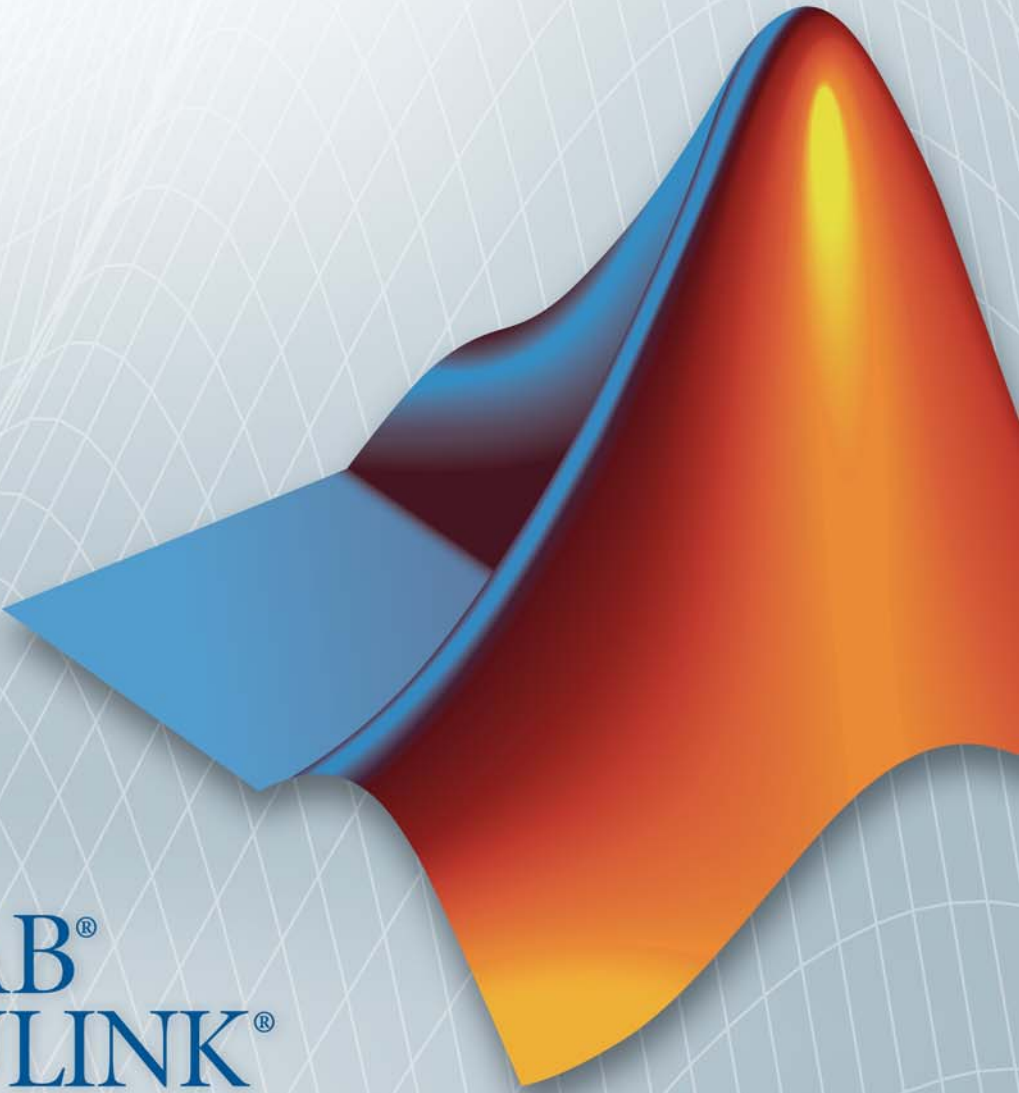
Simulink® Code Inspector™

Reference

R2011b



MATLAB® & SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® Code Inspector™ Reference

Revision History

September 2011	Online only	New for Version 1.0 (Release 2011b)

Contents

1 Function Reference

Code Inspection 1-2

Model Compatibility Checking 1-4

2 Class Reference

Code Inspection 2-1

3 Functions — Alphabetical List

4 Model Configuration Constraints Reference

About Model Configuration Constraints Reference . . . 4-2

Model Configuration Constraints 4-4
  Simulink Configuration Parameters 4-4
  Other Modelwide Attributes 4-17

5 Block Constraints Reference

About Block Constraints Reference **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**
From **5-13**
Gain **5-13**
Goto **5-14**
Inport **5-14**
Logical Operator **5-14**
Math Function **5-15**
MinMax **5-16**
Model **5-16**
Multiport Switch **5-17**
Mux **5-17**
Outport **5-18**
Product **5-18**
Relational Operator **5-19**
Saturation **5-19**
Selector **5-20**
S-Function **5-20**
Signal Conversion **5-21**
Subsystem **5-22**
Sum, Add, Subtract **5-22**
Switch **5-23**
Terminator **5-23**
Trigonometric Function **5-24**
Unit Delay **5-24**

Supported Blocks — By Category **5-25**

Commonly Used Blocks **5-25**
Discontinuity Blocks **5-26**
Discrete Blocks **5-26**
Logic and Bit Operation Blocks **5-26**
Math Operation Blocks **5-26**
Port & Subsystem Blocks **5-27**
Signal Attribute Blocks **5-27**
Signal Routing Blocks **5-27**
Sink Blocks **5-28**
Source Blocks **5-28**
User-Defined Functions **5-28**

6 **Model Advisor Checks**

Simulink® Code Inspector Checks **6-2**
Simulink® Code Inspector Checks Overview **6-4**
Check code generation settings **6-5**
Check data import/export settings **6-9**
Check diagnostic settings **6-10**
Check hardware implementation settings **6-12**
Check model reference settings **6-14**
Check optimization settings **6-15**
Check solver settings **6-18**
Check for unconnected objects in the model **6-19**
Check system target file setting **6-20**
Check function specification setting **6-21**
Check model arguments **6-22**
Check for unsupported blocks **6-23**
Check for tunable workspace variables **6-24**
Check for sample times in the model **6-25**
Check for usage of global data stores **6-26**
Check usage of Sources blocks **6-27**
Check usage of Signal Routing blocks **6-30**
Check usage of Math Operations blocks **6-42**
Check usage of Signal Attributes blocks **6-49**
Check usage of Logical and Bit Operations blocks **6-52**
Check usage of User-Defined Function blocks **6-55**
Check usage of Ports and Subsystems blocks **6-57**
Check usage of Discontinuities blocks **6-60**
Check usage of Sinks blocks **6-62**

Check usage of Discrete blocks **6-64**
Check usage of root Outport blocks **6-66**
Check usage of buses **6-67**

7 **Simulink® Code Inspector Dialog Box Parameters**

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

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

setTopModel (slci.Configuration)	Specify whether model being configured for code inspection is top model
slci.Configuration	Create code inspection object

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

Class Reference

Code Inspection

slci.Configuration	Control code inspection and compatibility checking for a model

3

Functions — Alphabetical List

slci.Configuration.checkCompatibility

Purpose

Check model compatibility with code inspection

Syntax

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

Description

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

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

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

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

Tips

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

Input Arguments

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

Name-Value Pair Arguments

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-value pair arguments in any order as `Name1,Value1, ,NameN,ValueN`.

`DisplayResults`

Specify whether to display results of the compatibility checks.

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

Default: `` `Summary' ``

Output Arguments

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

Examples

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

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

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

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

Alternatives

Open the Simulink Code Inspector dialog box from **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”

slci.Configuration.getCodeFolder

Purpose

Return code folder for code inspection

Syntax

folder = getCodeFolder(*cfgObj*)

Description

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

Input Arguments

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

Output Arguments

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

Examples

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

Alternatives

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

See Also

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

How To

- “Code Inspection”

Purpose

Return code placement for code inspection

Syntax

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

Description

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

Input Arguments

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

Output Arguments

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

Examples

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

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"

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

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

Output Arguments

value	True if model compatibility checking and code inspection should be performed for 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`

How To

- “Code Inspection”
- “Model Compatibility Checking”

slci.Configuration.getGenerateCode

Purpose

Return code generation option for code inspection

Syntax

`value = getGenerateCode(cfgObj)`

Description

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

Input Arguments

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

Output Arguments

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

Examples

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

Alternatives

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

See Also

`slci.Configuration.setGenerateCode`

How To

- “Code Inspection”

slci.Configuration.getReportFolder

Purpose

Return report folder for code inspection

Syntax

`folder = getReportFolder(cfgObj)`

Description

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

Input Arguments

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

Output Arguments

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

Examples

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

Alternatives

Open the Simulink Code Inspector dialog box from **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

`value = getTerminateOnIncompatibility(cfgObj)`

Description

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

Input Arguments

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

Output Arguments

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

Examples

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

Alternatives

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

See Also

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

How To

- “Code Inspection”

- “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

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

Output Arguments

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

Examples

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

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

Alternatives

Open the Simulink Code Inspector dialog box from **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(modelName);`.

Name-Value Pair Arguments

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-value pair arguments in any order as `Name1,Value1, ,NameN,ValueN`.

`DisplayResults`

Specify whether to display inspection results.

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

Default: `` `Summary' ``

Output Arguments

`results`

Structure containing the following fields:

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

Examples

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

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

Alternatives

Open the Simulink Code Inspector dialog box from **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

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

Examples

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

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

Alternatives

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

See Also `slci.Configuration.setCodePlacement` | `slci.Configuration.getCodeFolder`

How To

- "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 =
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”

Purpose

Specify model reference handling for model compatibility checking or code inspection

Syntax

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

Description

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

Input Arguments

`cfgObj`	Handle to a Simulink Code Inspector configuration object previously returned by *`cfgObj`*` = slci.Configuration(`*`modelName`*`);`.
`followModelLinks`	True if model compatibility checking and code inspection should be performed for 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 code inspection.

See Also

`slci.Configuration.getFollowModelLinks`

How To

- “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

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

Examples

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

Alternatives

Open the Simulink Code Inspector dialog box from **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"

slci.Configuration.setReportFolder

Purpose

Specify report folder for code inspection

Syntax

`setReportFolder(cfgObj, folder)`

Description

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

Input Arguments

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

Examples

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

Alternatives

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

See Also

`slci.Configuration.getReportFolder`

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

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

Examples

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

Alternatives

Open the Simulink Code Inspector dialog box from **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

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

Examples

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

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

Alternatives

Open the Simulink Code Inspector dialog box from **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 a model

Description

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

Construction

slci.Configuration	Create code inspection object

Methods

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

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

Copy Semantics

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

Examples

The Simulink Code Inspector 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

`cfgObj = slci.Configuration(modelName)`

Description

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

Input Arguments

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

Output Arguments

`cdgObj`	Handle to code inspection object.

Examples

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

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

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

Alternatives

Open the Simulink Code Inspector dialog box from **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"

Model Configuration Constraints Reference

- "About Model Configuration Constraints Reference" on page 4-2
- "Model Configuration Constraints" on page 4-4

About Model Configuration Constraints Reference

Simulink Code Inspector requires that you set a subset of Simulink® configuration parameters and other model attributes to specific values. "Simulink Configuration Parameters" on page 4-4 presents required settings for Configuration Parameters Dialog Box parameters and their equivalent command-line parameters. "Other Modelwide Attributes" 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**

Model Configuration Constraints

In this section...

"Simulink Configuration Parameters" on page 4-4
"Other Modelwide Attributes" on page 4-17

Simulink Configuration Parameters

- "Solver" on page 4-5
- "Data Import/Export" on page 4-5
- "Optimization" on page 4-6
- "Optimization: Signals and Parameters" on page 4-7
- "Diagnostics: Data Validity" on page 4-8
- "Diagnostics: Connectivity" on page 4-9
- "Diagnostics: Model Referencing" on page 4-10
- "Hardware Implementation" on page 4-10
- "Model Referencing" on page 4-12
- "Code Generation: General" on page 4-12
- "Code Generation: Comments" on page 4-12
- "Code Generation: Symbols" on page 4-13
- "Code Generation: Custom Code" on page 4-13
- "Code Generation: Interface" on page 4-14
- "Code Generation: SIL and PIL Verification" on page 4-15
- "Code Generation: Code Style" on page 4-16
- "Code Generation: Data Type Replacement" on page 4-16
- "Code Generation: Not in GUI" on page 4-17

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**
Input (`LoadExternalInput`)	Must be cleared (set to `off`).	Nonfatal	**Check data import/export settings > Verify 'Input' setting**
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 data (vs. double) (`BooleanDataType`)	Must be selected (set to `on`).	Nonfatal	**Check optimization settings > Verify 'Implement logic signals as Boolean data (vs. double)' setting**
Remove root level I/O zero initialization (`ZeroExternalMemory-AtStartup`)	Must be selected (equivalent to setting `ZeroExternalMemory-AtStartup` to `off`, not `on`, at the command line).	Nonfatal	**Check optimization settings > Verify 'Remove root level I/O zero initialization' setting**
Use memset to initialize floats and doubles to 0.0 (`InitFltsAndDblsTo-Zero`)	Must be cleared (equivalent to setting `InitFltsAndDblsToZero` to `on`, not `off`, at the command-line).	Nonfatal	**Check optimization settings > Verify 'Use memset to initialize floats and doubles to 0.0' setting**
Remove internal data zero initialization (`ZeroInternalMemory-AtStartup`)	Must be selected (equivalent to setting `ZeroInternalMemory-AtStartup` to `off`, not `on`, at the command line).	Nonfatal	**Check optimization settings > Verify 'Remove internal data zero initialization' 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**

Optimization Pane: General			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
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**

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**

Optimization Pane: Signals and Parameters			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Simplify array indexing (`StrengthReduction`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Simplify array indexing' setting**
Use memcpy for vector assignment (`EnableMemcpy`)	Must be cleared (set to `off`).	Nonfatal	**Check optimization settings > Verify 'Use memcpy for vector assignment' 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**
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**

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

Hardware Implementation

Hardware Implementation Pane			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
Number of bits: char (`ProdBitPerChar`)	Must be set to `8`.	Nonfatal	**Check hardware implementation settings > Verify 'char' setting**
Number of bits: short (`ProdBitPerShort`)	Must be set to `16`.	Nonfatal	**Check hardware implementation settings > Verify 'short' setting**
Number of bits: int (`ProdBitPerInt`)	Must be set to `32`.	Nonfatal	**Check hardware implementation settings > Verify 'int' setting**
Number of bits: long (`ProdBitPerLong`)	Must be set to `32`.	Nonfatal	**Check hardware implementation settings > Verify 'long' setting**

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.	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**
Target function library (`TargetFunction-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 ’Target function library’ 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**
GRT compatible call interface (`GRTInterface`)	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify ’GRT compatible 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**
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	Must be cleared (set to `off`).	Nonfatal	**Check code generation settings > Verify**

Code Generation Pane: SIL and PIL Verification			
Parameter	**Constraint**	**FATAL / Nonfatal**	**Compatibility Check**
(`CodeProfiling-Instrumentation`)			**'Instrument generated code for execution time measurement' 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 Attributes

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**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Model arguments	There must be no model arguments defined for the model.	Nonfatal	**Check model arguments**
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 ariables**
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**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
Root Outport block sample times	Root Outport blocks cannot specify a constant (`Inf`) sample time. This constraint prevents the root outport assignment from being moved to the model initialize function, which would cause the model functions to fail validation.	Nonfatal	**Check usage of root Outport blocks > Verify sample times**
Root Output block bus passing method	A root Outport block that passes a bus to a parent model must pass the bus as a structure. Otherwise, Simulink software might insert a hidden Signal Conversion block in the parent model, which is not supported for code inspection.	Nonfatal	**Check usage of root Outport blocks > Verify root Outports pass buses to parent models as structures**
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	Nonvirtual blocks cannot operate on a virtual bus, and Unit Delay blocks cannot operate on a virtual or nonvirtual bus. This constraint simplifies bus processing to promote	FATAL	**Check usage of buses > Verify that no blocks in the model operate on a virtual bus**

Attribute	Constraint	FATAL / Nonfatal	Compatibility Check
	traceability and readability of generated code.		

Block Constraints Reference

- "About Block Constraints Reference" on page 5-2
- "Block Constraints — Alphabetical List" on page 5-5
- "Supported Blocks — By Category" on page 5-25

About Block Constraints Reference

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 5-25 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

"From" on page 5-13

"Gain" on page 5-13

"Goto" on page 5-14

"Inport" on page 5-14

"Logical Operator" on page 5-14

"Math Function" on page 5-15

"MinMax" on page 5-16

"Model" on page 5-16

"Multiport Switch" on page 5-17

"Mux" on page 5-17

"Outport" on page 5-18

"Product" on page 5-18

"Relational Operator" on page 5-19

In this section...

"Saturation" on page 5-19
"Selector" on page 5-20
"S-Function" on page 5-20
"Signal Conversion" on page 5-21
"Subsystem" on page 5-22
"Sum, Add, Subtract" on page 5-22
"Switch" on page 5-23
"Terminator" on page 5-23
"Trigonometric Function" on page 5-24
"Unit Delay" on page 5-24

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 which the elements (potentially including other buses) meet the data type constraint.	FATAL	All block compatibility checks
	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	

Constraints that apply to all blocks			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	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	
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		

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		

Inport

Inport			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sources blocks > Check Inport blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify variable-dimension signals. **Variable-size signal** (`VarSizeSig`) must *not* be set to `Yes`.	Nonfatal	

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 port must be of the data type `boolean`.	FATAL	

Logical Operator			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
	Block must have at least two inports, except in the case of the NOT operator.	FATAL	
Block Parameters	No block-specific constraints		

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`, `mod`, or (for legacy models) `sqrt`. 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 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.

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**
	Control input port must have an integer data type and data input and output ports must all have the same data type.	Nonfatal	
	Block must have at least three inports.	FATAL	
Block Parameters	**Data port order** (`DataPortOrder`) must be set to `Zero-based contiguous` or `One-based contiguous` (not `Specify indices`).	Nonfatal	
	Integer rounding mode (`RndMeth`) must be set to `Zero` or `Floor`.	Nonfatal	

Mux

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

Outport

Outport			
	Constraint	**FATAL / Nonfatal**	**Compatibility Check**
Data Types	Constraints that apply to all blocks.		**Check usage of Sinks blocks > Check Outport blocks**
	No block-specific constraints		
Block Parameters	The block cannot specify variable-dimension signals. **Variable-size signal** (`VarSizeSig`) must *not* be set to `Yes`.	Nonfatal	

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	
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**
	Output port must be of the data type `boolean`.	FATAL	
Block Parameters	**Relational operator** (`Operator`) must be set to <=, ==, >=, ~=, <, or > (not `isInf`, `isNaN`, or `isFinite`).	FATAL	

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**	Nonfatal	

Saturation			
	Constraint	FATAL / Nonfatal	Compatibility Check
	limit rather than input ports (`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: • 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

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. Nonvirtual (atomic) subsystems are not supported.	FATAL	
	The block cannot have variants. **Variant** (`Variant`) must be set to `off`.	Nonfatal	

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**
	Block must have two inports.	FATAL	
	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 have the same data type.	Nonfatal	
Block Parameters	**Integer rounding mode** (`RndMeth`) must be set to `Zero` or `Floor`.	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 *not* be set to `cos + jsin` (complex exponential of the input).	Nonfatal	
	Approximation method (`ApproximationMethod`) must be set to `None`.	Nonfatal	

Unit Delay

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

Supported Blocks — By Category

In this section...

"Commonly Used Blocks" on page 5-25
"Discontinuity Blocks" on page 5-26
"Discrete Blocks" on page 5-26
"Logic and Bit Operation Blocks" on page 5-26
"Math Operation Blocks" on page 5-26
"Port & Subsystem Blocks" on page 5-27
"Signal Attribute Blocks" on page 5-27
"Signal Routing Blocks" on page 5-27
"Sink Blocks" on page 5-28
"Source Blocks" on page 5-28
"User-Defined Functions" on page 5-28

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 5-13
- "Inport" on page 5-14
- "Logical Operator" on page 5-14
- "Mux" on page 5-17
- "Outport" on page 5-18
- "Product" on page 5-18

- “Relational Operator” on page 5-19
- “Saturation” on page 5-19
- “Subsystem” on page 5-22
- “Sum, Add, Subtract” on page 5-22
- “Switch” on page 5-23
- “Terminator” on page 5-23
- “Unit Delay” on page 5-24

Discontinuity Blocks

- “Saturation” on page 5-19

Discrete Blocks

- “Unit Delay” on page 5-24

Logic and Bit Operation Blocks

- “Logical Operator” on page 5-14
- “Relational Operator” on page 5-19

Math Operation Blocks

- “Abs” on page 5-7
- “Gain” on page 5-13
- “Math Function” on page 5-15
- “MinMax” on page 5-16
- “Product” on page 5-18
- “Sum, Add, Subtract” on page 5-22
- “Trigonometric Function” on page 5-24

Port & Subsystem Blocks

- “Inport” on page 5-14
- “Model” on page 5-16
- “Outport” on page 5-18
- “Subsystem” on page 5-22

Signal Attribute Blocks

- “Data Type Conversion” on page 5-11
- “Data Type Duplicate” on page 5-12
- “Signal Conversion” on page 5-21

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 5-13
- “Goto” on page 5-14
- “Multiport Switch” on page 5-17
- “Mux” on page 5-17
- “Selector” on page 5-20
- “Switch” on page 5-23

Sink Blocks

- “Outport” on page 5-18
- “Terminator” on page 5-23

Source Blocks

- “Constant” on page 5-9
- “Inport” on page 5-14

User-Defined Functions

- “S-Function” on page 5-20

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 6-9
"Check diagnostic settings" on page 6-10
"Check hardware implementation settings" on page 6-12
"Check model reference settings" on page 6-14
"Check optimization settings" on page 6-15
"Check solver settings" on page 6-18
"Check for unconnected objects in the model" on page 6-19
"Check system target file setting" on page 6-20
"Check function specification setting" on page 6-21
"Check model arguments" on page 6-22
"Check for unsupported blocks" on page 6-23
"Check for tunable workspace variables" on page 6-24
"Check for sample times in the model" on page 6-25
"Check for usage of global data stores" on page 6-26
"Check usage of Sources blocks" on page 6-27
"Check usage of Signal Routing blocks" on page 6-30
"Check usage of Math Operations blocks" on page 6-42
"Check usage of Signal Attributes blocks" on page 6-49
"Check usage of Logical and Bit Operations blocks" on page 6-52
"Check usage of User-Defined Function blocks" on page 6-55
"Check usage of Ports and Subsystems blocks" on page 6-57
"Check usage of Discontinuities blocks" on page 6-60
"Check usage of Sinks blocks" on page 6-62

In this section...

"Check usage of Discrete blocks" on page 6-64

"Check usage of root Outport blocks" on page 6-66

"Check usage of buses" on page 6-67

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++ files rather than C files.	Go to Configuration Parameters **> Code Generation** and set **Language** to `C`.
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.
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**.

Subcheck	Condition	Recommended Action
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 ’Target function 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 **Target function library** to `C89/C90 (ANSI)` (equivalent to `ANSI_C` specified at the command line).
Verify ’GRT compatible call interface’ setting	The model is configured to generate model function calls compatible with the main program module of the GRT target. The GRT compatible call interface is not supported for code inspection.	Go to Configuration Parameters **> Code Generation > Interface** and clear the **GRT compatible call interface** option.
Verify ’Single output/update function’ setting	The model is configured to generate code in separate *`model`*`_output` and *`model`*`_update` functions, rather than a *`model`*`_step` function that combines the two.	Go to Configuration Parameters **> Code Generation > Interface** and select **Single output/update function**.

Subcheck	Condition	Recommended Action
Verify 'Terminate function required' setting	The model is configured to generate a *`model`*`_terminate` function, potentially containing model termination code to be executed during system shutdown. This is not supported for code inspection.	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.
Verify 'Suppress error status in real-time model data structure' setting	The model is configured to include an error status field in a generated `rtModel` data structure. The `rtModel` data structure is not supported for code inspection.	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 *`firstTime`* argument in the generated *`model`*`_initialize` function. This is not supported for code inspection.	In the MATLAB Command Window, set the model parameter `IncludeERTFirstTime` to `off`. For example, `set_param(gcs, 'IncludeERTFirstTime', 'off')`.

Subcheck	Condition	Recommended Action
Verify 'Pass root-level I/O as' setting	The model is configured to use packed structures, rather than individual arguments, to pass root-level model input and output values to the *`model`*`_step` function. This is not supported for code inspection. This parameter is applicable only to the top model in a model hierarchy.	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 'Instrument generated code for execution time measurement' 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

"Model Configuration Constraints" on page 4-4

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 'Input' setting	The model is configured to load data from a workspace, which is not compatible with code inspection.	Go to Configuration Parameters **>** **Data Import/Export** and clear the **Input** option.
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

"Model Configuration Constraints" on page 4-4

Check diagnostic settings

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

Description

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

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'Invalid root Inport/Outport block connection' setting	The model is not configured to generate an error if Simulink software detects invalid internal connections to the root-level Inport or Outport blocks. This potentially allows automatic insertion of hidden signal copy blocks at the model inports and outports, which is not supported for code inspection.	Go to Configuration Parameters **> Diagnostics > Model Referencing** and set **Invalid root Inport/Outport block connection** to `error`. If an error is generated, it identifies the locations at which you can manually insert Signal Conversion blocks to avoid the error and maintain traceability.
Verify 'Underspecified initialization detection' setting	The model is not configured to initialize block initial conditions using simplified behavior. The simplified behavior can improve the consistency of model results.	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

"Model Configuration Constraints" on page 4-4

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

"Model Configuration Constraints" on page 4-4

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

"Model Configuration Constraints" on page 4-4

Check optimization settings

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

Description

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

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify 'AdvancedOptControl' setting	The model is not configured to disable optimizations that are incompatible with Simulink Code Inspector.	In the MATLAB Command Window, set the model parameter `AdvancedOptControl` to `-SLCI`. For example, `set_param(gcs, 'AdvancedOptControl', '-SLCI')`.
Verify 'Implement logic signals as Boolean data (vs. double)' setting	The model is configured to implement logic signals with the `double` data type, rather than with the more memory-efficient `boolean` data type.	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 'Use memcpy for vector assignment' setting	The model is configured to optimize code generated for vector assignment by conditionally replacing `for` loops with `memcpy`, based on a threshold parameter.	Go to Configuration Parameters **> Optimization > Signals and Parameters** and clear the **Use memcpy for vector assignment** option.

Subcheck	Condition	Recommended Action
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**.
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 'Use memset to initialize floats and doubles to 0.0' setting	The model is configured to generate code that uses `memset` to initialize floating-point data to 0.0.	Go to Configuration Parameters **> Optimization** and clear the **Use memset to initialize floats and doubles to 0.0** option. (This is equivalent to `InitFltsAndDblsTo-Zero` being set to `on`, not `off`, at the command-line.)
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.

Subcheck	Condition	Recommended Action
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.
Verify 'Remove root level I/O zero initialization' setting	The model is configured to generate initialization code for all root-level inports and outports, without an optimization that can produce more efficient code for root-level inports and outports set to zero.	Go to Configuration Parameters **> Optimization** and select **Remove root level I/O zero initialization**. (This is equivalent to setting `ZeroExternalMemoryAtStartup` to `off`, not `on`, at the command-line.)
Verify 'Remove internal data zero initialization' setting	The model is configured to generate code that initializes internal work structures to zero.	Go to Configuration Parameters **> Optimization** and select **Remove internal data zero initialization**. (This is equivalent to setting `ZeroInternalMemoryAtStartup` to `off`, not `on`, at the command-line.)

See Also

"Model Configuration Constraints" on page 4-4

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 `FixedStepDiscrete` specified at the command line).

See Also

"Model Configuration Constraints" on page 4-4

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

“Model Configuration Constraints” on page 4-4

Check system target file setting

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

Description

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

Results and Recommended Actions

Subcheck	Condition	Recommended Action
Verify system target file setting	The system target file selected for the model is not `ert.tlc` or an ERT-derived target.	Go to Configuration Parameters **> Code Generation** and set **System target file** to `ert.tlc` or an ERT-derived target.

See Also

"Model Configuration Constraints" on page 4-4

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

"Model Configuration Constraints" on page 4-4

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

"Model Configuration Constraints" on page 4-4

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	One or more blocks in the model are not supported for code inspection. **Note** Supported blocks are listed in "Supported Blocks — By Category" on page 5-25, and also can be viewed in the `slcilib` block library.	Possible actions include: • Replace an unsupported block with a supported block. • Replace an unsupported block with an equivalent combination of supported blocks. • Replace an unsupported block with an S-Function block created using the Legacy Code Tool. • If one or more unsupported blocks cannot be removed, use referenced models to isolate the unsupported block(s), and/or use a partial verification work flow that omits the unsupported block(s).

See Also

Chapter 5, "Block Constraints Reference"

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

"Model Configuration Constraints" on page 4-4

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

"Model Configuration Constraints" on page 4-4

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: • 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

"Model Configuration Constraints" on page 4-4

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`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport 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: • 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. • 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 Reference”

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	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Bus Selector blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.

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: • 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. • 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.

Subcheck	Condition	Recommended Action
Check Data Store Memory blocks	The block state does not have storage class `Auto`. Values other than `Auto` require use of storage classes, which are not supported for code inspection.	Modify the block such that its code generation storage class is set to `Auto`. If the block state name does not resolve to a signal object, set **Storage Class** in the **State Attributes** tab of the block parameter dialog box to `Auto`. If the block state name does resolve to a signal object, set the `RTWInfo.StorageClass` property of the signal object to `Auto`.
	Block parameter **Initial value** (`InitialValue`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Correct the **Initial value** setting.
	Block parameter **Signal type** (`SignalType`) is set to `complex`. Complex values are not supported for code inspection.	Set **Signal type** to `auto` or `real`.
	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: • 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.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Data Store Read blocks **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 6-26.	The block cannot specify elements. Block parameter **Specify element(s) to select** (`DataStoreElements`) is set to a nonempty string.	Clear element selections from the **Element Selection** tab of the block dialog box.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block output signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Data Store Write blocks **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 6-26.	The block cannot specify elements. Block parameter **Specify element(s) to select** (`DataStoreElements`) is set to a nonempty string.	Clear element selections from the **Element Selection** tab of the block dialog box.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
Check From blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.
Check Goto blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Switch blocks	The first and third input ports and the output port do not have the same data type.	Modify the data ports to have the same data type. Consider selecting the block option **Require all data port inputs to have the same data type**.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block inport or outport 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.	
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.
	Block parameter **Data port order** (`DataPortOrder`) is set to `Specify indices`.	Set **Data port order** to `Zero-based contiguous` or `One-based contiguous`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block inport or outport 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.	
Check Mux blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
Check Demux blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.
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: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`,	Correct the listed block inport or outport.

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

See Also

Chapter 5, "Block Constraints Reference"

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 `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport 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 `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block inport or outport 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.	
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: `conj`, `transpose`, or `hermitian`.	Set **Function** to one of the following values: `exp`, `log`, `10^u`, `log10`, `magnitude^2`, `square`, `pow`, `reciprocal`, `hypot`, `rem`, `mod`, or (for legacy models) `sqrt`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block output signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport 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 `Element-wise(.*)`.	Set **Multiplication** to `Element-wise(.*)`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Sum blocks	Sum, Add, or Subtract blocks must have at least two inports.	Reconfigure the block to have at least two inports.
	Input and output ports do not all have the same data type.	Modify the port data types to match.
	Block parameter **Accumulator data type** (`AccumDataTypeStr`) does not use the same data type as the block inputs.	Modify the block to use the same data type for its inputs and accumulator. Consider setting **Accumulator data type** to `Inherit:  Same as first input`.
	Block parameter **Integer rounding mode** (`RndMeth`) is set to `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Trigonometry blocks	Block parameter **Function** (`Operator`) is set to `cos + jsin` (complex exponential of the input).	Set **Function** to any value other than `cos + jsin`.
	Block parameter **Approximation method** (`ApproximationMethod`) is not set to `None`.	Set **Approximation method** to `None`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
Check Minmax blocks	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 `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `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 Reference"

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 `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block output signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Data Type Duplicate blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	Correct the listed block inport or outport.

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: • 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. • 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 Reference"

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	Relational Operator block outport is not Boolean.	Modify the data type of the outport to `boolean`.
	Block parameter **Relational operator** (`Operator`) is set to an unsupported value: `isInf`, `isNaN`, or `isFinite`.	Set **Relational operator** to a supported value: <=, ==, >=, ~=, <, or >.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• Block output signal storage class is not set to `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Logic blocks	Logical Operator block outport is not Boolean.	Modify the data type of the outport to `boolean`.
	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.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `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 Reference”

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 `InitializeConditionsFcnSpec`, `StartFcnSpec`, or `TerminateFcnSpec`, rather than an `OutputFcnSpec`.	Modify the S-function configuration to specify an `OutputFcnSpec`.
	The S-function has more than one `dwork`.	Modify the S-function configuration to specify one `dwork`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or if the block supports buses, a bus for which the elements (potentially including	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	

See Also

Chapter 5, "Block Constraints Reference"

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 Model Reference blocks **Note** Referenced models cannot accept model arguments. For more information, see "Check model arguments" on page 6-22.	The Model block cannot have variants. Block option **Enable variants** (`Variant`) is selected (set to `on`).	Clear the **Enable variants** option.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Subsystem blocks	The subsystem is a nonvirtual (atomic) subsystem.	If possible, reconfigure the subsystem to be virtual (clear the Subsystem block option **Treat as atomic unit**). Alternatively, wrap the subsystem in a Model block, or explore other implementation options.
	The block cannot have variants. Block parameter **Variant** (`Variant`) is not set to `off`.	Set **Variant** to `off`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `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 Reference”

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 `Single`.	Set **Integer rounding mode** to `Zero` or `Floor`.

Subcheck	Condition	Recommended Action
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `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 Reference"

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 Outport blocks	The block cannot specify variable-dimension signals. Block parameter **Variable-size signal** (`VarSizeSig`) is set to `Yes`.	Set **Variable-size signal** to `No`.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `Auto`.	Correct the listed block inport or outport.

Subcheck	Condition	Recommended Action
	• Block has constant (`Inf`) sample time and an outport has been testpointed.	
Check Terminator blocks	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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 `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 Reference"

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 `RTWInfo.StorageClass` property of the signal object to `Auto`.
	Block parameter **Initial conditions** (`X0`) is empty, is nonfinite, has a MATLAB structure as a value, is complex, has two or more dimensions, or specifies the range (`:`) operator.	Correct the **Initial conditions** setting.
	Violates a constraint that applies to all blocks: • Block inport or outport is not of data type `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `boolean`, or 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
	• 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 `Auto`. • Block has constant (`Inf`) sample time and an outport has been testpointed.	

See Also

Chapter 5, "Block Constraints Reference"

Check usage of root Outport blocks

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

Description

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

Results and Recommended Actions

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

See Also

"Model Configuration Constraints" on page 4-4

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 virtual or nonvirtual bus.	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.

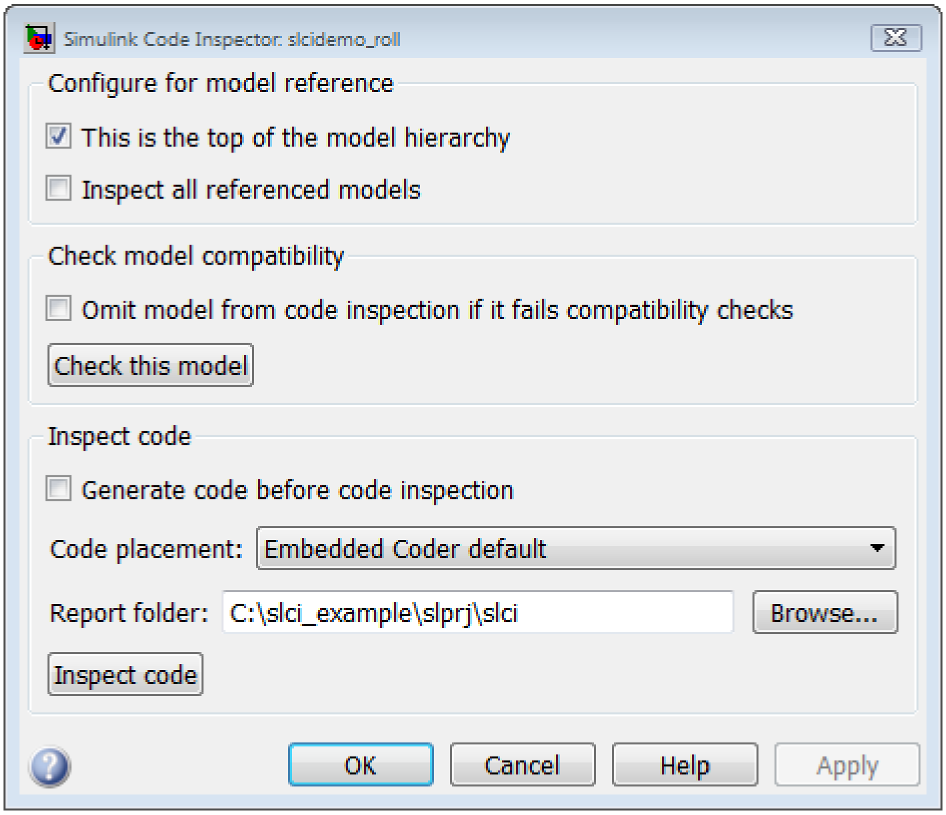
See Also

"Model Configuration Constraints" on page 4-4

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.



Configure for model reference

[✓] This is the top of the model hierarchy

[ ] Inspect all referenced models

Check model compatibility

[ ] Omit model from code inspection if it fails compatibility checks

Inspect code

[ ] Generate code before code inspection

Code placement: Embedded Coder default

Report folder: C:\slci_example\slprj\slci

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

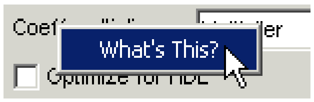
Simulink Code Inspector Dialog Box Overview

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 `slprj/slci` relative to the location of the model.

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

Command-Line Information

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

See Also

"Code Inspection"