inspection should place code inspection report artifacts.

Command-Line Information

The equivalent Simulink Code Inspector configuration method for selecting or clearing this option is `slci.Configuration.setReportFolder`.

See Also

- “Inspect Code Using the Graphical User Interface”
- “Inspect Code Using the Command-Line Interface”