

Simulink[®] Code Inspector[™]

Reference

R2012a

MATLAB[®]
& SIMULINK[®]

How to Contact MathWorks



www.mathworks.com
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab@mathworks.com)
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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

© COPYRIGHT 2011–2012 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

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

Function Reference

1

Code Inspection	1-2
Model Compatibility Checking	1-4

Class Reference

2

Code Inspection	2-2
-----------------------	-----

Functions — Alphabetical List

3

Model Configuration Constraints

4

About Model Configuration Constraints	4-2
Simulink Configuration Parameter Constraints	4-4
Solver	4-5
Data Import/Export	4-5
Optimization	4-5
Optimization: Signals and Parameters	4-7
Diagnostics: Data Validity	4-7
Diagnostics: Connectivity	4-8
Diagnostics: Model Referencing	4-9

Hardware Implementation	4-9
Model Referencing	4-11
Code Generation: General	4-11
Code Generation: Comments	4-11
Code Generation: Symbols	4-12
Code Generation: Custom Code	4-12
Code Generation: Interface	4-13
Code Generation: SIL and PIL Verification	4-15
Code Generation: Code Style	4-15
Code Generation: Data Type Replacement	4-15
Code Generation: Not in GUI	4-16
Other Modelwide Attribute Constraints	4-17

Block Constraints

5

About Block Constraints	5-2
Block Constraints — Alphabetical List	5-5
All Blocks	5-6
Abs	5-7
Bus Assignment	5-8
Bus Creator	5-8
Bus Selector	5-9
Constant	5-9
Data Store Memory	5-10
Data Store Read	5-10
Data Store Write	5-11
Data Type Conversion	5-11
Data Type Duplicate	5-12
Demux	5-12
DocBlock	5-13
Enable Port	5-13
From	5-14
Gain	5-14
Goto	5-15
Ground	5-15
Inport	5-15
Logical Operator	5-16

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)	5-16
Math Function	5-19
MinMax	5-20
Model	5-20
Model Info	5-21
Multiport Switch	5-21
Mux	5-22
Outport	5-22
Product	5-22
Relational Operator	5-23
Rounding Function	5-24
Saturation	5-24
Selector	5-25
S-Function	5-25
Signal Conversion	5-26
Subsystem, Atomic Subsystem, Enabled Subsystem	5-27
Sum, Add, Subtract	5-27
Switch	5-28
Terminator	5-28
Trigonometric Function	5-29
Unit Delay	5-29

Supported Blocks — By Category	5-30
Commonly Used Blocks	5-30
Discontinuity Blocks	5-31
Discrete Blocks	5-31
Logic and Bit Operation Blocks	5-31
Lookup Tables	5-31
Math Operation Blocks	5-31
Model-Wide Utilities	5-32
Port & Subsystem Blocks	5-32
Signal Attribute Blocks	5-32
Signal Routing Blocks	5-32
Sink Blocks	5-33
Source Blocks	5-33
User-Defined Functions	5-33

Simulink® Code Inspector™ Checks	6-2
Simulink® Code Inspector™ Checks Overview	6-4
Check code generation settings	6-5
Check data import/export settings	6-10
Check diagnostic settings	6-11
Check hardware implementation settings	6-13
Check model reference settings	6-15
Check optimization settings	6-16
Check solver settings	6-19
Check for unconnected objects in the model	6-20
Check system target file setting	6-21
Check function specification setting	6-22
Check model arguments	6-23
Check conditional input branch execution setting	6-24
Check for unsupported blocks	6-25
Check for tunable workspace variables	6-26
Check for sample times in the model	6-27
Check for usage of global data stores	6-28
Check for Signal Conversion blocks automatically inserted on signals entering block input ports	6-29
Check for usage of fixed-point instrumentation	6-30
Check for root Outputport blocks being conditionally assigned	6-31
Check usage of Sources blocks	6-32
Check usage of Signal Routing blocks	6-35
Check usage of Math Operations blocks	6-47
Check usage of Signal Attributes blocks	6-55
Check usage of Logical and Bit Operations blocks	6-58
Check usage of Lookup Tables blocks	6-61
Check usage of User-Defined Function blocks	6-65
Check usage of Ports and Subsystems blocks	6-67
Check usage of Discontinuities blocks	6-70
Check usage of Sinks blocks	6-72
Check usage of Discrete blocks	6-74
Check usage of root Outputport blocks	6-76
Check usage of buses	6-77

Simulink® Code Inspector™ Dialog Box Parameters

7

Simulink Code Inspector Dialog Box	7-2
Simulink Code Inspector Dialog Box Overview	7-4
This is the top of the model hierarchy	7-5
Inspect all referenced models	7-6
Omit model from code inspection if it fails compatibility check	7-7
Generate code before code inspection	7-8
Code placement	7-9
Code folder	7-10
Report folder	7-11

Function Reference

Code Inspection (p. 1-2)

Inspect code generated from a model

Model Compatibility Checking
(p. 1-4)

Prepare for code inspection

Code Inspection

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

scli.Configuration

Create code inspection object

scli.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

Class Reference

Code Inspection

slci.Configuration

Control code inspection and compatibility checking for model

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(modeName);`

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: <ul style="list-style-type: none">• 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

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

Examples

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

slci.Configuration.checkCompatibility

```
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 **Tools** 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

- “Model Compatibility Checking”
- “Code Inspection”

Purpose	Return code folder for code inspection		
Syntax	<code>folder = getCodeFolder(cfgObj)</code>		
Description	<code>folder = getCodeFolder(cfgObj)</code> returns the path to a code folder, as previously specified using <code>slci.Configuration.setCodeFolder</code> . 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 <code>slci.Configuration.setCodePlacement</code> .		
Input Arguments	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code>;</td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;		
Output Arguments	<table><tr><td><code>folder</code></td><td>String specifying a folder path or, if you have not previously set a code folder value, '' (default).</td></tr></table>	<code>folder</code>	String specifying a folder path or, if you have not previously set a code folder value, '' (default).
<code>folder</code>	String specifying a folder path or, if you have not previously set a code folder value, '' (default).		
Examples	<pre>>> 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 >></pre>		
Alternatives	Open the Simulink Code Inspector dialog box from Tools menu of the model window and use the dialog box to configure and run code inspection.		

slci.Configuration.getCodeFolder

See Also

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

How To

- “Code Inspection”

slci.Configuration.getCodePlacement

Purpose	Return code placement for code inspection		
Syntax	<code>value = getCodePlacement(cfgObj)</code>		
Description	<code>value = getCodePlacement(cfgObj)</code> 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	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code></td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>		
Output Arguments	<table><tr><td><code>value</code></td><td>String specifying one of the following values:<ul style="list-style-type: none">• <code>Single folder</code> if the generated code has been repackaged to reside in a single, user-defined folder.• <code>Embedded Coder default</code> (default) if the generated code resides in the default folders created by code generation.</td></tr></table>	<code>value</code>	String specifying one of the following values: <ul style="list-style-type: none">• <code>Single folder</code> if the generated code has been repackaged to reside in a single, user-defined folder.• <code>Embedded Coder default</code> (default) if the generated code resides in the default folders created by code generation.
<code>value</code>	String specifying one of the following values: <ul style="list-style-type: none">• <code>Single folder</code> if the generated code has been repackaged to reside in a single, user-defined folder.• <code>Embedded Coder default</code> (default) if the generated code resides in the default folders created by code generation.		
Examples	<pre>>> 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 >></pre>		

slci.Configuration.getCodePlacement

Alternatives Open the Simulink Code Inspector dialog box from **Tools** 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 • “Code Inspection”

slci.Configuration.getFollowModelLinks

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 all descendants of this model in the model reference hierarchy.

Input Arguments

<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code> .
---------------------	----------------------------------------------------------------------------------------------------------------------------------------

Output Arguments

<code>value</code>	True if model compatibility checking and code inspection should be performed for all descendants 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 **Tools** 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`

slci.Configuration.getFollowModelLinks

How To

- “Code Inspection”
- “Model Compatibility Checking”

Purpose	Return code generation option for code inspection		
Syntax	<code>value = getGenerateCode(cfgObj)</code>		
Description	<code>value = getGenerateCode(cfgObj)</code> returns the value of a code inspection option that specifies whether to generate model code as part of code inspection.		
Input Arguments	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code></td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>		
Output Arguments	<table><tr><td><code>value</code></td><td>True if model code should be generated at the beginning of code inspection; false otherwise. The default is false.</td></tr></table>	<code>value</code>	True if model code should be generated at the beginning of code inspection; false otherwise. The default is false.
<code>value</code>	True if model code should be generated at the beginning of code inspection; false otherwise. The default is false.		
Examples	<pre>>> config = slci.Configuration('slcidemo_roll'); >> config.setGenerateCode(true) >> value = config.getGenerateCode() value = 1 >></pre>		
Alternatives	Open the Simulink Code Inspector dialog box from Tools menu of the model window and use the dialog box to configure and run code inspection.		
See Also	<code>slci.Configuration.setGenerateCode</code>		
How To	<ul style="list-style-type: none">• “Code Inspection”		

slci.Configuration.getReportFolder

Purpose	Return report folder for code inspection		
Syntax	<code>folder = getReportFolder(cfgObj)</code>		
Description	<code>folder = getReportFolder(cfgObj)</code> returns the path to a folder in which code inspection places code inspection report artifacts.		
Input Arguments	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code></td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>		
Output Arguments	<table><tr><td><code>folder</code></td><td>String specifying a folder path. If you have not previously set a report folder value, the default is <code>slprj/slci</code>, relative to the location of the model.</td></tr></table>	<code>folder</code>	String specifying a folder path. If you have not previously set a report folder value, the default is <code>slprj/slci</code> , relative to the location of the model.
<code>folder</code>	String specifying a folder path. If you have not previously set a report folder value, the default is <code>slprj/slci</code> , relative to the location of the model.		
Examples	<pre>>> 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 >></pre>		
Alternatives	Open the Simulink Code Inspector dialog box from Tools menu of the model window and use the dialog box to configure and run code inspection.		

See Also

`slci.Configuration.setReportFolder`

How To

- “Code Inspection”

slci.Configuration.getTerminateOnIncompatibility

Purpose	Return termination option for code inspection		
Syntax	<code>value = getTerminateOnIncompatibility(cfgObj)</code>		
Description	<code>value = getTerminateOnIncompatibility(cfgObj)</code> 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	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code></td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>		
Output Arguments	<table><tr><td><code>value</code></td><td>True if code inspection should terminate if a model fails code inspection; false otherwise. The default is false.</td></tr></table>	<code>value</code>	True if code inspection should terminate if a model fails code inspection; false otherwise. The default is false.
<code>value</code>	True if code inspection should terminate if a model fails code inspection; false otherwise. The default is false.		
Examples	<pre>>> config = slci.Configuration('slcidemo_roll'); >> config.setTerminateOnIncompatibility(true) >> value = config.getTerminateOnIncompatibility() value = 1 >></pre>		
Alternatives	Open the Simulink Code Inspector dialog box from Tools menu of the model window and use the dialog box to configure and run code inspection.		
See Also	<code>slci.Configuration.setTerminateOnIncompatibility</code> <code>slci.Configuration.checkCompatibility</code>		
How To	<ul style="list-style-type: none">• “Code Inspection”		

slci.Configuration.getTerminateOnIncompatibility

- “Model Compatibility Checking”

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

<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code> .
---------------------	----------------------------------------------------------------------------------------------------------------------------------------

Output Arguments

<code>value</code>	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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

See Also

`slci.Configuration.setTopModel`

How To

- “Code Inspection”

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 as appropriate, 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(modeName);`.

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: <ul style="list-style-type: none">• 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`).

slci.Configuration.inspect

```
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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

See Also `slci.Configuration.checkCompatibility`

How To

- “Code Inspection”
- “Model Compatibility Checking”

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	<i>cfgObj</i>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;
	<i>folder</i>	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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

slci.Configuration.setCodeFolder

See Also

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

How To

- “Code Inspection”

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 =
```

slci.Configuration.setCodePlacement

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

Alternatives Open the Simulink Code Inspector dialog box from **Tools** 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

- “Code Inspection”

slci.Configuration.setFollowModelLinks

Purpose	Specify model reference handling for model compatibility checking or code inspection				
Syntax	<code>setFollowModelLinks(cfgObj, followModelLinks)</code>				
Description	<code>setFollowModelLinks(cfgObj, followModelLinks)</code> specifies whether model compatibility checking and code inspection should be performed for all descendants of this model in the model reference hierarchy.				
Input Arguments	<table><tr><td><code>cfgObj</code></td><td>Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code>;</td></tr><tr><td><code>followModelLinks</code></td><td>True if model compatibility checking and code inspection should be performed for all descendants of this model in the model reference hierarchy; false otherwise. The default is false.</td></tr></table>	<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;	<code>followModelLinks</code>	True if model compatibility checking and code inspection should be performed for all descendants of this model in the model reference hierarchy; false otherwise. The default is false.
<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;				
<code>followModelLinks</code>	True if model compatibility checking and code inspection should be performed for all descendants of this model in the model reference hierarchy; false otherwise. The default is false.				
Examples	<pre>>> config = slci.Configuration('slcidemo_roll'); >> config.setFollowModelLinks(true) >> value = config.getFollowModelLinks() value = 1 >></pre>				
Alternatives	Open the Simulink Code Inspector dialog box from Tools menu of the model window and use the dialog box to configure and run code inspection.				
See Also	<code>slci.Configuration.getFollowModelLinks</code>				
How To	<ul style="list-style-type: none">• “Code Inspection”• “Model Compatibility Checking”				

slci.Configuration.setGenerateCode

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

<i>cfgObj</i>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName)</code> ;
<i>generateCode</i>	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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

See Also `slci.Configuration.getGenerateCode`

How To

- “Code Inspection”

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

<i>cfgObj</i>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code> .
<i>folder</i>	String specifying a folder path. If you have not previously set a report folder value, the default is <code>slprj/slci</code> , 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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

See Also `slci.Configuration.getReportFolder`

slci.Configuration.setReportFolder

How To

- “Code Inspection”

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

<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>terminate</code>	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 **Tools** 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

- “Code Inspection”
- “Model Compatibility Checking”

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

<code>cfgObj</code>	Handle to a Simulink Code Inspector configuration object previously returned by <code>cfgObj = slci.Configuration(modelName);</code>
<code>top</code>	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 **Tools** menu of the model window and use the dialog box to configure and run code inspection.

See Also `slci.Configuration.getTopModel`

How To

- “Code Inspection”

slci.Configuration

Purpose	Control code inspection and compatibility checking for model	
Description	An <code>slci.Configuration</code> object configures code inspection and compatibility checking for a model.	
Construction	<code>slci.Configuration</code>	Create code inspection object
Methods	<code>checkCompatibility</code>	Check model compatibility with code inspection
	<code>getCodeFolder</code>	Return code folder for code inspection
	<code>getCodePlacement</code>	Return code placement for code inspection
	<code>getFollowModelLinks</code>	Return model reference handling for model compatibility checking or code inspection
	<code>getGenerateCode</code>	Return code generation option for code inspection
	<code>getReportFolder</code>	Return report folder for code inspection
	<code>getTerminateOnIncompatibility</code>	Return termination option for code inspection
	<code>getTopModel</code>	Return top-model attribute for code inspection
	<code>inspect</code>	Inspect code generated from model
	<code>setCodeFolder</code>	Specify code folder for code inspection

<code>setCodePlacement</code>	Specify code placement for code inspection
<code>setFollowModelLinks</code>	Specify model reference handling for model compatibility checking or code inspection
<code>setGenerateCode</code>	Specify whether to generate code before code inspection
<code>setReportFolder</code>	Specify report folder for code inspection
<code>setTerminateOnIncompatibility</code>	Specify whether to terminate code inspection if model is incompatible
<code>setTopModel</code>	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 demo `slcidemo_intro` shows how to programmatically run the compatibility checker and the Code Inspector and report results. The demo 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

- “Code Inspection”

- “Model Compatibility Checking”

Purpose	Create code inspection object		
Syntax	<code>cfgObj = slci.Configuration(modelName)</code>		
Description	<code>cfgObj = slci.Configuration(modelName)</code> creates an object of class <code>slci.Configuration</code> and returns a handle to it.		
Input Arguments	<table><tr><td><code>modelName</code></td><td>Name of the model for which you are configuring code inspection and compatibility checking.</td></tr></table>	<code>modelName</code>	Name of the model for which you are configuring code inspection and compatibility checking.
<code>modelName</code>	Name of the model for which you are configuring code inspection and compatibility checking.		
Output Arguments	<table><tr><td><code>cdgObj</code></td><td>Handle to code inspection object.</td></tr></table>	<code>cdgObj</code>	Handle to code inspection object.
<code>cdgObj</code>	Handle to code inspection object.		
Examples	<p>This example creates a code inspection object, <code>config</code>, and uses it to check the specified model for compatibility with code inspection.</p> <pre>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</pre>		
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	<ul style="list-style-type: none">• “Code Inspection”• “Model Compatibility Checking”		

slci.ExportTraceReport

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 3-37. *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 3-37. *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 3-37. *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 with no 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 demo 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.

slci.ExportTraceReport

- 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

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 demo model `slcidemo_roll_orig`.

1 Open the demo 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

- “Model Compatibility Checking”

Model Configuration Constraints

- “About Model Configuration Constraints” on page 4-2
- “Simulink Configuration Parameter Constraints” on page 4-4
- “Other Modelwide Attribute Constraints” on page 4-17

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 4-17 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 any 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” on page 4-5
“Data Import/Export” on page 4-5
“Optimization” on page 4-5
“Optimization: Signals and Parameters” on page 4-7
“Diagnostics: Data Validity” on page 4-7
“Diagnostics: Connectivity” on page 4-8
“Diagnostics: Model Referencing” on page 4-9
“Hardware Implementation” on page 4-9
“Model Referencing” on page 4-11
“Code Generation: General” on page 4-11
“Code Generation: Comments” on page 4-11
“Code Generation: Symbols” on page 4-12
“Code Generation: Custom Code” on page 4-12
“Code Generation: Interface” on page 4-13
“Code Generation: SIL and PIL Verification” on page 4-15
“Code Generation: Code Style” on page 4-15
“Code Generation: Data Type Replacement” on page 4-15
“Code Generation: Not in GUI” on page 4-16

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 parameters (InlineParams)	Must be selected (set to on).	FATAL	Check optimization settings > Verify 'Inline parameters' setting
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

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

Diagnostics Pane: Data Validity			
Parameter	Constraint	FATAL / Nonfatal	Compatibility Check
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 ErrorOnBusTreatedAs-Vector specified at the command line).	FATAL	Check diagnostic settings > Verify Bus signal treated as vector setting
Non-bus signals treated as bus signals (NonBusSignalsTreated-AsBus)	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/Output 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/Output 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

Hardware Implementation Pane			
Parameter	Constraint	FATAL / Nonfatal	Compatibility Check
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
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

Model Referencing

Model Referencing Pane			
Parameter	Constraint	FATAL / Nonfatal	Compatibility Check
Total number of instances allowed per top model (ModelReferenceNum-InstancesAllowed)	Must be set to Multiple (Multi at the command line) or Zero. If set to Single, the model interface might fail validation.	Nonfatal	Check model reference settings > Verify 'Total number of instances allowed per top model' 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

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

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
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) in the Configuration Parameters dialog box or ANSI_C at the command line.	Nonfatal	Check code generation settings > Verify 'Code replacement library' setting
Shared code placement (UtilityFuncGeneration)	Must be set to Shared location.	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: SIL and PIL Verification

Code Generation Pane: SIL and PIL 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 be no 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
Model arguments	There must be no model arguments defined for the model.	Nonfatal	Check model arguments
Conditional input branch execution	The model must enable local block outputs when using conditional input branch execution.	Nonfatal	Check conditional input branch execution setting
Unsupported blocks	There must be no blocks in the model that are not supported by Simulink Code Inspector.	Nonfatal	Check for unsupported blocks
Tunable workspace variables	The model cannot reference workspace variables that are tunable. This would require use of storage classes, which are not supported for code inspection.	Nonfatal	Check for tunable workspace variables

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Usage of sample times	The model cannot use multiple, variable, continuous, or asynchronous sample times.	FATAL	Check for sample times in the model
Usage of global data stores	Data Store Read and Data Store Write blocks cannot reference externally-defined signal objects as global data stores. They trigger creation of a hidden Data Store Memory block at the root level of the model, which is not supported for code inspection.	FATAL	Check for usage of global data stores
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 Output blocks	Root outputs of referenced models cannot be connected to conditionally executed subsystems, either directly or through a nonvirtual Bus Creator, Bus Selector, or Signal Conversion block.	Nonfatal	Check for root Output blocks being conditionally assigned
Root Output block sample times	Root Output blocks cannot specify a constant (Inf) sample time. This constraint prevents the root output assignment from being moved to the model initialize function, which would cause the model functions to fail validation.	Nonfatal	Check usage of root Output blocks > Verify sample times
Root Output block bus passing method	A root Output 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 Output blocks > Verify root Outputs 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

Block Constraints

- “About Block Constraints” on page 5-2
- “Block Constraints — Alphabetical List” on page 5-5
- “Supported Blocks — By Category” on page 5-30

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 any 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, if any
- Block parameter constraints that apply to the block, if any

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 any 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 all 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 or Floor.	Nonfatal	

“All Blocks” on page 5-6 lists constraints that apply to all supported blocks.

“Supported Blocks — By Category” on page 5-30 presents the supported blocks by category and provides links to the block-specific constraints.

Note All 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...

“All Blocks” on page 5-6

“Abs” on page 5-7

“Bus Assignment” on page 5-8

“Bus Creator” on page 5-8

“Bus Selector” on page 5-9

“Constant” on page 5-9

“Data Store Memory” on page 5-10

“Data Store Read” on page 5-10

“Data Store Write” on page 5-11

“Data Type Conversion” on page 5-11

“Data Type Duplicate” on page 5-12

“Demux” on page 5-12

“DocBlock” on page 5-13

“Enable Port” on page 5-13

“From” on page 5-14

“Gain” on page 5-14

“Goto” on page 5-15

“Ground” on page 5-15

“Inport” on page 5-15

“Logical Operator” on page 5-16

“1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)” on page 5-16

“Math Function” on page 5-19

“MinMax” on page 5-20

“Model” on page 5-20

In this section...
“Model Info” on page 5-21
“Multiport Switch” on page 5-21
“Mux” on page 5-22
“Outport” on page 5-22
“Product” on page 5-22
“Relational Operator” on page 5-23
“Rounding Function” on page 5-24
“Saturation” on page 5-24
“Selector” on page 5-25
“S-Function” on page 5-25
“Signal Conversion” on page 5-26
“Subsystem, Atomic Subsystem, Enabled Subsystem” on page 5-27
“Sum, Add, Subtract” on page 5-27
“Switch” on page 5-28
“Terminator” on page 5-28
“Trigonometric Function” on page 5-29
“Unit Delay” on page 5-29

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, or boolean. If the block supports buses, the ports can be buses for	FATAL	All block compatibility checks

Constraints that apply to all blocks			
	Constraint	FATAL / Nonfatal	Compatibility Check
	which the elements (potentially including other buses) meet the data type constraint.		
	Input and output ports must be noncomplex. Complex values are not supported for code inspection.	Nonfatal	
	Input and output ports must be scalars or vectors (not multidimensional).	Nonfatal	
	Input and output ports must not use frame-based signals.	Nonfatal	
	Output signal storage class must be set to Auto. Values other than Auto require use of storage classes, which are not supported for code inspection.	Nonfatal	
	Output port must not be testpointed 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 all have the same data type.	Nonfatal	

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

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
	This block can only operate on a virtual bus. This constraint simplifies bus processing to promote traceability and readability of generated code.	FATAL	
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	
	Interpret vector parameters as 1-D (VectorParams1D) must be selected (set to on).	Nonfatal	

Data Store Memory

Data Store Memory			
	Constraint	FATAL / Nonfatal	Compatibility Check
Data Types	Constraints that apply to all blocks.		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	
	Interpret vector parameters as 1-D (VectorParams1D) must be selected (set to on).	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	

Note Data Store Read and Data Store Write blocks cannot reference externally-defined signal objects as global data stores. For more information, see modelwide constraints.

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	

Note Data Store Read and Data Store Write blocks cannot reference externally-defined signal objects as global data stores. For more information, see modelwide constraints.

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	

Data Type Conversion			
	Constraint	FATAL / Nonfatal	Compatibility Check
	Integer rounding mode (RndMeth) must be set to Zero or Floor.	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		

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 blocks are not supported at the root level of the model.	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		

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 all 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).	Nonfatal	
	Integer rounding mode (RndMeth) must be set to Zero or Floor.	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		

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 <i>not</i> be set to Yes.	Nonfatal	

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.	FATAL	
	Block must have at least two inports, except in the case of the NOT operator.	FATAL	
	The block input ports should all 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 all have the same data type.	Nonfatal	
	Input and output ports must be scalars.	Nonfatal	
Block Parameters	Number of table dimensions (NumberOfTableDimensions) must be set to 1 or 2.	Nonfatal	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	FATAL / Nonfatal	Compatibility Check
	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	
	Table data (Table) must not: be empty, be nonfinite, have a MATLAB structure as a value,	FATAL	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	FATAL / Nonfatal	Compatibility Check
	be complex, have two or more dimensions, or specify the range (:) operator.		
	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) must use the same data type as the block output.	Nonfatal	

1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table			
	Constraint	FATAL / Nonfatal	Compatibility Check
	Integer rounding mode (RndMeth) must be set to Zero.	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 all 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, pow, reciprocal, hypot, rem, or mod. You cannot select conj, transpose, or hermitian.	FATAL	
	Integer rounding mode (RndMeth) must be set to Zero or Floor.	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 all 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 or Floor.	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	

Note Referenced models cannot accept model arguments. For more information, see modelwide constraints.

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 all have the same data type.	Nonfatal	
	Block must have at least three inports.	FATAL	
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 or Floor.	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		

Output

Output			
	Constraint	FATAL / Nonfatal	Compatibility Check
Data Types	Constraints that apply to all blocks.		Check usage of Sinks blocks > Check Output blocks
	No block-specific constraints		
Block Parameters	The block cannot specify variable-dimension signals. Variable-size signal (VarSizeSig) must <i>not</i> be set to Yes.	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 all have the same data type.	Nonfatal	

Product			
	Constraint	FATAL / Nonfatal	Compatibility Check
Block Parameters	Number of inputs (inputs) must be set to 2, **, /*, or */.	Nonfatal	
	Multiplication (Multiplication) must be set to Element-wise(.*)).	Nonfatal	
	Integer rounding mode (RndMeth) must be set to Zero or Floor.	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 all have the same data type.	Nonfatal	
Block Parameters	Relational operator (Operator) must be set to <=, ==, >=, ~=, <, or > (not isInf, isNaN, or isFinite).	FATAL	

Rounding Function

Rounding Function			
	Constraint	FATAL / Nonfatal	Compatibility Check
Data Types	Constraints that apply to all blocks.		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 all 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 or Floor.	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
	No block-specific constraints		
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
	All arguments must be scalars, or vectors of fixed dimension.	Nonfatal	
Block Parameters	S-functions: <ul style="list-style-type: none"> • 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. 	Nonfatal	

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	

Subsystem, Atomic Subsystem, Enabled Subsystem

Subsystem, Atomic Subsystem, Enabled 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 virtual subsystems, or inlined atomic or enabled subsystems.	FATAL	

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 all have the same data type.	Nonfatal	
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 or Floor.	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 all have the same data type.	Nonfatal	
Block Parameters	Integer rounding mode (RndMeth) must be set to Zero or Floor.	Nonfatal	
	Allow different data input sizes (AllowDiffInputSizes) must not be selected (must be set to off).	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	No block-specific constraints		

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 <i>not</i> be set to cos + jsin (complex exponential of the input) or sincos.	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	

Supported Blocks – By Category

In this section...
“Commonly Used Blocks” on page 5-30
“Discontinuity Blocks” on page 5-31
“Discrete Blocks” on page 5-31
“Logic and Bit Operation Blocks” on page 5-31
“Lookup Tables” on page 5-31
“Math Operation Blocks” on page 5-31
“Model-Wide Utilities” on page 5-32
“Port & Subsystem Blocks” on page 5-32
“Signal Attribute Blocks” on page 5-32
“Signal Routing Blocks” on page 5-32
“Sink Blocks” on page 5-33
“Source Blocks” on page 5-33
“User-Defined Functions” on page 5-33

Commonly Used Blocks

- “Bus Creator” on page 5-8
- “Bus Selector” on page 5-9
- “Constant” on page 5-9
- “Data Type Conversion” on page 5-11
- “Demux” on page 5-12
- “Gain” on page 5-14
- “Ground” on page 5-15
- “Inport” on page 5-15
- “Logical Operator” on page 5-16

- “Mux” on page 5-22
- “Outport” on page 5-22
- “Product” on page 5-22
- “Relational Operator” on page 5-23
- “Saturation” on page 5-24
- “Subsystem, Atomic Subsystem, Enabled Subsystem” on page 5-27
- “Sum, Add, Subtract” on page 5-27
- “Switch” on page 5-28
- “Terminator” on page 5-28
- “Unit Delay” on page 5-29

Discontinuity Blocks

- “Saturation” on page 5-24

Discrete Blocks

- “Unit Delay” on page 5-29

Logic and Bit Operation Blocks

- “Logical Operator” on page 5-16
- “Relational Operator” on page 5-23

Lookup Tables

- “1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)” on page 5-16

Math Operation Blocks

- “Abs” on page 5-7
- “Gain” on page 5-14

- “Math Function” on page 5-19
- “MinMax” on page 5-20
- “Product” on page 5-22
- “Rounding Function” on page 5-24
- “Sum, Add, Subtract” on page 5-27
- “Trigonometric Function” on page 5-29

Model-Wide Utilities

- “DocBlock” on page 5-13
- “Model Info” on page 5-21

Port & Subsystem Blocks

- “Enable Port” on page 5-13
- “Inport” on page 5-15
- “Model” on page 5-20
- “Outport” on page 5-22
- “Subsystem, Atomic Subsystem, Enabled Subsystem” on page 5-27

Signal Attribute Blocks

- “Data Type Conversion” on page 5-11
- “Data Type Duplicate” on page 5-12
- “Signal Conversion” on page 5-26

Signal Routing Blocks

- “Bus Assignment” on page 5-8
- “Bus Creator” on page 5-8
- “Bus Selector” on page 5-9

- “Data Store Memory” on page 5-10
- “Data Store Read” on page 5-10
- “Data Store Write” on page 5-11
- “Demux” on page 5-12
- “From” on page 5-14
- “Goto” on page 5-15
- “Multiport Switch” on page 5-21
- “Mux” on page 5-22
- “Selector” on page 5-25
- “Switch” on page 5-28

Sink Blocks

- “Outport” on page 5-22
- “Terminator” on page 5-28

Source Blocks

- “Constant” on page 5-9
- “Ground” on page 5-15
- “Inport” on page 5-15

User-Defined Functions

- “S-Function” on page 5-25

Model Advisor Checks

Simulink Code Inspector Checks

In this section...

- “Simulink® Code Inspector™ Checks Overview” on page 6-4
- “Check code generation settings” on page 6-5
- “Check data import/export settings” on page 6-10
- “Check diagnostic settings” on page 6-11
- “Check hardware implementation settings” on page 6-13
- “Check model reference settings” on page 6-15
- “Check optimization settings” on page 6-16
- “Check solver settings” on page 6-19
- “Check for unconnected objects in the model” on page 6-20
- “Check system target file setting” on page 6-21
- “Check function specification setting” on page 6-22
- “Check model arguments” on page 6-23
- “Check conditional input branch execution setting” on page 6-24
- “Check for unsupported blocks” on page 6-25
- “Check for tunable workspace variables” on page 6-26
- “Check for sample times in the model” on page 6-27
- “Check for usage of global data stores” on page 6-28
- “Check for Signal Conversion blocks automatically inserted on signals entering block input ports” on page 6-29
- “Check for usage of fixed-point instrumentation” on page 6-30
- “Check for root Outport blocks being conditionally assigned” on page 6-31
- “Check usage of Sources blocks” on page 6-32
- “Check usage of Signal Routing blocks” on page 6-35
- “Check usage of Math Operations blocks” on page 6-47
- “Check usage of Signal Attributes blocks” on page 6-55

In this section...

“Check usage of Logical and Bit Operations blocks” on page 6-58

“Check usage of Lookup Tables blocks” on page 6-61

“Check usage of User-Defined Function blocks” on page 6-65

“Check usage of Ports and Subsystems blocks” on page 6-67

“Check usage of Discontinuities blocks” on page 6-70

“Check usage of Sinks blocks” on page 6-72

“Check usage of Discrete blocks” on page 6-74

“Check usage of root Outport blocks” on page 6-76

“Check usage of buses” on page 6-77

Simulink Code Inspector Checks Overview

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

See Also

- “Consulting 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.	Go to Configuration Parameters > Code Generation and set Language to C or C++.
Verify 'Shared code placement' setting	The model is not configured to generate shared utility code to a shared location. If shared utility code is generated into <i>model.c</i> , the Code Inspector reports the code as unverified.	Go to Configuration Parameters > Code Generation > Interface and set Shared code placement to Shared location.
Verify 'Source file' setting	Custom code is configured to appear near the top of the generated model source file.	Go to Configuration Parameters > Code Generation > Custom Code and clear the Source file field.
Verify 'Initialize function' setting	Custom code is configured to appear in the generated model initialize function.	Go to Configuration Parameters > Code Generation > Custom Code and clear the Initialize function field.
Verify 'Terminate function' setting	Custom code is configured to appear in the generated model terminate function.	Go to Configuration Parameters > Code Generation > Custom Code and clear the Terminate function field.

Subcheck	Condition	Recommended Action
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.	Go to Configuration Parameters > Code Generation > Interface and clear the Combine signal/state structures option.
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.	Go to Configuration Parameters > Code Generation > Comments and 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.	Go to Configuration Parameters > Code Generation > Symbols and 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.	Go to Configuration Parameters > Code Generation > Code Style and 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.	Go to Configuration Parameters > Code Generation > Data Type Replacement and clear the Replace data type names in the generated code option.
Verify 'Code replacement library' setting	A code replacement library other than C89/C90 (ANSI), the ANSI C library supported for code inspection, is selected for the model.	Go to Configuration Parameters > Code Generation > Interface and set Code replacement library to C89/C90 (ANSI) (equivalent to ANSI_C specified at the command line).

Subcheck	Condition	Recommended Action
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.	Go to Configuration Parameters > Code Generation > Interface and clear the Classic call interface option.
Verify 'Single output/update function' setting	The model is configured to generate code in separate <i>model_output</i> and <i>model_update</i> functions, rather than a <i>model_step</i> function that combines the two.	Go to Configuration Parameters > Code Generation > Interface and select Single output/update function .
Verify 'Terminate function required' setting	The model is configured to generate a <i>model_terminate</i> function, potentially containing model termination code to be executed during system shutdown. This is not supported for code inspection.	Go to Configuration Parameters > Code Generation > Interface and clear the Terminate function required option.
Verify 'Generate reusable code' setting	The model is not configured to generate reusable, multi-instance code that is reentrant. This option is applicable only to the top model in a model hierarchy.	Go to Configuration Parameters > Code Generation > Interface and select Generate reusable code .
Verify 'MAT-file logging' setting	The model is configured to log execution data to a MAT-file. This is not supported for code inspection.	Go to Configuration Parameters > Code Generation > Interface and clear the MAT-file logging option.
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.	Go to Configuration Parameters > Code Generation > Interface and clear the Support: non-finite numbers option.
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.	Go to Configuration Parameters > Code Generation > Interface and clear the Support: absolute time option.

Subcheck	Condition	Recommended Action
Verify 'Suppress error status in real-time model data structure' setting	The model is configured to include an error status field in a generated <code>rtModel</code> data structure. The <code>rtModel</code> data structure is not supported for code inspection.	Go to Configuration Parameters > Code Generation > Interface and select Suppress error status in real-time model data structure .
Verify 'IncludeERT-FirstTime' setting	The model is configured to include the <code>firstTime</code> argument in the generated <code>model_initialize</code> function. This is not supported for code inspection.	In the MATLAB Command Window, set the model parameter <code>IncludeERTFirstTime</code> to off. For example, <code>set_param(gcs, 'IncludeERTFirstTime', 'off')</code> .
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 <code>model_step</code> function. This is not supported for code inspection. This parameter is applicable only to the top model in a model hierarchy.	Go to Configuration Parameters > Code Generation > Interface and set Pass root-level I/O as to Individual arguments.
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.	Go to Configuration Parameters > Code Generation > SIL and PIL Verification and 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.	Go to Configuration Parameters > Code Generation > SIL and PIL Verification and clear the Measure function execution times option.
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.	Go to Configuration Parameters > Code Generation > Interface and 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.	Go to Configuration Parameters > Data Import/Export and clear the Initial state option.

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/Output 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 Output blocks. This potentially allows automatic insertion of hidden signal copy blocks at the model inports and outports, which is not supported for code inspection.	Go to Configuration Parameters > Diagnostics > Model Referencing and set Invalid root Inport/Output 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Connectivity and 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Data Validity and 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.	Go to Configuration Parameters > Diagnostics > Connectivity and 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.	Go to Configuration Parameters > Hardware Implementation and 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.	Go to Configuration Parameters > Hardware Implementation and, under Emulation hardware (code generation only) , select None .

See Also

Simulink Configuration Parameter Constraints

Check model reference settings

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

Description

This check verifies that model reference settings are compatible with code inspection.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Total number of instances allowed per top model' setting	The model is configured such that it can be referenced at most once in a model reference hierarchy (versus multiple or zero times). This might cause the model interface to fail validation.	Go to Configuration Parameters > Model Referencing and set Total number of instances allowed per top model to Multiple or Zero .

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 <code>AdvancedOptControl</code> to <code>-SLCI</code> . For example, <code>set_param(gcs, 'AdvancedOptControl', '-SLCI')</code> .
Verify 'Implement logic signals as Boolean data (vs. double)' setting	The model is configured to implement logic signals with the <code>double</code> data type, rather than with the more memory-efficient <code>boolean</code> data type.	Go to Configuration Parameters > Optimization and select Implement logic signals as Boolean data (vs. double) .
Verify 'Inline parameters' setting	The model is configured to use symbolic names (instead of inline numerical values) for tunable model parameters in generated code.	Go to Configuration Parameters > Optimization > Signals and Parameters and select Inline parameters .
Verify 'Optimize initialization code for model reference' setting	The model is configured to generate initialization code for all blocks that have states, without an optimization that can produce more efficient code for referenced models.	Go to Configuration Parameters > Optimization and select Optimize initialization code for model reference .

Subcheck	Condition	Recommended Action
Verify 'Inline invariant signals' setting	The model is configured to use symbolic names (instead of inline numerical values) for invariant signals in generated code.	Go to Configuration Parameters > Optimization > Signals and Parameters and select Inline invariant signals .
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.	Go to Configuration Parameters > Optimization and 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.	Go to Configuration Parameters > Optimization and clear the Remove code from floating-point to integer conversions with saturation that maps NaN to zero option.
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.	Go to Configuration Parameters > Optimization and clear the Remove code that protects against division arithmetic exceptions option.

Subcheck	Condition	Recommended Action
Verify 'Pack Boolean data into bitfields' setting	The model is configured to store Boolean signals as one-bit bitfields.	Go to Configuration Parameters > Optimization > Signals and Parameters and clear the Pack Boolean data into bitfields option.
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.	Go to Configuration Parameters > Optimization > Signals and Parameters and clear the Simplify array indexing option.

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.	Go to Configuration Parameters > Solver and set Type to Fixed-step .
Verify 'Solver' setting	The model is configured with a solver other than a fixed-step discrete solver.	Go to Configuration Parameters > Solver and set Solver to discrete (no continuous states) (equivalent to <code>FixedStepDiscrete</code> 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 any 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 4-17

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 <code>ert.tlc</code> or an ERT-derived target.	Go to Configuration Parameters > Code Generation and set System target file to <code>ert.tlc</code> 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.	Go to Configuration Parameters > Code Generation > Interface , 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 4-17

Check model arguments

Check that the model does not have parameter arguments.

Description

This check verifies that no model arguments are defined for this model.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check model arguments	Model arguments are specified for referencing this model. Model arguments are not supported for code inspection.	Remove the model arguments. Open Model Explorer, go to the Model Hierarchy pane, select the Model Workspace of the model, and in the Model arguments (for referencing this model) field, remove the specified arguments.

See Also

“Other Modelwide Attribute Constraints” on page 4-17

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 option **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 option 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.	Go to Configuration Parameters > Optimization > Signals and Parameters and select Enable local block outputs .

See Also

“Other Modelwide Attribute Constraints” on page 4-17

Check for unsupported blocks

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

Description

This check updates the model diagram and reports any 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	<p>One or more blocks in the model are not supported for code inspection.</p> <hr/> <p>Note Supported blocks are listed in “Supported Blocks — By Category” on page 5-30, and also can be viewed in the <code>slcilib</code> block library.</p> <hr/>	<p>Possible actions include:</p> <ul style="list-style-type: none"> • 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

Chapter 5, “Block Constraints”

Check for tunable workspace variables

Check for tunable workspace variables referenced by the model.

Description

This check updates the model diagram and reports any tunable workspace variables referenced by the model.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check for tunable workspace variables referenced by the model	One or more workspace variables referenced by the model are tunable. This requires use of storage classes, which are not supported for code inspection.	Modify workspace variables or modify the model so that the model no longer references tunable workspace variables.

See Also

“Other Modelwide Attribute Constraints” on page 4-17

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 any 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 4-17

Check for usage of global data stores

Check for usage of global data store memory that might impact compatibility with Simulink Code Inspector.

Description

This check updates the model diagram and reports any externally-defined signal objects that are referenced as global data stores by Data Store Read or Data Store Write blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify global data store usage	An externally-defined signal object is referenced as a global data store by a Data Store Read or Data Store Write block. This might trigger creation of a hidden Data Store Memory block at the root level of the model, which is not supported for code inspection.	Possible actions include: <ul style="list-style-type: none"> • If possible, avoid use of externally defined signal objects that are referenced as global data stores by Data Store Read or Data Store Write blocks. • Move the affected Data Store Read or Data Store Write blocks into Model blocks.

See Also

“Other Modelwide Attribute Constraints” on page 4-17

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 any 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 4-17

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 any fixed-point instrumentation incompatibilities.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify usage of fixed-point instrumentation	The model configuration option 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 4-17

Check for root Output blocks being conditionally assigned

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

Description

This check updates the model diagram and verifies that root outputs of referenced models are not connected to conditionally executed subsystems, either directly or through a nonvirtual Bus Creator, Bus Selector, or Signal Conversion block.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify that root Outputs are not assigned conditionally	<p>A root output of a referenced model is connected to a conditionally executed subsystem. Code inspection is not supported for submodels for which root outputs are assigned by blocks inside conditionally executed subsystems. This applies equally to:</p> <ul style="list-style-type: none"> • Direct connections to conditionally executed subsystems • Connections through a nonvirtual Bus Creator, Bus Selector, or Signal Conversion block, because those blocks propagate initial conditions 	<p>This check only applies to referenced models. If this model will be the top model in the hierarchy, go to Configuration Parameters > Model Referencing and set Total number of instances allowed per top model to Zero, which will suppress the check. Otherwise, modify the model so that the root outputs are not connected to conditionally executed subsystems, either directly or through a nonvirtual Bus Creator, Bus Selector, or Signal Conversion block.</p>

See Also

“Other Modelwide Attribute Constraints” on page 4-17

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 any incompatibilities it finds in Sources blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Inport blocks	The block cannot specify variable-dimension signals. Block parameter Variable-size signal (VarSizeSig) is set to Yes.	Set Variable-size signal to No.
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or outport is complex. Block inport or outport is multidimensional (not a scalar or a vector). Block inport or outport uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
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.	Correct the Constant value setting.
	Block option Interpret vector parameters as 1-D (VectorParams1D) is cleared (set to off).	Select Interpret vector parameters as 1-D .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to Auto. 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
Check Ground blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to Auto. Block has constant (Inf) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

See Also

Chapter 5, “Block Constraints”

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 any incompatibilities it finds in Signal Routing blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Bus Creator blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
<p>Check Bus Selector blocks</p>	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to Auto. Block has constant (Inf) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
Check Bus Assignment blocks	The block is operating on a nonvirtual bus.	Modify the model such that the block operates on a virtual bus. This action simplifies bus processing to promote traceability and readability of generated code.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
Check Data Store Memory blocks	The block state does not have storage class <code>Auto</code> . Values other than <code>Auto</code> 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 <code>Auto</code> . 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 <code>Auto</code> . If the block state name does resolve to a signal object, set the <code>CoderInfo.StorageClass</code> property of the signal object to <code>Auto</code> .
	Block parameter Initial value (<code>InitialValue</code>) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (<code>:</code>) operator.	Correct the Initial value setting.
	Block parameter Signal type (<code>SignalType</code>) is set to complex. Complex values are not supported for code inspection.	Set Signal type to <code>auto</code> or <code>real</code> .
	Block option Interpret vector parameters as 1-D (<code>VectorParams1D</code>) is cleared (set to off).	Select Interpret vector parameters as 1-D .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	
Check Data Store Read blocks	The block cannot specify elements. Block parameter Specify element(s) to select (<code>DataStoreElements</code>) is set to a nonempty string.	Clear element selections from the Element Selection tab of the block dialog box.
Note Data Store Read and Data Store Write blocks cannot reference externally-defined signal objects as global data stores. For more information, see “Check for usage of global data stores” on page 6-28.	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
<p>Check Data Store Write blocks</p>	<p>The block cannot specify elements. Block parameter Specify element(s) to select (<code>DataStoreElements</code>) is set to a nonempty string.</p>	<p>Clear element selections from the Element Selection tab of the block dialog box.</p>
<p>Note Data Store Read and Data Store Write blocks cannot reference externally-defined signal objects as global data stores. For more information, see “Check for usage of global data stores” on page 6-28.</p>	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
Check From blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	Correct the listed block inport or outport.
Check Goto blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
<p>Check Switch blocks</p>	<p>The first and third input ports and the output port do not have the same data type.</p>	<p>Modify the data ports to have the same data type. Consider selecting the block option Require all data port inputs to have the same data type.</p>
	<p>Block parameter Integer rounding mode (<code>RndMeth</code>) is set to an unsupported value.</p>	<p>Set Integer rounding mode to <code>Zero</code> or <code>Floor</code>.</p>
	<p>Block option Allow different data input sizes (<code>AllowDiffInputSizes</code>) is selected.</p>	<p>Clear the Allow different data input sizes option.</p>
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an output has been testpointed. 	
Check Multiport Switch blocks	Data input and output ports do not all have the same data type.	Modify the data ports to have the same data type. Consider selecting the block option Require all data 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 (<i>RndMeth</i>) is set to an unsupported value.	Set Integer rounding mode to <i>Zero</i> or <i>Floor</i> .
	Block option Allow different data input sizes (<i>AllowDiffInputSizes</i>) is selected.	Clear the Allow different data input sizes option.
	Block parameter Data port for default case (<i>DataPortForDefault</i>) is not set to <i>Last data port</i> .	Set Data port for default case to <i>Last data port</i> .

Subcheck	Condition	Recommended Action
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	<p>Correct the listed block inport or outport.</p>
<p>Check Mux blocks</p>	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. 	<p>Correct the listed block inport or outport.</p>

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an output has been testpointed. 	
Check Demux blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or output is not of data type <i>double</i>, <i>single</i>, <i>int8</i>, <i>uint8</i>, <i>int16</i>, <i>uint16</i>, <i>int32</i>, <i>uint32</i>, or <i>boolean</i>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an output has been testpointed. 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
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) .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	Correct the listed block inport or output.

See Also

Chapter 5, “Block Constraints”

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 any incompatibilities it finds in Math Operations blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Absolute blocks	Input and output ports do not all 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 or Floor.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
Check Gain blocks	Input and output ports do not all 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.	Correct 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) .	Set Multiplication to Element-wise(K.*u) .
	Block parameter Integer rounding mode (RndMeth) is set to an unsupported value.	Set Integer rounding mode to Zero or Floor .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an outport has been testpointed. 	
Check Math blocks	Input and output ports do not all have the same data type	Modify the port data types to match.
	Function (Operator) is set to an unsupported value: <i>conj</i> , <i>transpose</i> , or <i>hermitian</i> .	Set Function to one of the following values: <i>exp</i> , <i>log</i> , 10^u , <i>log10</i> , <i>magnitude^2</i> , <i>square</i> , <i>pow</i> , <i>reciprocal</i> , <i>hypot</i> , <i>rem</i> , or <i>mod</i> .
	Block parameter Integer rounding mode (<i>RndMeth</i>) is set to an unsupported value.	Set Integer rounding mode to <i>Zero</i> or <i>Floor</i> .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or outport is not of data type <i>double</i>, <i>single</i>, <i>int8</i>, <i>uint8</i>, <i>int16</i>, <i>uint16</i>, <i>int32</i>, <i>uint32</i>, or <i>boolean</i>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block inport or output uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
Check Product blocks	Input and output ports do not all have the same data type.	Modify the port data types to match.
	Block parameter Number of inputs (inputs) is not set to 2, **, /*, or */.	Set Number of inputs to 2, **, /*, or */.
	Block parameter Multiplication (Multiplication) is not set to <code>Element-wise(.*)</code> .	Set Multiplication to <code>Element-wise(.*)</code> .
	Block parameter Integer rounding mode (RndMeth) is set to an unsupported value.	Set Integer rounding mode to <code>Zero</code> or <code>Floor</code> .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an outport has been testpointed. 	
Check Sum blocks	Input and output ports do not all have the same data type.	Modify the port data types to match.
	Block parameter Accumulator data type (<i>AccumDataTypeStr</i>) 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 <i>Inherit: Same as first input</i> .
	Block parameter Integer rounding mode (<i>RndMeth</i>) is set to an unsupported value.	Set Integer rounding mode to <i>Zero</i> or <i>Floor</i> .
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <i>double</i>, <i>single</i>, <i>int8</i>, <i>uint8</i>, <i>int16</i>, <i>uint16</i>, <i>int32</i>, <i>uint32</i>, or <i>boolean</i>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
Check Trigonometry blocks	Block parameter Function (Operator) is set to <code>cos + jsin</code> (complex exponential of the input) or <code>sincos</code> .	Set Function to a value other than <code>cos + jsin</code> or <code>sincos</code> .
	Block parameter Approximation method (ApproximationMethod) is not set to <code>None</code> .	Set Approximation method to <code>None</code> .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or outport is complex. Block inport or outport is multidimensional (not a scalar or a vector). Block inport or outport uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
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: <code>double</code> , <code>single</code> , <code>int8</code> , <code>uint8</code> , <code>int16</code> , <code>uint16</code> , <code>int32</code> , or <code>uint32</code> .
	Input and output ports do not all 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 or Floor.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
Check Rounding Function blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to Auto. Block has constant (Inf) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

See Also

Chapter 5, “Block Constraints”

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 any 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 or Floor.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or outport is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or outport is complex. Block inport or outport is multidimensional (not a scalar or a vector). Block inport or outport uses frame-based signals. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
<p>Check Data Type Duplicate blocks</p>	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
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: <ul style="list-style-type: none"> • Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to Auto. • Block has constant (Inf) sample time and an outport has been testpointed. 	Correct the listed block inport or outport.

See Also

Chapter 5, “Block Constraints”

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 any 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 output is not either an enumerated type with default value 0, or boolean.	Modify the output data type to be either an enumerated type with default value 0, or boolean.
	Block input ports do not all 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: <code>isInf</code> , <code>isNaN</code> , or <code>isFinite</code> .	Set Relational operator to a supported value: <code><=</code> , <code>==</code> , <code>>=</code> , <code>~=</code> , <code><</code> , or <code>></code> .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). 	Correct the listed block inport or output.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	
Check Logic blocks	Logical Operator block output is not Boolean.	Modify the data type of the output to <code>boolean</code> .
	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 all have the same data type.	Configure the input ports to have the same data type.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none">• Block has constant (Inf) sample time and an outport has been testpointed.	

See Also

Chapter 5, “Block Constraints”

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 any 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 all have the same data type.	Modify the input and output ports to all 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 option Begin index search using previous index result (BeginIndexSearchUsingPreviousIndexResult) is selected (set to on).	Clear the Begin index search using previous index result .

Subcheck	Condition	Recommended Action
	Block option Support tunable table size in code generation (SupportTunableTableSize) is selected (set to on).	Clear the Support tunable table size in code generation option.
	Block option 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 option.
	Block option Saturate on integer overflow (SaturateOnIntegerOverflow) is selected (set to on).	Clear the Saturate on integer overflow option.
	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.	Correct the Table data setting.
	Block parameter Breakpoints 1 (BreakpointsForDimension1) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (:) operator.	Correct 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.	Correct the Breakpoints 2 setting.

Subcheck	Condition	Recommended Action
	Block parameter Breakpoints 1 (BreakpointsForDimension1DataTypeStr) is not using the same data type as the block input.	Modify the data types to match.
	Block parameter Breakpoints 2 (BreakpointsForDimension2DataTypeStr) 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.
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none">• Block inport or outport uses frame-based signals.• Block output signal storage class is not set to <i>Auto</i>.• Block has constant (<i>Inf</i>) sample time and an outport has been testpointed.	

See Also

Chapter 5, “Block Constraints”

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 any 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 all arguments are scalars or vectors of fixed dimension.
	The Legacy Code Tool S-function specifies a <code>InitializeConditionsFcnSpec</code> , <code>StartFcnSpec</code> , or <code>TerminateFcnSpec</code> , rather than an <code>OutputFcnSpec</code> .	Modify the S-function configuration to specify an <code>OutputFcnSpec</code> .
	The S-function has more than one <code>dwork</code> .	Modify the S-function configuration to specify one <code>dwork</code> .
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<p>other buses) meet the data type constraint.</p> <ul style="list-style-type: none"> • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <i>Auto</i>. • Block has constant (<i>Inf</i>) sample time and an output has been testpointed. 	

See Also

Chapter 5, “Block Constraints”

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 any 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 <code>boolean</code> .	Correct the signal data type.
	Block option Show output port (<code>ShowOutputPort</code>) is selected.	Clear the option Show output port .
	The Enable Port block is located at the root level of the model.	Remove or relocate the Enable Port block.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or outport is complex. Block inport or outport is multidimensional (not a scalar or a vector). Block inport or outport uses frame-based signals. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	
<p>Check Model Reference blocks</p>	<p>The Model block cannot have variants. Block option Enable variants (Variant) is selected (set to on).</p>	<p>Clear the Enable variants option.</p>
<p>Note Referenced models cannot accept model arguments. For more information, see “Check model arguments” on page 6-23.</p>	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. Block has constant (<code>Inf</code>) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

Subcheck	Condition	Recommended Action
Check Subsystem blocks	The subsystem is not a virtual subsystem or an inlined atomic or enabled subsystem.	If possible, reconfigure the subsystem to be either virtual (clear the Subsystem block option Treat as atomic unit), or an inlined atomic or enabled subsystem. Alternatively, wrap the subsystem in a Model block, or explore other implementation options.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> • Block inport or output is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or output is complex. • Block inport or output is multidimensional (not a scalar or a vector). • Block inport or output uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	Correct the listed block inport or output.

See Also

Chapter 5, “Block Constraints”

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 any incompatibilities it finds in Discontinuities blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Saturate blocks	Input and output ports do not all 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.	Correct 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.	Correct 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 or Floor.

Subcheck	Condition	Recommended Action
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> • Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. • Block inport or outport is complex. • Block inport or outport is multidimensional (not a scalar or a vector). • Block inport or outport uses frame-based signals. • Block output signal storage class is not set to <code>Auto</code>. • Block has constant (<code>Inf</code>) sample time and an outport has been testpointed. 	<p>Correct the listed block inport or outport.</p>

See Also

Chapter 5, “Block Constraints”

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 any incompatibilities it finds in Sinks blocks.

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Check Output blocks	The block cannot specify variable-dimension signals. Block parameter Variable-size signal (VarSizeSig) is set to Yes.	Set Variable-size signal to No.
	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or outport is not of data type <code>double</code>, <code>single</code>, <code>int8</code>, <code>uint8</code>, <code>int16</code>, <code>uint16</code>, <code>int32</code>, <code>uint32</code>, or <code>boolean</code>, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or outport is complex. Block inport or outport is multidimensional (not a scalar or a vector). Block inport or outport uses frame-based signals. Block output signal storage class is not set to <code>Auto</code>. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none"> Block has constant (Inf) sample time and an output has been testpointed. 	
Check Terminator blocks	<p>Violates a constraint that applies to all blocks:</p> <ul style="list-style-type: none"> Block inport or output is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. Block inport or output is complex. Block inport or output is multidimensional (not a scalar or a vector). Block inport or output uses frame-based signals. Block output signal storage class is not set to Auto. Block has constant (Inf) sample time and an output has been testpointed. 	<p>Correct the listed block inport or output.</p>

See Also

Chapter 5, “Block Constraints”

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 any 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 <code>CoderInfo.StorageClass</code> 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.	Correct the Initial conditions setting.
	Violates a constraint that applies to all blocks: <ul style="list-style-type: none"> Block inport or outport is not of data type double, single, int8, uint8, int16, uint16, int32, uint32, or boolean, or if the block supports buses, a bus for which the elements (potentially including other buses) meet the data type constraint. 	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	<ul style="list-style-type: none">• Block inport or output is complex.• Block inport or output is multidimensional (not a scalar or a vector).• Block inport or output uses frame-based signals.• Block output signal storage class is not set to <code>Auto</code>.• Block has constant (<code>Inf</code>) sample time and an output has been testpointed.	

See Also

Chapter 5, “Block Constraints”

Check usage of root Output blocks

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

Description

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

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify sample times	One or more root Output blocks specify a constant (Inf) sample time. This will cause the model functions to fail validation, because the root output assignment is moved to the model initialize function.	Set the sample times of the root Output blocks to explicit, nonconstant sample times.
Verify root Outputs pass buses to parent models as structures	One or more root Output 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 Output block dialog box and select the option Output as nonvirtual bus in parent model (BusOutputAsStruct).

See Also

“Other Modelwide Attribute Constraints” on page 4-17

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 any 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 no nonvirtual block operates on a virtual bus, and no Unit Delay block operates on a bus. This action simplifies bus processing to promote traceability and readability of generated code.

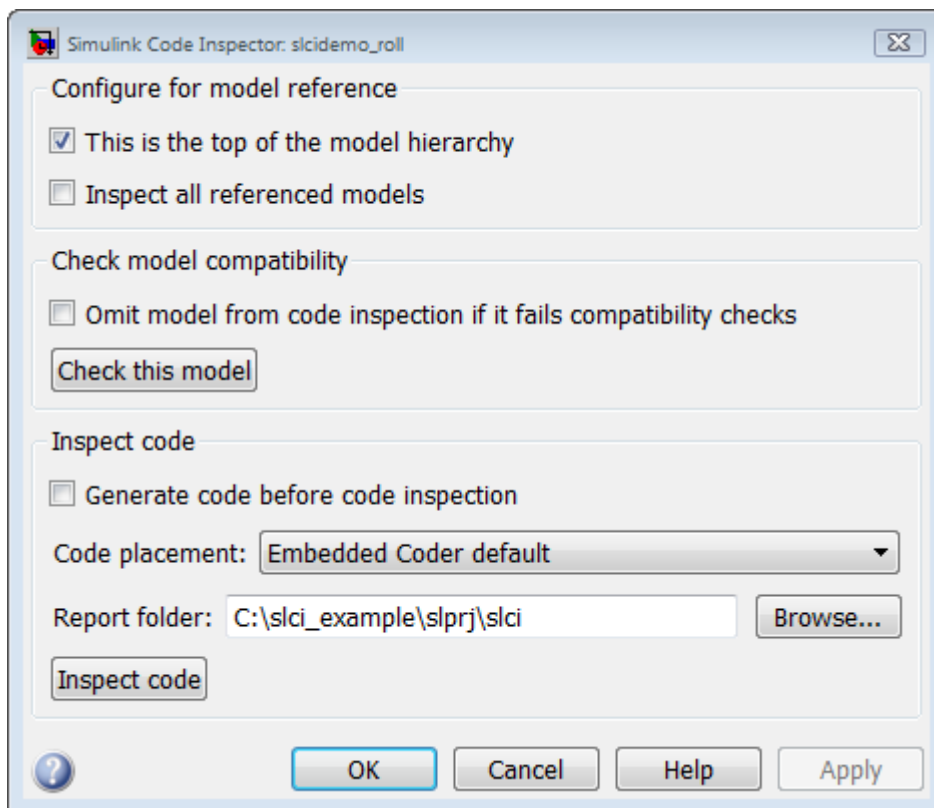
See Also

“Other Modelwide Attribute Constraints” on page 4-17

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” on page 7-4

“This is the top of the model hierarchy” on page 7-5

“Inspect all referenced models” on page 7-6

“Omit model from code inspection if it fails compatibility check” on page 7-7

“Generate code before code inspection” on page 7-8

“Code placement” on page 7-9

“Code folder” on page 7-10

“Report folder” on page 7-11

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

- “Code Inspection”
- “Model Compatibility Checking”

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

“Code Inspection”

Inspect all referenced models

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

Settings

Default: off



On

Model compatibility checking and code inspection are performed for all 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

- “Code Inspection”
- “Model Compatibility Checking”

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

- “Code Inspection”
- “Model Compatibility Checking”

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

“Code Inspection”

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

“Code Inspection”

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

“Code Inspection”

Report folder

Specify a report folder for code inspection.

Settings

Default: Subfolder `s1prj/s1ci` 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 `s1ci.Configuration.setReportFolder`.

See Also

“Code Inspection”