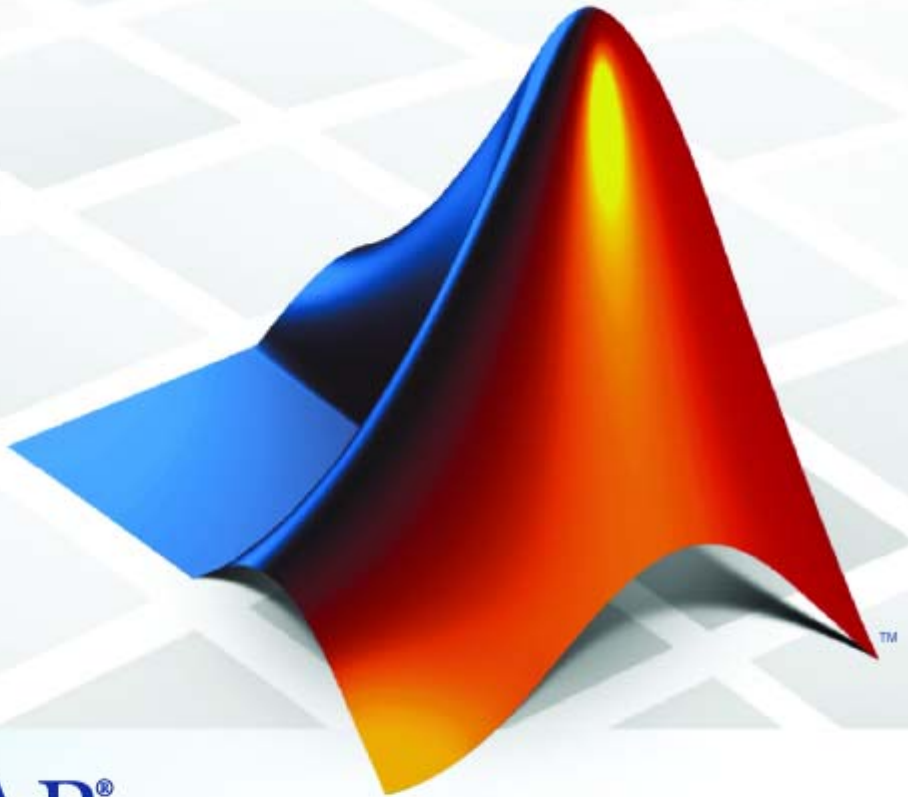


# MathWorks™ Automotive Advisory Board

Control Algorithm Modeling Guidelines  
Using MATLAB®, Simulink®, and Stateflow®  
(Version 2.0)



MATLAB®  
& SIMULINK®

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto: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.

*MathWorks™ Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow® (Version 2.0)*

© COPYRIGHT 2007–2009 by MathWorks™ Automotive Advisory Board

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

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

### Trademarks

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

### Patents

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### Revision History

March 2009      Online only      Release 2009a

## Introduction

### 1

<b>Presentation of Guidelines Hosted by The MathWorks</b> .....	1-2
<b>Motivation</b> .....	1-3
<b>Guideline Template</b> .....	1-4
Guideline ID .....	1-5
Guideline Title .....	1-5
Priority .....	1-6
Scope .....	1-7
MATLAB Versions .....	1-8
Prerequisites .....	1-8
Description .....	1-8
Rationale .....	1-9
Last Change .....	1-10
Model Advisor Check .....	1-10
<b>Document Usage</b> .....	1-11

## Naming Conventions

### 2

<b>General Guidelines</b> .....	2-2
<b>Model Content</b> .....	2-7

## Model Architecture

---

### 3

Simulink and Stateflow Partitioning .....	3-2
Subsystem Hierarchies .....	3-14
J-MAAB Model Architecture Decomposition .....	3-21

## Model Configuration Options

---

### 4

Model Configuration Options .....	4-2
-----------------------------------	-----

## Simulink

---

### 5

Diagram Appearance .....	5-2
Signals .....	5-27
Block Usage .....	5-35
Block Parameters .....	5-55
Simulink Patterns .....	5-62

**6**

---

Chart Appearance .....	6-2
Stateflow Data and Operations .....	6-20
Events .....	6-39
Statechart Patterns .....	6-43
Flowchart Patterns .....	6-49

**Recommendations for Automation Tools**

**A**

**Guideline Writing**

**B**

**Flowchart Reference**

**C**

**Background Information on Basic Blocks and Signals**

**D**

---

Basic Blocks .....	D-2
Signals and Signal Labels .....	D-3



# Introduction

---

- “Presentation of Guidelines Hosted by The MathWorks” on page 1-2
- “Motivation” on page 1-3
- “Guideline Template” on page 1-4
- “Document Usage” on page 1-11

## **Presentation of Guidelines Hosted by The MathWorks**

This presentation of the MathWorks™ Automotive Advisory Board (MAAB) guidelines, Version 2.0, is based on the document, of the same title, authored by the MAAB working group. In addition to the information included in the original document, this presentation includes references to corresponding Model Advisor MAAB checks that you can apply if you are licensed to use Simulink® and Simulink® Verification and Validation™ software.



## Motivation

The MathWorks Automotive Advisory Board (MAAB) guidelines are important for project success and teamwork—both in-house and when cooperating with partners or subcontractors. Observing the guidelines is a key prerequisite to achieving:

- Problem-free system integration
- Well-defined interfaces
- Uniform appearance of models, code, and documentation
- Reusable and readable models
- Problem-free exchange of models
- A simple, effective process
- Professional documentation
- Understandable presentations
- Fast software changes
- Cooperation with subcontractors
- Successful transitions of research or predevelopment projects to product development

## Guideline Template

In this section...
“Guideline ID” on page 1-5
“Guideline Title” on page 1-5
“Priority” on page 1-6
“Scope” on page 1-7
“MATLAB Versions” on page 1-8
“Prerequisites” on page 1-8
“Description” on page 1-8
“Rationale” on page 1-9
“Last Change” on page 1-10
“Model Advisor Check” on page 1-10

Guideline descriptions are documented, using the following template. Companies that want to create additional guidelines are encouraged to use the same template.

<b>ID: Title</b>	<i>XX_nnnn</i> : Title of the guideline (unique, short)
<b>Priority</b>	Mandatory, Strongly recommended, or Recommended
<b>Scope</b>	MAAB, NA-MAAB, J-MAAB, Specific Company (for optional local company usage)
<b>MATLAB® Versions</b>	One of the following: All RX, RY, RZ RX and earlier RX and later RX through RY
<b>Prerequisites</b>	Links to guidelines, which are prerequisites to this guideline (ID: Title)

<b>Description</b>	Description of the guideline (text, images)
<b>Rationale</b>	Motivation for the guideline
<b>Last Change</b>	Version number of last change
<b>Model Advisor Check</b>	Title of and link to the corresponding Model Advisor check, if a check exists

---

**Note** The elements of this template are the minimum required items for understanding and exchanging guidelines. You can add project or vendor fields to this template as long as their meaning does not overlap with existing fields. Such additions are encouraged if they help to integrate other guideline templates and lead to a wider acceptance of the core template.

---

## Guideline ID

- The guideline ID is built out of two lowercase letters (representing the origin of the rule) and a four-digit number, separated by an underscore.
- Once a new guideline has an ID, the ID does not change.
- The ID is used for references to guidelines.
- The two letter prefixes **na**, **jp**, **jc** and **eu** are reserved for future MAAB committee rules.
- Legacy prefixes, **db**, **jm**, **hd**, and **ar**, are reserved. The MAAB committee will not use these prefixes for new rules.
- No new rules are to be written with these legacy prefixes.

## Guideline Title

- The title should be a short, but unique description of the guidelines area of application (for example, length of names)
- The title is used for the Prerequisites field and for custom checker tools.
- The title text should appear with a hyperlink that links to the guideline.

**Note** The title should not be a redundant short description of the guidelines content, because while the latter may change over time, the title should remain stable.

## Priority

Each guideline must be rated with one of the following priorities:

- Mandatory
- Strongly recommended
- Recommended

The priority describes the importance of the guideline and determines the consequences of violations.

<b>Mandatory</b>	<b>Strongly Recommended</b>	<b>Recommended</b>
<b>Definition</b>		
Guidelines that all companies agree to that are absolutely essential  Guidelines that all companies conform to 100%	Guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming to the guideline 100%  Models should conform to these guidelines to the greatest extent possible; however, 100% compliance is not required	Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model  Guidelines where conformance is preferred, but not required
<b>Consequences:</b> If the guideline is violated,		

<b>Mandatory</b>	<b>Strongly Recommended</b>	<b>Recommended</b>
Essential items are missing The model might not work properly	The quality and appearance deteriorates An adverse effect on maintainability, portability, and reusability might occur	The appearance does not conform with other projects
<b>Waiver Policy:</b> If the guideline is intentionally ignored,		
The reasons must be documented		

## Scope

The scope of a guideline may be set to one of the following:

<b>Scope</b>	<b>Description</b>
MAAB (MathWorks Automotive Advisory Board)	A group of automotive manufacturers and suppliers that work closely together with The MathWorks™. MAAB includes the subgroups J-MAAB and NA-MAAB.
J-MAAB (Japan MAAB)	A subgroup of MAAB that includes automotive manufacturers and suppliers in Japan and works closely with The MathWorks. Rules with J-MAAB scope are local to Japan.
NA-MAAB (North American MAAB)	A subgroup of MAAB that includes automotive manufacturers and suppliers in the United States and Europe and works closely with The MathWorks. Rules with NA-MAAB scope are local to the United States and Europe.

## MATLAB Versions

The guidelines support all versions of the MATLAB and Simulink products. If the rule applies to specific versions, the versions are identified in the MATLAB versions field. The version information is in one of the following formats.

Format	Definition
All	All versions of MATLAB
RX, RY, or RZ	A specific version of MATLAB
RX and earlier	Versions of MATLAB until version RX
RX and later	Versions of MATLAB from version RX to the current version
RX through RY	Versions of MATLAB between RX and RY

## Prerequisites

- The Prerequisite field is for links to other guidelines that are prerequisites for this guideline (logical conjunction).
- Use the guideline ID (for consistency) and the title (for readability) for the links.
- The Prerequisites field should not contain any other text.

## Description

- This field contains a detailed description of the guideline.
- If needed, add images and tables.

---

**Note** If formal notation (math, regular expression, syntax diagrams, and exact numbers/limits) is available, use it to unambiguously describe a guideline and specify an automated check. However, a human, understandable, informal description must always be provided for daily reference.

---

## Rationale

This field lists the reasons that apply for a given guideline. You can recommend guidelines for one or more of the following reasons:

Rationale	Description
Readability	Easily understood algorithms <ul style="list-style-type: none"> <li>• Readable models</li> <li>• Uniform appearance of models, code, and documentation</li> <li>• Clean interfaces</li> <li>• Professional documentation</li> </ul>
Workflow	Effective development process and workflow <ul style="list-style-type: none"> <li>• Ease of maintenance</li> <li>• Rapid model changes</li> <li>• Reusable components</li> <li>• Problem-free exchange of models</li> <li>• Model portability</li> </ul>
Simulation	Efficient simulation and analysis <ul style="list-style-type: none"> <li>• Simulation speed</li> <li>• Simulation memory</li> <li>• Model instrumentation</li> </ul>
Verification and validation	Ability to verify and validate a model and generated code with <ul style="list-style-type: none"> <li>• Requirements traceability</li> <li>• Testing</li> <li>• Problem-free system integration</li> <li>• Clean interfaces</li> </ul>
Code generation	Generation of code that is efficient and effective for embedded systems <ul style="list-style-type: none"> <li>• Fast software changes</li> <li>• Robustness of generated code</li> </ul>

## **Last Change**

The Last change field contains the document version number.

## **Model Advisor Check**

The Simulink Verification and Validation product includes MAAB checks, which correspond to a subset of MAAB guidelines, that you can select and run with the Simulink Model Advisor. The Model Advisor check field contains the title of and a link to the corresponding Model Advisor MAAB check, if a check exists.

For a list of available Model Advisor checks for the MAAB guidelines, see “MathWorks Automotive Advisory Board Checks” in the Simulink Verification and Validation documentation. For information on using the Model Advisor, see “Consulting the Model Advisor” in the Simulink documentation.



## Document Usage

- Chapter 2, “Naming Conventions” and Chapter 3, “Model Architecture” provide basic guidelines that apply to all types of models.
- Chapter 5, “Simulink” and Chapter 6, “Stateflow” deal with specific rules for those environments.
- Some guidelines are dependent on other guidelines and are explicitly listed throughout the document.

For information on automated checking of the guidelines, see Appendix A, “Recommendations for Automation Tools”.



# Naming Conventions

---

- “General Guidelines” on page 2-2
- “Model Content” on page 2-7

### **General Guidelines**

ar\_0001: Filenames

ar\_0002: Directory names

<b>ID: Title</b>	ar_0001: Filenames
<b>Priority</b>	Mandatory
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	A file name conforms to the following constraints:

## Form

*filename = name.extension*

- *name*: no leading digits, no blanks
- *extension*: no blanks

## Uniqueness

All file names within the parent project directory

## Allowed Characters

*name*:

abcdefghijklmnopqrstuvwxyz  
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
0123456789\_

*extension*:

abcdefghijklmnopqrstuvwxyz  
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
0123456789

## Underscores

*name*:

## ar\_0001: Filenames

---

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

*extension:*

Should not use underscores

### **Rationale**

- Readability
- Workflow

### **Last Changed**

V1.0

### **Model Advisor Check**

“Check for invalid file names”

**Priority** Mandatory

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** A directory name conforms to the following constraints:

**Form**

*directory name = name*

*name*: no leading digits, no blanks

**Uniqueness**

All directory names within the parent project directory

**Allowed characters**

*name*:

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9 \_

**Underscores**

*name*:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

**Rationale**

- Readability

- Workflow

## ar\_0002: Directory names

---

**Last  
Changed**

V1.0

**Model  
Advisor  
Check**

“Check for invalid model directory names”



### Model Content

jc\_0201: Usable characters for  
Subsystem names

jc\_0211: Usable characters for  
Inport blocks and Outport blocks

jc\_0221: Usable characters for  
signal line names

jc\_0231: Usable characters for  
block names

na\_0014: Use of local language in  
Simulink and Stateflow

# jc\_0201: Usable characters for Subsystem names

---

<b>ID: Title</b>	jc_0201: Usable characters for Subsystem
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	The names of all Subsystem blocks should conform to the following constraints:

## **Form**

*name:*

- Should not start with a number
- Should not include blank spaces

## **Allowed Characters**

*name:*

a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 \_

## **Underscores**

*name:*

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

# jc\_0201: Usable characters for Subsystem names

---

## **Rationale**

- Readability
- Workflow
- Code generation

## **Last Changed**

V2.0

## **Model Advisor Check**

“Check whether subsystem block names include invalid characters”

# jc\_0211: Usable characters for Inport blocks and Output blocks

---

**ID: Title** jc\_0211: Usable characters for Inport blocks and Output blocks

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** The names of all Inport blocks and Output blocks should conform to the following constraints:

## **Form**

*name:*

- Should not start with a number
- Should not include blank spaces

## **Allowed Characters**

*name:*

a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 \_

## **Underscores**

*name:*

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

# jc\_0211: Usable characters for Inport blocks and Output blocks

---

## **Rationale**

- Readability
- Workflow
- Code generation

## **Last Changed**

V2.0

## **Model Advisor Check**

“Check whether Inport and Output block names include invalid characters”

# jc\_0221: Usable characters for signal line names

---

**ID: Title** jc\_0221: Usable characters for signal line names

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Identifies named signals constraints

## **Form**

*name:*

- Should not start with a number
- Should not include blank spaces
- Should not include any control characters

## **Allowed Characters**

*name:*

a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 \_

## **Underscores**

*name:*

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

# jc\_0221: Usable characters for signal line names

---

## **Rationale**

- Readability
- Workflow
- Code generation

## **Last Changed**

V2.0

## **Model Advisor Check**

“Check whether signal line names include invalid characters”

# jc\_0231: Usable characters for block names

---

**ID: Title** jc\_0231: Usable characters for block names

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** jc\_0201: Usable characters for Subsystem names

**Description** The names of all blocks should conform to the following constraints:

**Form**

*name:*

- Should not start with a number
- Should not include blank spaces
- Should not use double byte characters
- Carriage returns are allowed

**Allowed Characters**

*name:*

a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 \_

---

**Note** This rule does not apply to Subsystem blocks.

---

- Rationale**
- Readability
  - Workflow
  - Code generation



# jc\_0231: Usable characters for block names

---

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

“Check whether block names include invalid characters”

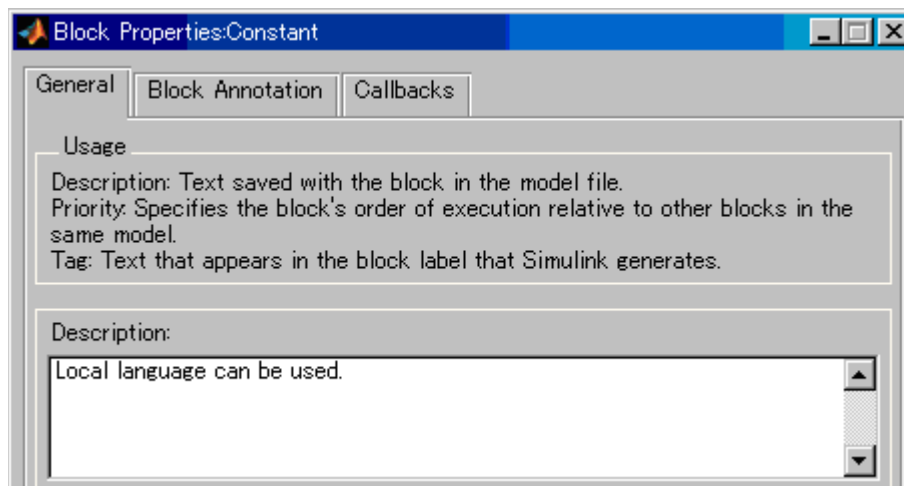
# na\_0014: Use of local language in Simulink and Stateflow

---

<b>ID: Title</b>	na_0014: Use of local language in Simulink and Stateflow
<b>Priority</b>	Strongly recommended
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	Use the local language in descriptive fields only. Descriptive fields are text entry points that do not affect code generation or simulation. Examples of descriptive fields include the <b>Description</b> field in the Block Properties dialog box.

## Simulink Examples

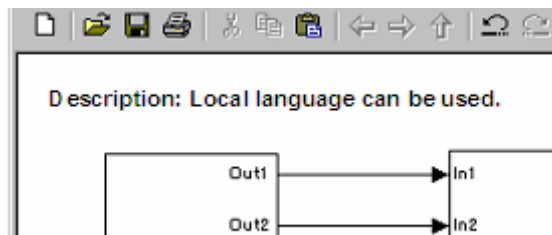
- The **Description** field in the Block Properties dialog box



- Text annotation entered directly in the model

# na\_0014: Use of local language in Simulink and Stateflow

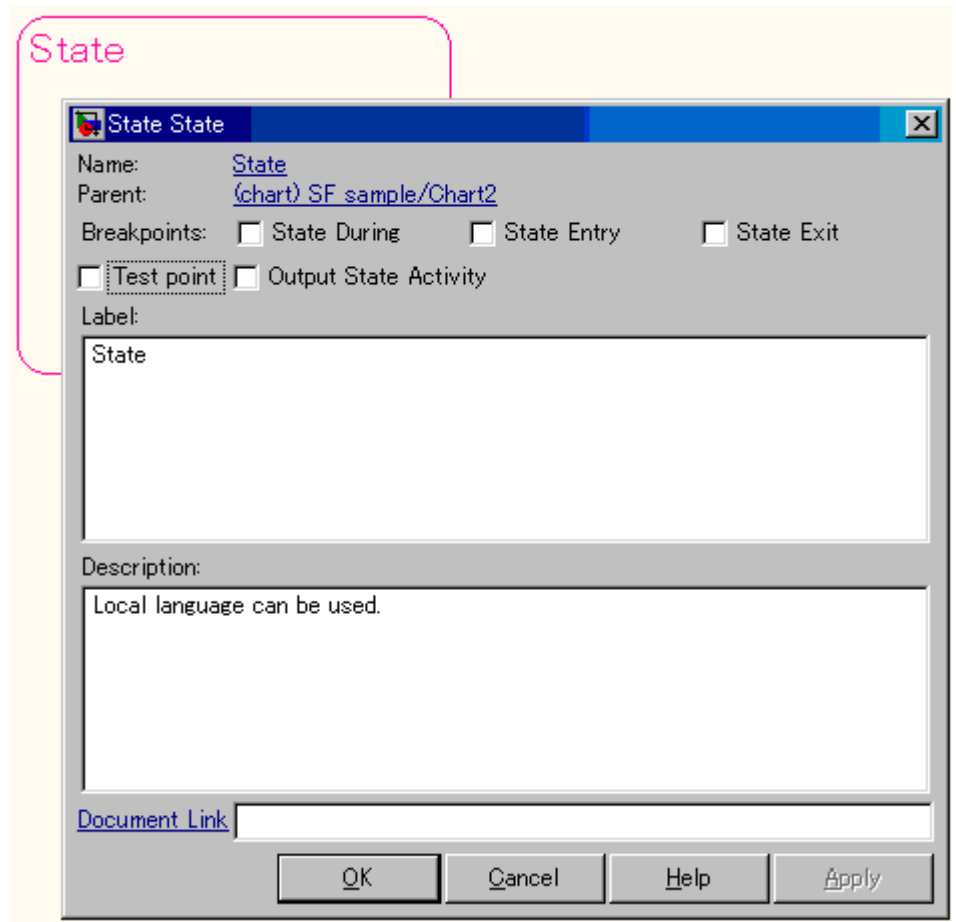
---



## Stateflow® Examples

- The **Description** field of chart and state Properties

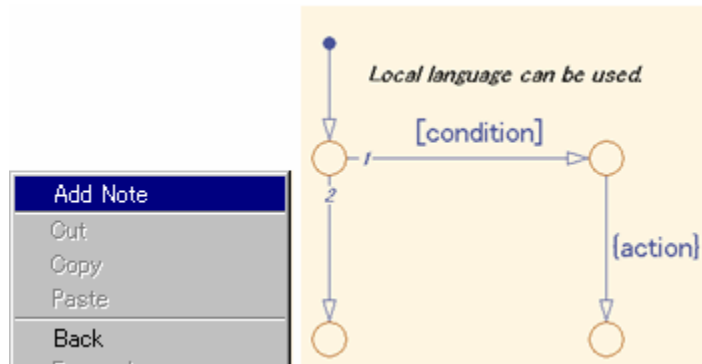
# na\_0014: Use of local language in Simulink and Stateflow



- Annotation description added using **Add Note**

# na\_0014: Use of local language in Simulink and Stateflow

---



---

**Note** It is possible that Simulink cannot open a model that includes local language on different character encoding systems. Therefore, pay attention when using local characters for exchanging models between countries.

---

## Rationale

- Readability
- Workflow

## Last Changed

V2.0

## Model Advisor Check

“Check whether signal line names include invalid characters”

## **na\_0014: Use of local language in Simulink and Stateflow**

---

# Model Architecture

---

- “Simulink and Stateflow Partitioning” on page 3-2
- “Subsystem Hierarchies” on page 3-14
- “J-MAAB Model Architecture Decomposition” on page 3-21

## **Simulink and Stateflow Partitioning**

na\_0006: Guidelines for mixed use  
of Simulink and Stateflow

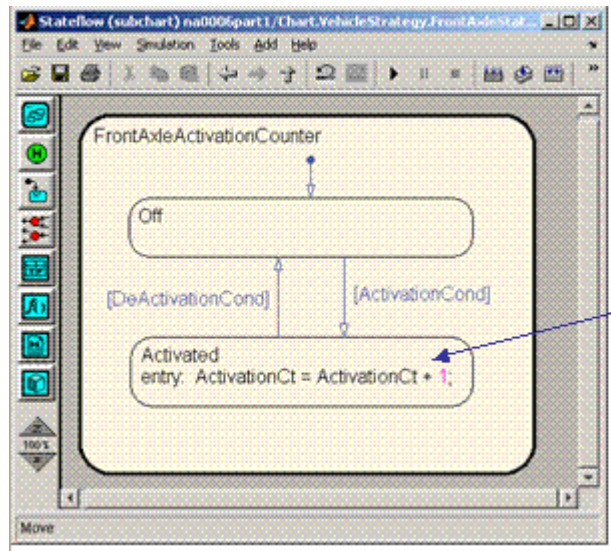
na\_0007: Guidelines for use of Flow  
Charts, Truth Tables and State  
Machines



# na\_0006: Guidelines for mixed use of Simulink and Stateflow

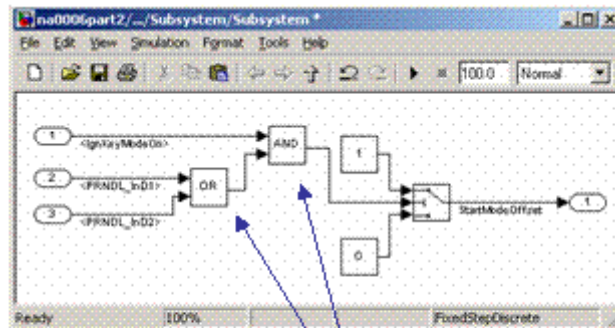
<b>ID: Title</b>	na_0006: Guidelines for mixed use of Simulink and Stateflow
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.</p> <ul style="list-style-type: none"><li>• If the function primarily involves complicated logical operations, use Stateflow features.  Use Stateflow features to implement modal logic, where the control function to be performed at the current time depends on a combination of <i>past and present logical conditions</i>.</li><li>• If the function primarily involves numerical operations, use Simulink features.</li></ul> <p><b>Specifics</b></p> <ul style="list-style-type: none"><li>• If the primary nature of the function is logical, but some simple numerical calculations are done to support the logic, implement the simple numerical functions using the Stateflow action language.</li></ul>

# na\_0006: Guidelines for mixed use of Simulink and Stateflow



Embedded simple math operation

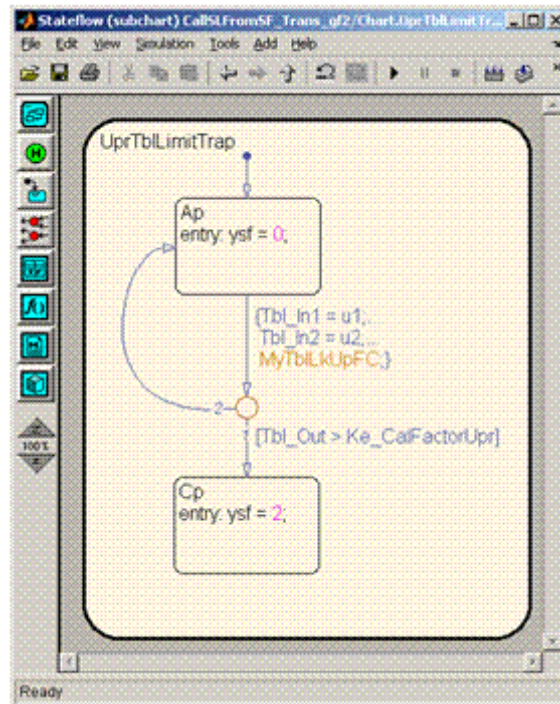
- If the primary nature of the function is numeric, but some simple logical operations are done to support the arithmetic, implement the simple logical functions with Simulink features.



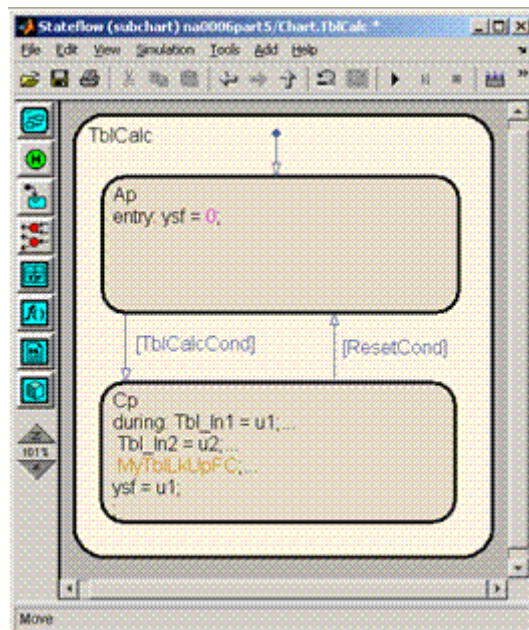
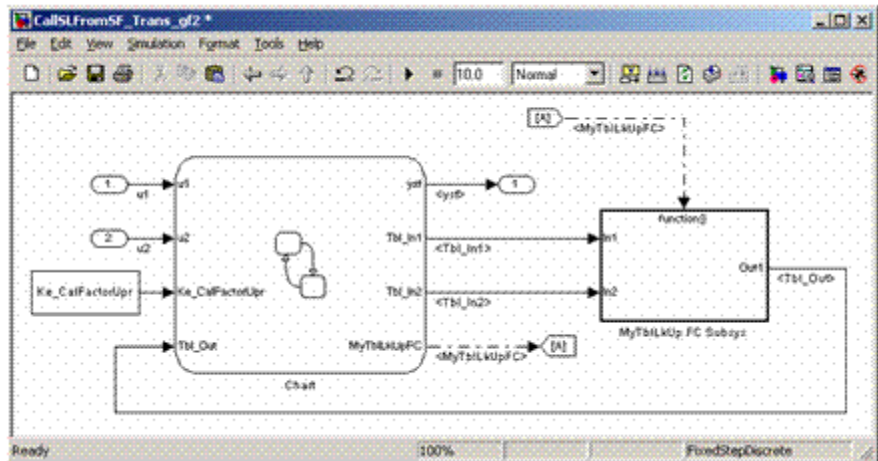
Embedded simple logic operations

# na\_0006: Guidelines for mixed use of Simulink and Stateflow

- If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, use a Simulink subsystem to implement the numerical calculations. A Stateflow chart should invoke the execution of the subsystem, using a function call.

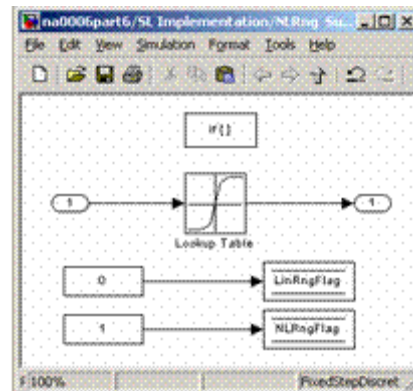
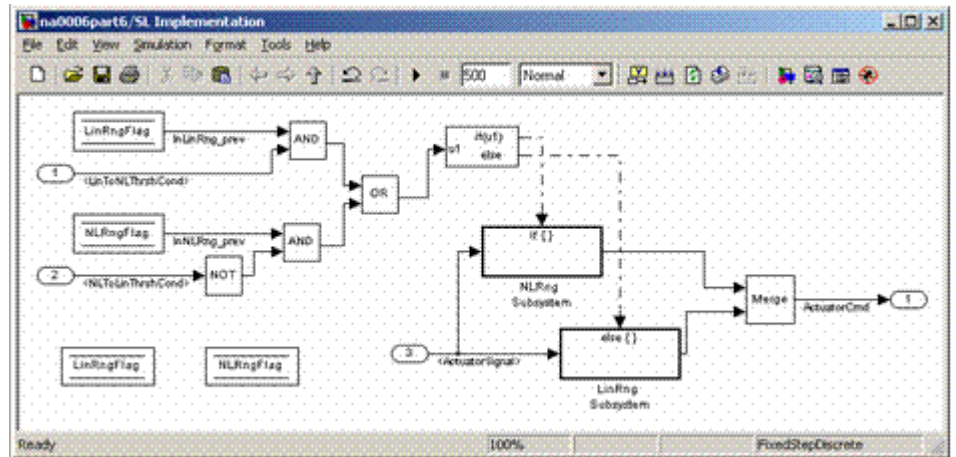


# na\_0006: Guidelines for mixed use of Simulink and Stateflow



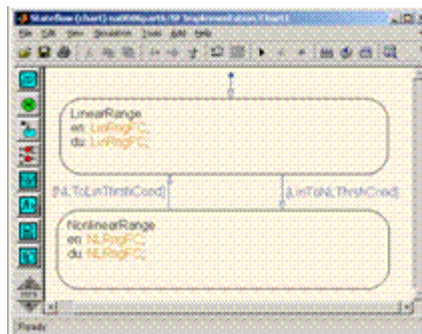
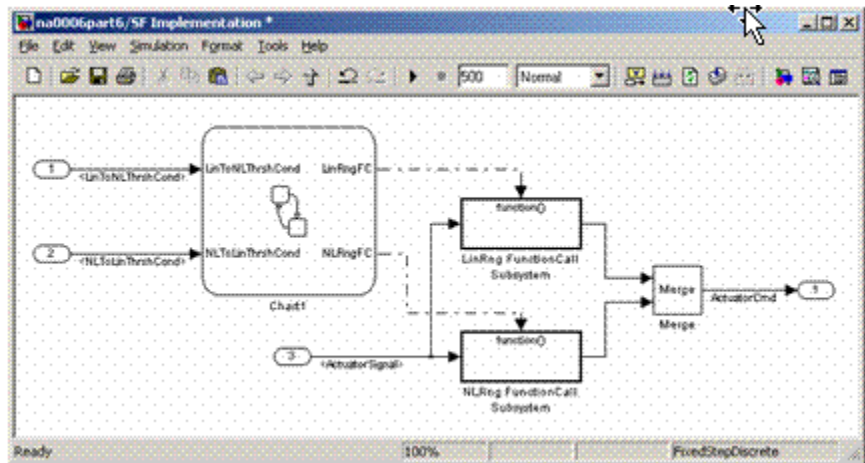


# na\_0006: Guidelines for mixed use of Simulink and Stateflow



**Incorrect**

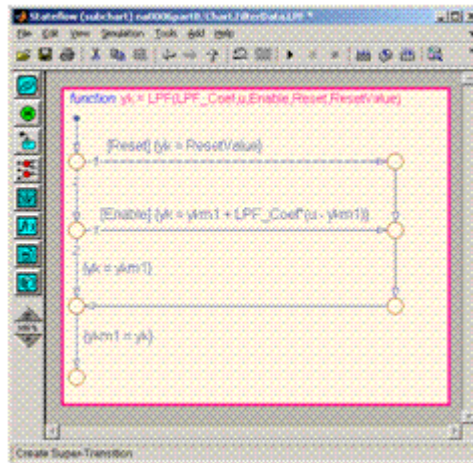
# na\_0006: Guidelines for mixed use of Simulink and Stateflow



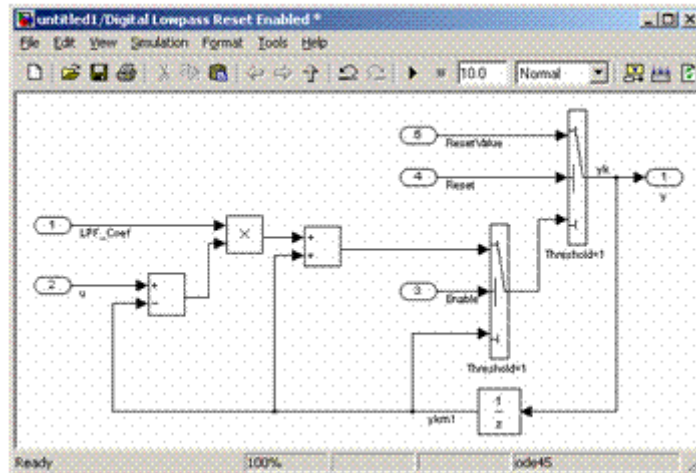
## Correct

- Use Simulink to implement numerical expressions containing continuously-valued states, such as: difference equations, integrals, derivatives, and filters.

# na\_0006: Guidelines for mixed use of Simulink and Stateflow



**Incorrect**



**Correct**

- Use Simulink to implement numerical expressions containing continuously-valued states, such as: difference equations, integrals, derivatives, and filters.



# na\_0006: Guidelines for mixed use of Simulink and Stateflow

---

<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Simulation</li><li>• Verification and Validation</li><li>• Code Generation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

# na\_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

---

<b>ID: Title</b>	na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	na_0006: Guidelines for mixed use of Simulink and Stateflow
<b>Description</b>	<p>Within Stateflow, the choice of whether to use a flow chart or a state chart to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.</p> <ul style="list-style-type: none"><li>• If the primary nature of the function segment is to calculate modes of operation or discrete-valued states, use state charts. Some examples are:<ul style="list-style-type: none"><li>▪ Diagnostic models with pass, fail, abort, and conflict states</li><li>▪ Model that calculates different modes of operation for a control algorithm</li></ul></li><li>• If the primary nature of the function segment involves <code>if-then-else</code> statements, use flowcharts or truth tables.</li></ul> <p><b>Specifics</b></p> <p>If the primary nature of a function segment is to calculate modes or states, but <code>if-then-else</code> statements are required, add a flow chart to a state within the state chart. (See “Flowchart Patterns” on page 6-49.)</p>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>

# na\_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

---

- Simulation
- Verification and Validation
- Code Generation

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# na\_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

---

## Subsystem Hierarchies

db\_0040: Model hierarchy

db\_0143: Similar block types on  
the model levels

db\_0144: Use of Subsystems

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.

# db\_0143: Similar block types on the model levels

**ID: Title** db\_0143: Similar block types on the model levels

**Priority** Strongly recommended



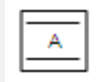


**Scope** NA-MAAB

**MATLAB Versions** All

**Prerequisites** None

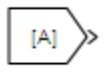
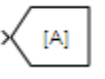

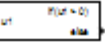
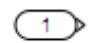
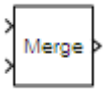
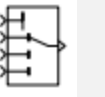

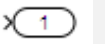
**Description** You must design every level of a model with building blocks of the same type; only subsystems or only basic blocks.

## Blocks that You Can Place at any Model Level

Block	Example
Bus Creator	
Bus Selector	
Data Store Memory	
Demux	
Enable (not on highest model level)	

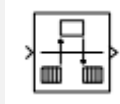
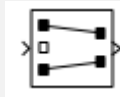
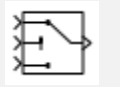
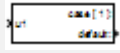



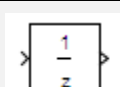
# db\_0143: Similar block types on the model levels

## Blocks that You Can Place at any Model Level (Continued)

Block	Example
From	
Goto	
Ground	
If	
Inport	
Merge	
Multiport Switch	
Mux	
Outport	

# db\_0143: Similar block types on the model levels

## Blocks that You Can Place at any Model Level (Continued)

Block	Example
Rate Transition	
Selector	
Switch	
Switch Case	
Terminator	
Trigger (not on highest model level)	
Type Conversion	
Unit Delay	

**Note** You cannot place Trigger or Enable blocks at the root level of a model.

## db\_0143: Similar block types on the model levels

---

### **Rationale**

- Readability
- Workflow
- Verification and Validation

### **Last Changed**

V2.0

### **Model Advisor Check**

“Check for systems that mix primitive blocks and subsystems”



<b>ID: Title</b>	db_0144: Use of Subsystems
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>Group blocks in a Simulink diagram together into subsystems based on functional decomposition of the algorithm, or portion thereof, represented in the diagram.</p> <p>Avoid grouping blocks into subsystems primarily for saving space in the diagram. Each subsystem in the diagram should represent a unit of functionality required to accomplish the purpose of the model or submodel.</p>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li><li>• Code Generation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

## db\_0040: Model hierarchy

---

<b>ID: Title</b>	db_0040: Model hierarchy
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	The model hierarchy should correspond to the functional structure of the control system.
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li><li>• Code Generation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

### **J-MAAB Model Architecture Decomposition**

jc\_0301: Controller model

jc\_0311: Top layer/root level

jc\_0321: Trigger layer

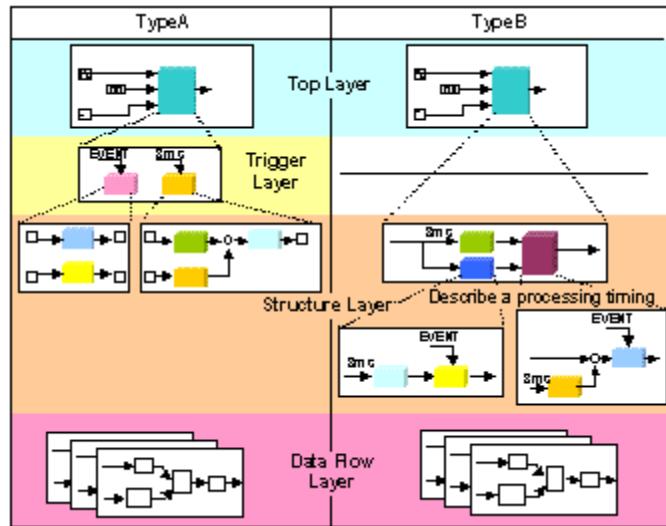
jc\_0331: Structure layer

jc\_0341: Data flow layer

# jc\_0301: Controller model

---

<b>ID: Title</b>	jc_0301: Controller model
<b>Priority</b>	Mandatory
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>Control models are organized using the following hierarchical structure. Details on each layer are provided in corresponding rules.</p> <ul style="list-style-type: none"><li>• Top layer (root level), jc_0311: Top layer/root level</li><li>• Trigger layer, jc_0321: Trigger layer</li><li>• Structure layer. jc_0331: Structure layer</li><li>• Data flow layer, jc_0341: Data flow layer</li></ul> <p>Use of the Trigger level is optional. In the following figure, Type A shows the use of a trigger level while Type B shows a model without a trigger level.</p>



## Controller Model

**Rationale**

Workflow

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# jc\_0311: Top layer/root level

---

**ID: Title** jc\_0311: Top layer/root level

**Priority** Mandatory

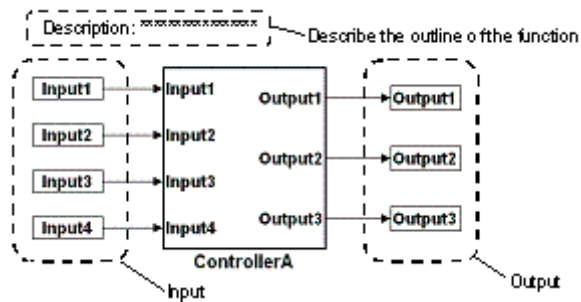
**Scope** J-MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Items to describe in a top layer are as follows:

- Overview: Explanation of model feature overview
- Input: Input variables
- Output: Output variables



## Top Layer Example

**Rationale** Workflow

**Last Changed** V2.0

**Model  
Advisor  
Check**

Not applicable

# jc\_0321: Trigger layer

---

**ID: Title** jc\_0321: Trigger layer

**Priority** Mandatory

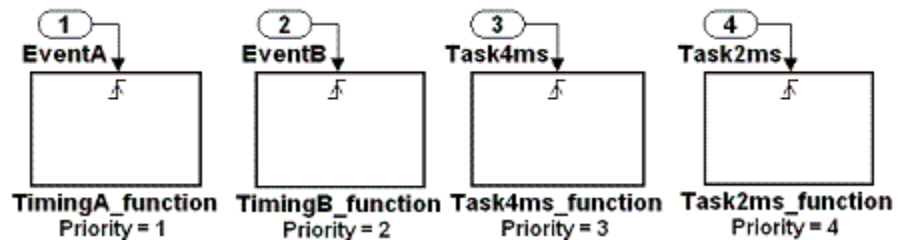
**Scope** J-MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** A trigger layer indicates the processing timing by using Triggered Subsystem or Function-Call Subsystem blocks.

- The blocks should set Priority, if needed.
- The priority value must be displayed as a block annotation. You should be able to understand the priority-based order without having to open the block.



## Trigger Layer Example

### Rationale

- Readability
- Workflow
- Code Generation



**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# jc\_0331: Structure layer

**ID: Title** jc\_0331: Structure layer

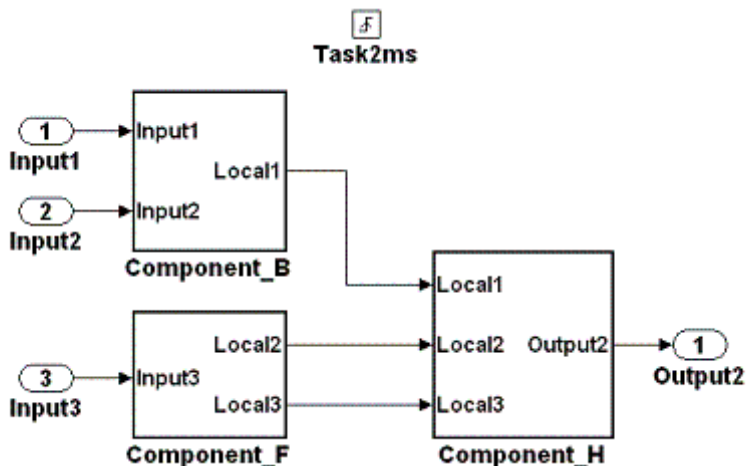
**Priority** Mandatory

**Scope** J-MAAB

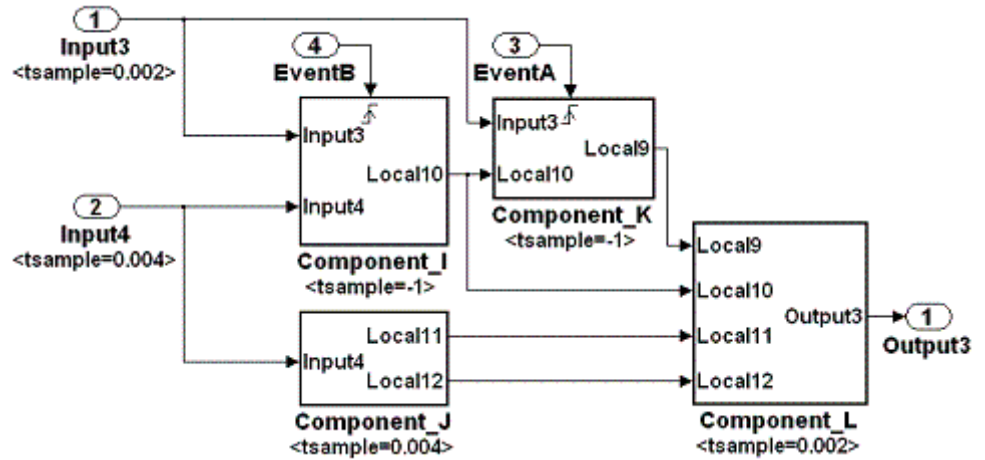
**MATLAB Versions** All

**Prerequisites** None

- Description**
- Describe a structure layer like the following structure layer example.
  - In the case of Type B, specify sample time at an Inport block or a Subsystem block to define task time of the subsystem.
  - In the case of Type B, use a block annotation at an Inport block or a Subsystem block and display sample time to clarify task time of the subsystem.
  - A subsystem of a structure layer should be an atomic subsystem.



**Structure Layer Example (Type A: No Description of Processing Timing)**



**Structure Layer Example (Type B: Description of Processing Timing)**

**Rationale**

- Workflow

**Last Changed**

V2.0

**Model Advisor Check**

Not applicable

# jc\_0341: Data flow layer

**ID: Title** jc\_0341: Data flow layer

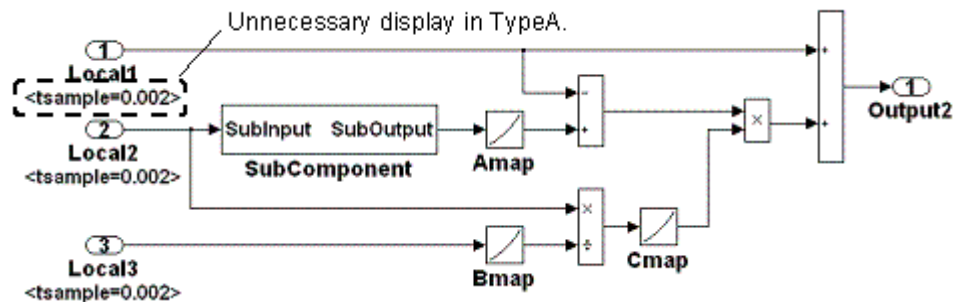
**Priority** Mandatory

**Scope** J-MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Describe a data flow layer as in the following example. In the case of Type A, use a block annotation at an Inport block and display its sample time to clarify execution timing of the signal.



## Data Flow Layer Example

**Rationale** Workflow

**Last Changed** V2.0

**Model Advisor Check** Not applicable

# Model Configuration Options

---

## **Model Configuration Options**

jc\_0011: Optimization parameters  
for Boolean data types

jc\_0021: Model diagnostic settings

# jc\_0011: Optimization parameters for Boolean data types

<b>ID: Title</b>	jc_0011: Optimization parameters for Boolean data types
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	na_0002: Appropriate implementation of fundamental logical and numerical operations

**Description** The optimization option for Boolean data types must be enabled (on).

<b>MATLAB Version</b>	<b>Option Name</b>
R13SP2 and earlier	Boolean Logic signals
R14 and later	Use logic signals as Boolean data. (versus double)

- Rationale**
- Workflow
  - Code Generation

**Last Changed** V2.0

**Model Advisor Check** “Check optimization parameters for Boolean data types”

# jc\_0021: Model diagnostic settings

---

**ID: Title** jc\_0021: Model diagnostic settings

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** The following diagnostics must be enabled. An enabled diagnostic is set to warning or error. Setting the diagnostic option to none is not permitted. Diagnostics that are not listed may be set to any value (none, warning, or error).

## **Solver Diagnostics**

- Algebraic loop
- Minimize algebraic loop

## **Sample Time Diagnostics**

- Multitask rate transition

## **Data Validity Diagnostics**

- Inf or NaN block output
- Duplicate data store names

## **Connectivity**

- Unconnected block input ports
- Unconnected block output ports
- Unconnected line
- Unspecified bus object at root Outport block



- Mux blocks used to create bus signals
- Invalid function-call connection
- Element name mismatch

## **Rationale**

- Workflow
- Code Generation

## **Last Changed**

V2.0

## **Model Advisor Check**

“Check model diagnostic settings”

## **jc\_0021: Model diagnostic settings**

---

# Simulink

---

- “Diagram Appearance” on page 5-2
- “Signals” on page 5-27
- “Block Usage” on page 5-35
- “Block Parameters” on page 5-55
- “Simulink Patterns” on page 5-62

## Diagram Appearance

db\_0032: Simulink signal appearance

db\_0042: Port block in Simulink models

db\_0043: Simulink font and font size

db\_0140: Display of basic block parameters

db\_0141: Signal flow in Simulink models

db\_0142: Position of block names

db\_0146: Triggered, enabled, conditional Subsystems

jc\_0061: Display of block names

jc\_0081: Icon display for Port block

jc\_0171: Maintaining signal flow when using Goto and From blocks

jc\_0281: Naming of Trigger Port block and Enable Port block

jm\_0002: Block resizing

jm\_0010: Port block names in Simulink models

jm\_0013: Annotations

na\_0004: Simulink model appearance

# na\_0004: Simulink model appearance

**ID: Title** na\_0004: Simulink model appearance

**Priority** Recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** The model appearance settings should conform to the following guidelines when the model is released. You can change the settings during the development process.

<b>View Options</b>	<b>Setting</b>
Model Browser	Cleared
Screen color	White
Status Bar	Selected
Toolbar	Selected
Zoom factor	Normal (100%)
<b>Block Display Options</b>	<b>Setting</b>
Background Color	White
Foreground Color	Black
Execution Context Indicator	Cleared
Library Link Display	None
Linearization Indicators	Selected
Model/Block I/O Mismatch	Cleared
Model Block Version	Cleared
Sample Time Colors	Cleared
Sorted Order	Cleared

# na\_0004: Simulink model appearance

---

Signal Display Options	Setting
Port Data Types	Cleared
Signal Dimensions	Cleared
Storage Class	Cleared
Test point Indicators	Selected
Viewer Indicators	Selected
Wide Non-scalar Lines	Selected

## Rationale

- Readability
- Workflow

## Last Changed

V2.0

## Model Advisor Check

“Check for Simulink diagrams that have nonstandard appearance attributes”

# db\_0043: Simulink font and font size

---

<b>ID: Title</b>	db_0043: Simulink font and font size
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	All text elements (block names, block annotations, and signal labels) except free text annotations within a model, must have the same font style and font size. Select font style and font size for legibility.

---

**Note** The selected font should be portable (for example, the Simulink and Stateflow default font) or convertible between platforms (for example, Arial or Helvetica 12pt).

---

<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for difference in font and font sizes”

# db\_0042: Port block in Simulink models

---

<b>ID: Title</b>	db_0042: Port block in Simulink models
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>In a Simulink model, ports must comply with the following rules:</p> <ul style="list-style-type: none"><li>• Place Inport blocks on the left side of the diagram; you may move them to prevent signal crossings.</li><li>• Place Outport blocks on the right side of the diagram; you may move them to prevent signal crossings.</li><li>• You may use duplicate Inport blocks at the subsystem level, if required, but avoid doing so, if possible.</li></ul> <p>Do not use duplicate Inport blocks at the root level.</p>
<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for invalid port positioning and configuration”



# na\_0005: Port block name visibility in Simulink models

**ID: Title** na\_0005: Port block name visibility in Simulink models

**Priority** Strongly recommended

**Scope** MAAB

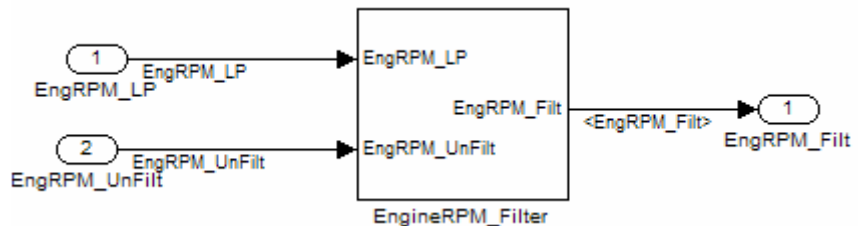
**MATLAB Versions** All

**Prerequisites** None

**Description** While for some items, it is not possible to define a single approach that may apply to all organizations' internal processes, it is important that, at least within a given organization, a single consistent approach is followed. An organization applying the guidelines must enforce one of the following alternatives.

Apply one of the following practices:

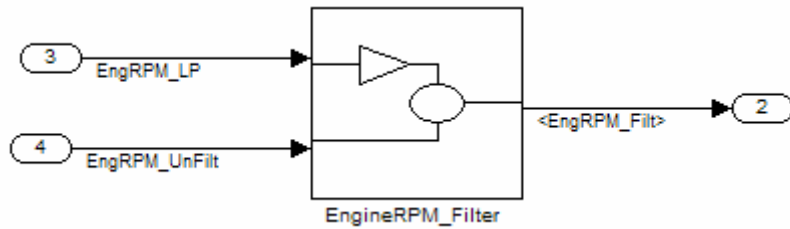
- The name of an Inport or Outport block is not hidden. (**Format > Hide Name** is not allowed.)



- The name of an Inport or Outport block must be hidden. (**Format > Hide Name** is used.)

**Exception:** The names cannot be hidden inside library subsystem blocks.

# na\_0005: Port block name visibility in Simulink models



## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

“Check visibility of port block names”

# jc\_0081: Icon display for Port block

**ID: Title** jc\_0081: Icon display for Port block

**Priority** Recommended

**Scope** MAAB

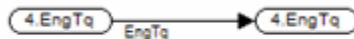
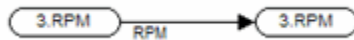
**MATLAB Versions** R14 and later

**Prerequisites** None

**Description** The Icon display setting should be set to Port number for Inport and Outport blocks.



**Correct**



**Incorrect**

**Rationale** Readability

**Last Changed** V2.0

## jc\_0081: Icon display for Port block

---

### **Model Advisor Check**

“Check whether model has unconnected block input ports, output ports, or signal lines”

**ID: Title** jm\_0002: Block resizing

**Priority** Mandatory

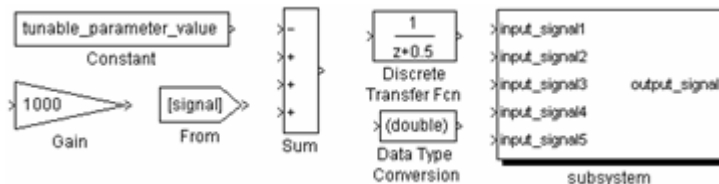
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** All blocks in a model must be sized such that the icon is completely visible and recognizable. In particular, any displayed text (for example, tunable parameters, file names, or equations) in the icon must be readable.

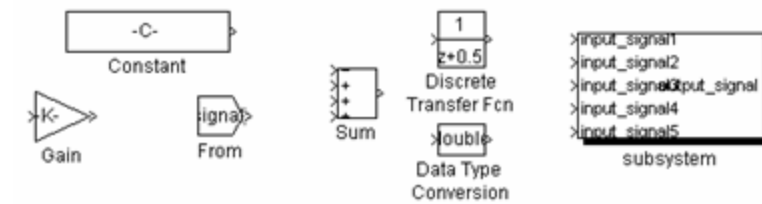
This guideline requires that you resize blocks with variable icons or blocks with a variable number of inputs and outputs. In some cases, it may not be practical or desirable to resize the icon of a subsystem block so that all of the input and output names within it are readable. In such cases, you may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. If you do this, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block.



**Correct**

# jm\_0002: Block resizing

---



**Incorrect**

**Rationale**

Readability

**Last Changed**

V2.0

**Model Advisor Check**

Not applicable

## db\_0142: Position of block names

---

<b>ID: Title</b>	db_0142: Position of block names
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	If shown, place the name of a block below the block.
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check whether block names do not appear below blocks”

# jc\_0061: Display of block names

---

<b>ID: Title</b>	jc_0061: Display of block names
<b>Priority</b>	Recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<ul style="list-style-type: none"><li>• Display a block name when it provides descriptive information.</li><li>• Do not display a block name if the block function is known and understood from the block appearance.</li></ul>
<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check the display attributes of block names”



# db\_0146: Triggered, enabled, conditional Subsystems

---

<b>ID: Title</b>	db_0146: Triggered, enabled, conditional Subsystems
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>Place blocks that define subsystems as conditional or iterative at a consistent location at the top of the subsystem diagram. This applies to the following types of subsystem blocks:</p> <ul style="list-style-type: none"><li>• Function call</li><li>• Enabled</li><li>• Triggered</li><li>• If /Else Action</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for improperly positioned Trigger and Enable blocks”

# db\_0140: Display of basic block parameters

---

<b>ID: Title</b>	db_0140: Display of basic block parameters
<b>Priority</b>	Recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	Display important parameters with values other than the block default values.

---

**Note** The attribute string is one method to support this. The block annotation tab allows you to add the attribute information that you want.

---

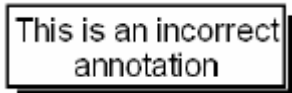
- |                  |                                                                                                     |
|------------------|-----------------------------------------------------------------------------------------------------|
| <b>Rationale</b> | <ul style="list-style-type: none"><li>• Readability</li><li>• Verification and Validation</li></ul> |
|------------------|-----------------------------------------------------------------------------------------------------|

<b>Last Changed</b>	V2.0
---------------------	------

<b>Model Advisor Check</b>	“Check for display of nondefault block attributes”
----------------------------	----------------------------------------------------

<b>ID: Title</b>	jm_0013: Annotations
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	R12.1
<b>Prerequisites</b>	None
<b>Description</b>	Annotations should not have a drop shadow. ( <b>Format &gt; Show Drop Shadow</b> is not allowed.)

This is a correct  
annotation



This is an incorrect  
annotation

<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check whether annotations have drop shadows”

# db\_0032: Simulink signal appearance

---

**ID: Title** db\_0032: Simulink signal appearance

**Priority** Strongly recommended

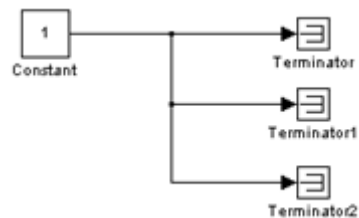
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Signal lines

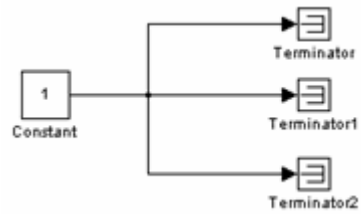
- Should not cross each other, if possible
- Are drawn with right angles
- Are not drawn one upon the other
- Do not cross any blocks
- Should not split into more than two sublimes at a single branching point



**Correct**

# db\_0032: Simulink signal appearance

---



**Incorrect**

**Rationale**

- Readability
- Workflow

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# db\_0141: Signal flow in Simulink models

**ID: Title** db\_0141: Signal flow in Simulink models

**Priority** Strongly recommended

**Scope** MAAB

**Versions** All

**Prerequisites** None

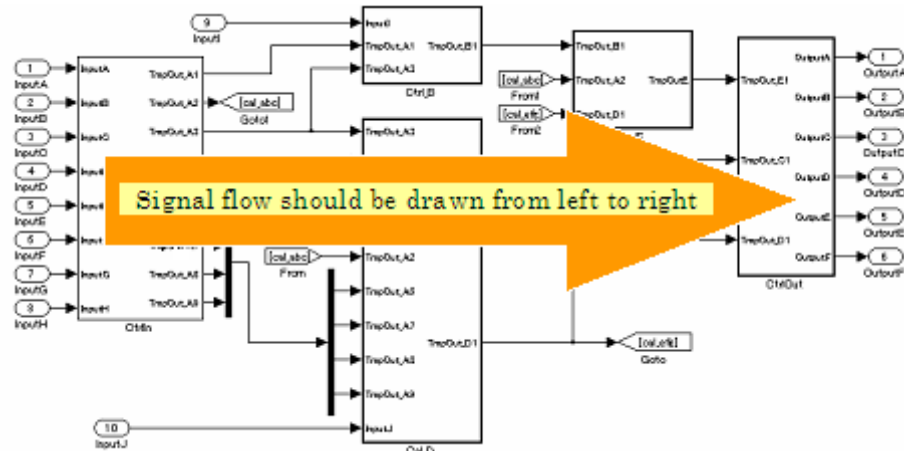
**Description** The signal flow in a model is from left to right.

**Exception:** Feedback loops

Sequential blocks or subsystems are arranged from left to right.

**Exception:** Feedback loops

Parallel blocks or subsystems are arranged from top to bottom.



**Rationale** • Readability

• Workflow

- Verification and Validation

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

“Check for proper use of Switch blocks”

# jc\_0171: Maintaining signal flow when using Goto and From blocks

**ID: Title** jc\_0171: Maintaining signal flow when using Goto and From blocks

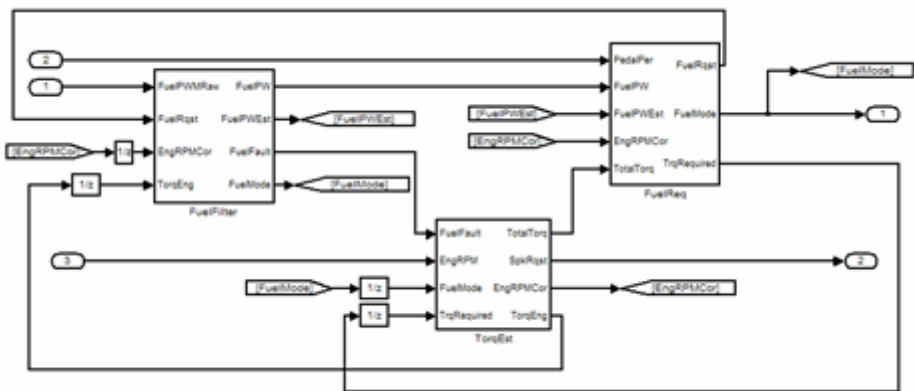
**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

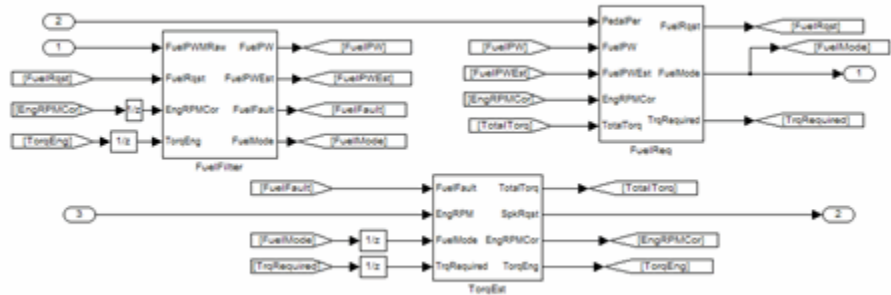
- Description**
- You must maintain visual depiction of signal flow between subsystems.
  - You can use Goto and From blocks provided that you use at least one signal line between connected subsystems.
  - If the subsystems are connected in a feed-forward and a feedback loop, you must connect at least one signal line for each direction.



**Correct**



# jc\_0171: Maintaining signal flow when using Goto and From blocks



## Incorrect

### Rationale

- Readability
- Workflow
- Verification and Validation

### Last Changed

V2.0

### Model Advisor Check

Not applicable

# jm\_0010: Port block names in Simulink models

---

**ID: Title** jm\_0010: Port block names in Simulink models

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites**

- db\_0042: Port block in Simulink models
- na\_0005: Port block name visibility in Simulink models

**Description** For some items, though you may not be able to define a single approach for internal processes of all organizations, within a given organization, try to follow a single, consistent approach. An organization applying the guidelines must enforce one of the following:

- The names of Inport blocks and Outport blocks must match the corresponding signal or bus names.

**Exceptions:**

- When any combination of an Inport block, an Outport block, and any other block have the same block name, use a suffix or prefix on the Inport and Outport blocks.
- One common suffix / prefix is `in_` for Inport blocks and `_out` for Outport blocks.
- You may use any suffix or prefix on the ports, however, the prefix that you select must be consistent.
- Library blocks and reusable subsystems that encapsulate generic functionality.
- When the names of Inport and Outport blocks are hidden, apply a consistent naming practice for the blocks. Suggested practices include leaving the default names (for example, `Out1`), giving them

# jm\_0010: Port block names in Simulink models

---

the same name as the associated signal, or giving them a shortened or mangled version of the name of the associated signal.

## **Rationale**

- Readability
- Workflow
- Simulation

## **Last Changed**

V2.0

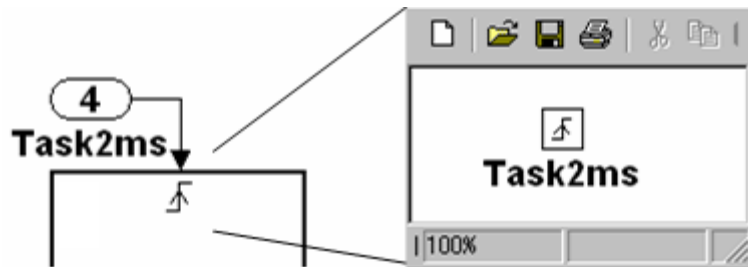
## **Model Advisor Check**

“Check for mismatches between names of ports and corresponding signals”

# jc\_0281: Naming of Trigger Port block and Enable Port block

---

<b>ID: Title</b>	jc_0281: Naming of Trigger Port block and Enable Port block
<b>Priority</b>	Strongly recommended
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	For Trigger and Enable port blocks, match the block name of the signal triggering the subsystem.



<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check Trigger and Enable block port names”

# jc\_0281: Naming of Trigger Port block and Enable Port block

---

## Signals

db\_0081: Unconnected signals, block inputs and block outputs

db\_0097: Position of labels for signals and busses

na\_0008: Display of labels on signals

na\_0009: Entry versus propagation of signal labels

The preceding guidelines apply to signals and signal labels. For background information, see “Signals and Signal Labels” on page D-3.

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.

# na\_0008: Display of labels on signals

---

<b>ID: Title</b>	na_0008: Display of labels on signals
<b>Priority</b>	Recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<ul style="list-style-type: none"><li>• A label must be displayed on a signal originating from the following blocks:<ul style="list-style-type: none"><li>▪ Inport block</li><li>▪ From block (block icon exception applies – see the Note below)</li><li>▪ Data Store Read block (block icon exception applies)</li><li>▪ Subsystem block or Stateflow chart block (block icon exception applies)</li><li>▪ Constant block (block icon exception applies)</li><li>▪ Bus Selector block (the tool forces this to happen)</li><li>▪ Demux block</li><li>▪ Selector block</li></ul></li><li>• A label must be displayed on any signal connected to the following destination blocks (directly or by way of a basic block that performs a nontransformative operation):<ul style="list-style-type: none"><li>▪ Outport block</li><li>▪ Goto block</li><li>▪ Data Store Write block</li><li>▪ Bus Creator block</li></ul></li></ul>

# na\_0008: Display of labels on signals

---

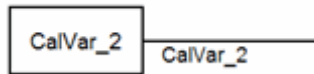
- Mux block
- Subsystem block
- Chart block

---

**Note** Block icon exception (applicable only where called out): If the signal label is visible in the originating block icon display, the connected signal does not need to have the label displayed, unless the signal label is needed elsewhere due to a destination-based rule.

---

- In addition, a label may be displayed on any other signal of interest to you or your customers.



## Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

“Check for proper labeling on signal lines”

# na\_0009: Entry versus propagation of signal labels

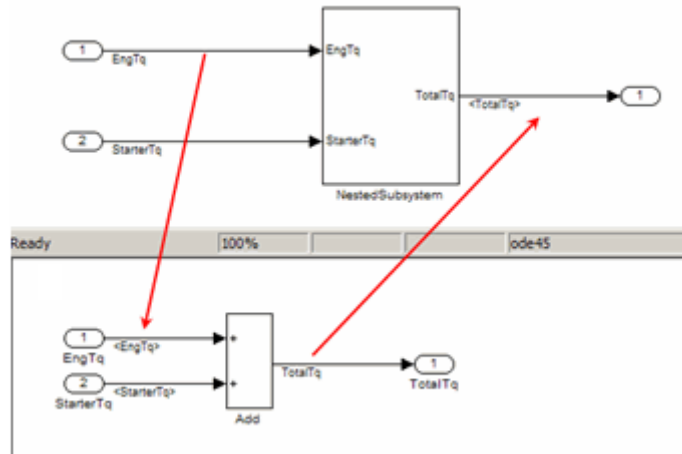
---

<b>ID: Title</b>	na_0009: Entry versus propagation of signal labels
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	na_0008: Display of labels on signals
<b>Description</b>	<p>If a label is present on a signal, the following rules define whether that label is created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the less than (&lt;) character).</p> <ul style="list-style-type: none"><li>• Any displayed signal label must be <i>entered</i> for signals that:<ul style="list-style-type: none"><li>▪ Originate from an Inport at the Root (top) Level of a model</li><li>▪ Originate from a basic block that performs a transformative operation (For the purpose of interpreting this rule only, the Bus Creator block, Mux block, and Selector block are considered to be included among the blocks that perform transformative operations.)</li></ul></li><li>• Any displayed signal label must be <i>propagated</i> for signals that:<ul style="list-style-type: none"><li>▪ Originate from an Inport block in a nested subsystem Exception: If the nested subsystem is a library subsystem, a label may be entered on the signal coming from the Inport to accommodate reuse of the library block.</li><li>▪ Originate from a basic block that performs a nontransformative operation</li><li>▪ Originate from a Subsystem or Stateflow chart block</li></ul></li></ul>



# na\_0009: Entry versus propagation of signal labels

Exception: If the connection originates from the output of a library subsystem block instance, a new label may be entered on the signal to accommodate reuse of the library block.



## Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

“Check for propagated labels on signal lines”

# db\_0097: Position of labels for signals and busses

---

<b>ID: Title</b>	db_0097: Position of labels for signals and busses
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>The labels must be visually associated with the corresponding signal and not overlap other labels, signals, or blocks.</p> <p>Labels should be located consistently below horizontal lines and close to the corresponding source or destination block.</p>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

# db\_0081: Unconnected signals, block inputs and block outputs

---

<b>ID: Title</b>	db_0081: Unconnected signals, block inputs and block outputs
<b>Priority</b>	Mandatory
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>A system must not have any:</p> <ul style="list-style-type: none"><li>• Unconnected subsystem or basic block inputs</li><li>• Unconnected subsystem or basic block outputs</li><li>• Unconnected signal lines</li></ul> <p>In addition:</p> <ul style="list-style-type: none"><li>• An otherwise unconnected input should be connected to a ground block</li><li>• An otherwise unconnected output should be connected to a terminator block</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V2.0

## db\_0081: Unconnected signals, block inputs and block outputs

---

### **Model Advisor Check**

“Check whether model has unconnected block input ports, output ports, or signal lines”

## Block Usage

hd\_0001: Prohibited Simulink sinks

jc\_0121: Use of the Sum block

jc\_0131: Use of Relational Operator block

jc\_0141: Use of the Switch block

jc\_0161: Use of Data Store Read/Write/Memory blocks

jm\_0001: Prohibited Simulink standard blocks inside controllers

na\_0002: Appropriate implementation of fundamental logical and numerical operations

na\_0003: Simple logical expressions in If Condition block

na\_0011: Scope of Goto and From blocks

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.

# na\_0003: Simple logical expressions in If Condition block

---

**ID: Title** na\_0003: Simple logical expressions in If Condition block

**Priority** Mandatory

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** A logical expression may be implemented within an If Condition block instead of building it up with logical operation blocks, if the expression contains two or fewer primary expressions. A primary expression is defined as one of the following:

- An input
- A constant
- A constant parameter
- A parenthesized expression containing no operators except zero or one instance of the following operators: < , <= , > , >= , ~= , == , ~. (See the following examples.)

## **Exception**

A logical expression may contain more than two primary expressions if both of the following are true:

- The primary expressions are all inputs
- Only one type of logical operator is present

## **Examples of Acceptable Exceptions**

- $u1 \mid u2 \mid u3 \mid u4 \mid u5$
- $u1 \ \& \ u2 \ \& \ u3 \ \& \ u4$

# na\_0003: Simple logical expressions in If Condition block

## Examples of Primary Expressions

- $u1$
- $5$
- $K$
- $(u1 > 0)$
- $(u1 \leq G)$
- $(u1 > U2)$
- $(\sim u1)$

## Examples of Acceptable Logical Expressions

- $u1 \mid u2$
- $(u1 > 0) \& (u1 < 20)$
- $(u1 > 0) \& (u2 < u3)$
- $(u1 > 0) \& (\sim u2)$

## Examples of Unacceptable Logical Expressions

$u1 \& u2 \mid u3$	(too many primary expressions)
$u1 \& (u2 \mid u3)$	(unacceptable operator within primary expression)
$(u1 > 0) \& (u1 < 20) \& (u2 > 5)$	(too many primary expressions that are not inputs)
$(u1 > 0) \& ((2 * u2) > 6)$	(unacceptable operator within primary expression)

## Rationale

- Readability
- Workflow

## na\_0003: Simple logical expressions in If Condition block

---

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable



# na\_0002: Appropriate implementation of fundamental logical and numerical operations

---

**ID: Title** na\_0002: Appropriate implementation of fundamental logical and numerical operations

**Priority** Mandatory

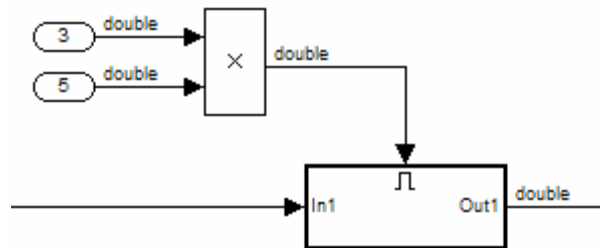
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description**

- Blocks that are intended to perform numerical operations must not be used to perform logical operations.

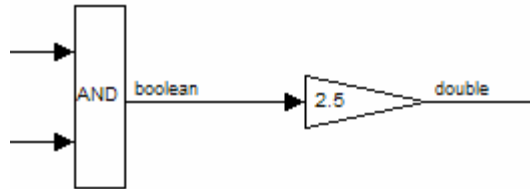


**Incorrect**

- A logical output should never be connected directly to the input of blocks that operate on numerical inputs.
- The result of a logical expression fragment should never be operated on by a numerical operator.

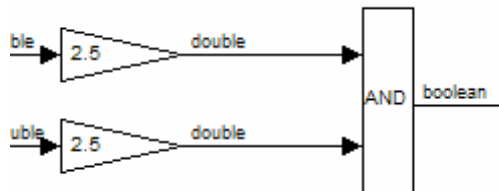
# na\_0002: Appropriate implementation of fundamental logical and numerical operations

---



## Incorrect

- Blocks that are intended to perform logical operations must not be used to perform numerical operations.
- A numerical output should never be connected to the input of blocks that operate on logical inputs.



## Incorrect

### Rationale

- Readability
- Workflow

### Last Changed

V2.0

### Model Advisor Check

Not applicable

# jm\_0001: Prohibited Simulink standard blocks inside controllers

---

**ID: Title** jm\_0001: Prohibited Simulink standard blocks inside controllers

**Priority** Mandatory

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Controller models must be designed from discrete blocks.

The following sources *are not* allowed:

- Signal Generator
- Step
- Ramp
- Sine Wave
- Repeating Sequence
- Discrete Pulse Generator
- Pulse Generator
- Chirp Signal
- Clock
- Digital Clock
- From File
- From Workspace
- Random Number
- Uniform Random Number
- Band-Limited White Noise

The following continuous blocks *are not* allowed:

- Integrator
- Derivative
- Transport Delay
- Variable Transport Delay

# jm\_0001: Prohibited Simulink standard blocks inside controllers

---

State-Space  
Transfer Fcn  
Zero-Pole

The following additional blocks *are not* allowed. The MAAB Style guide group recommends not using the following blocks. The list may be extended by individual companies.

Slider Gain  
Algebraic Constraint  
Manual Switch  
Complex to Magnitude-Angle  
Magnitude-Angle to Complex  
Complex to Real-Imag  
Real-Imag to Complex  
Hit Crossing  
Polynomial  
MATLAB Fcn  
Goto Tag Visibility  
Probe

## Rationale

- Readability
- Workflow
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

“Check for blocks that are not discrete ”

# hd\_0001: Prohibited Simulink sinks

---

<b>ID: Title</b>	hd_0001: Prohibited Simulink sinks
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>Controller models must be designed from discrete blocks.</p> <p>The following sink blocks <i>are not</i> allowed:</p> <ul style="list-style-type: none"><li>Scope</li><li>XY Graph</li><li>Display</li><li>To File</li><li>To Workspace</li><li>Stop Simulation</li><li>Floating Scope</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for prohibited sink blocks”

# na\_0011: Scope of Goto and From blocks

---

<b>ID: Title</b>	na_0011: Scope of Goto and From blocks
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	For signal flows, From and Goto blocks must use local scope.

---

**Note** Control flow signals may use global scope.

---

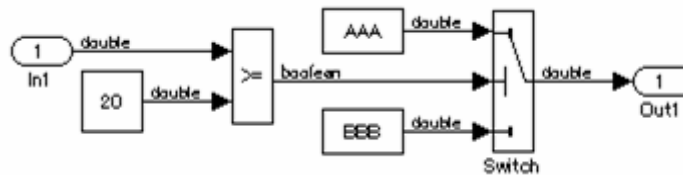
- Rationale**
- Readability
  - Workflow
  - Code Generation

**Last Changed** V2.0

**Model Advisor Check** “Check for proper scope of From and Goto blocks”

<b>ID: Title</b>	jc_0141: Use of the Switch block
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<ul style="list-style-type: none"><li>• The switch condition, input 2, must be a Boolean value.</li><li>• The block parameter, <b>Criteria for passing first input</b>, should be set to <code>u2~=0</code>.</li><li>• The block parameter, <b>Criteria for passing first input</b>, must not be set to <code>u2&gt;threshold</code> for R13 versions of MATLAB.</li></ul>

# jc\_0141: Use of the Switch block



**Function Block Parameters: Switch**

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The input 1 pass-through criteria are input 2 greater than or equal, greater than, or not equal to the threshold. The first and third input ports are data ports, and the second input port is the control port.

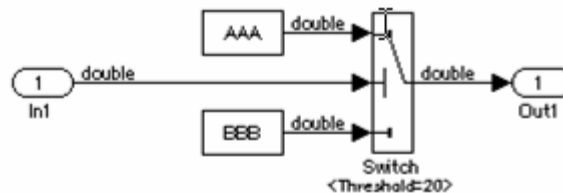
Main | Signal Data Types

Criteria for passing first input:  $u2 \sim = 0$

Threshold: 0

$u2 \geq \text{Threshold}$   
 $u2 > \text{Threshold}$   
 $u2 \sim = 0$

**Correct**



Main | Signal Data Types

Criteria for passing first input:  $u2 \geq \text{Threshold}$

Threshold: 20

**Incorrect**



**Rationale**

- Readability
- Workflow

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

“Check for proper use of Switch blocks”

# jc\_0121: Use of the Sum block

---

**ID: Title** jc\_0121: Use of the Sum block

**Priority** Recommended

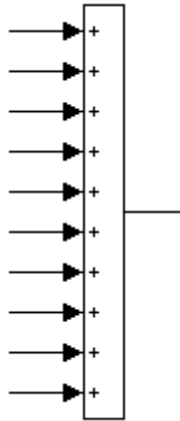
**Scope** MAAB

**MATLAB Versions** All

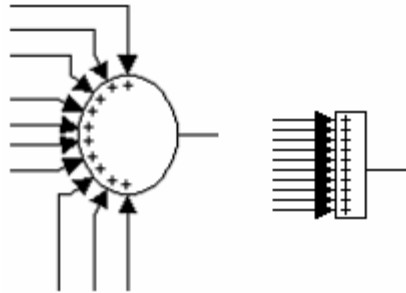
**Prerequisites** None

**Description** Sum blocks should:

- Use the “rectangular” shape.
- Be sized so that the input signals do not overlap.



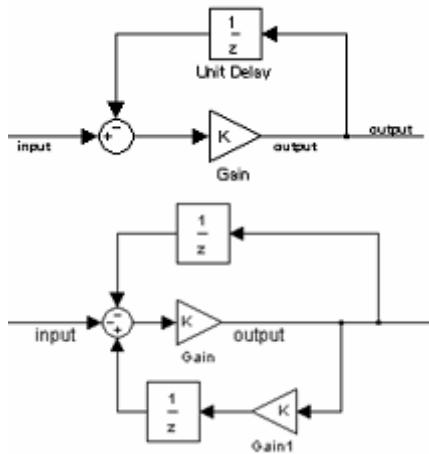
**Correct**



## Incorrect

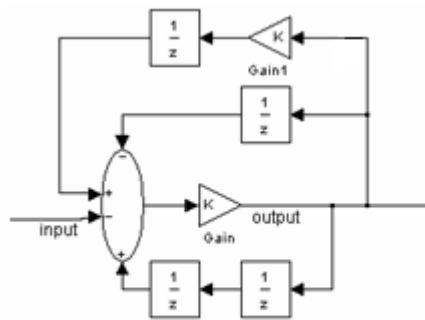
You may use the round shape in feedback loops.

- There should be no more than three inputs.
- Position the inputs at 90,180,270 degrees.
- Position the output at 0 degrees.

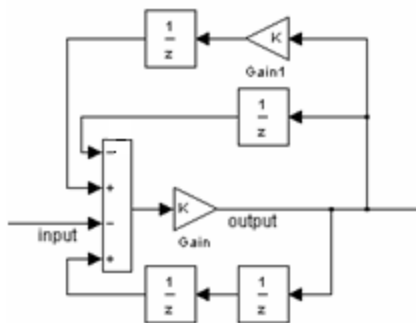


## Correct

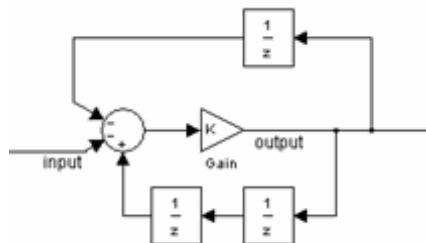
# jc\_0121: Use of the Sum block



**Incorrect**



**Correct**



**Incorrect**

<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

# jc\_0131: Use of Relational Operator block

---

**ID: Title** jc\_0131: Use of Relational Operator block

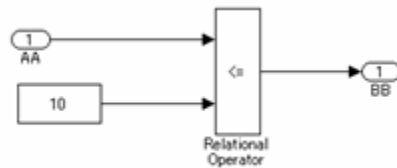
**Priority** Recommended

**Scope** J-MAAB

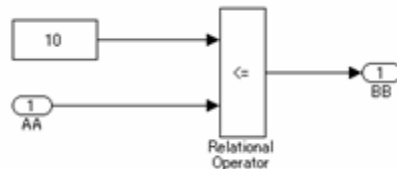
**MATLAB Versions** All

**Prerequisites** None

**Description** When the relational operator is used to compare a signal to a constant value, the constant input should be the second (lower) input signal.



**Correct**



**Incorrect**

**Rationale** Readability

**Last Changed** V2.0

**Model  
Advisor  
Check**

“Check for proper position of constants used in Relational Operator blocks”

# jc\_0161: Use of Data Store Read/Write/Memory blocks

---

<b>ID: Title</b>	jc_0161: Use of Data Store Read/Write/Memory blocks
<b>Priority</b>	Strongly recommended
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	jc_0341: Data flow layer
<b>Description</b>	<ul style="list-style-type: none"><li>• Prohibited in a data flow layer</li><li>• Allowed between subsystems running at different rates</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable



## Block Parameters

db\_0110: Tunable parameters in basic blocks

db\_0112: Indexing

na\_0010: Grouping data flows into signals

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.

# db\_0112: Indexing

---

<b>ID: Title</b>	db_0112: Indexing
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>One-based indexing [1, 2, 3,...] is for:</p> <ul style="list-style-type: none"><li>• MATLAB<ul style="list-style-type: none"><li>Workspace variables and structures</li><li>Local variables of functions written in M-code</li><li>Global variables</li></ul></li><li>• Simulink<ul style="list-style-type: none"><li>Signal vectors and matrices</li><li>Parameter vectors and matrices</li><li>S-function input and output signal vectors and matrices in M-code</li><li>S-function parameter vectors and matrices in M-code</li><li>S-function local variables in M-code</li></ul></li><li>• Stateflow<ul style="list-style-type: none"><li>Input and output signal vectors and matrices</li><li>Parameter vectors and matrices</li><li>Local variables</li></ul></li></ul> <p>Zero-based indexing [0, 1, 2, ...] is for:</p> <ul style="list-style-type: none"><li>• Simulink<ul style="list-style-type: none"><li>S-function input and output signal vectors and matrices in C</li></ul></li></ul>

S-function input parameters in C

S-function parameter vectors and matrices in C

S-function local variables in C

- Stateflow

Custom variables and structures in C

- C code

Local variables and structures

Global variables

## **Rationale**

- Readability
- Workflow
- Code Generation

## **Last Changed**

V2.0

## **Model Advisor Check**

“Check for blocks not using one-based indexing”

# na\_0010: Grouping data flows into signals

---

**ID: Title** na\_0010: Grouping data flows into signals

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** **Vectors**

The individual scalar signals composing a vector must have common functionality, data types, dimensions, and units. The most common example of a vector signal is sensor or actuator data that is grouped into an array indexed by location. The output of a Mux block must always be a vector. The inputs to a Mux block must always be scalars.

**Busses**

Signals that do not meet criteria for use as a vector, as previously described, must only be grouped into bus signals. Use Bus Selector blocks only with a bus signal input; do not use them to extract scalar signals from vector signals.

**Examples**

Some examples of vector signals include:

Vector type	Size
Row vector	[1 n]
Column vector	[n 1]
Wheel speed vector	[1 Number of wheels]
Cylinder vector	[1 Number of cylinders]

# na\_0010: Grouping data flows into signals

Vector type	Size
Position vector based on 2D coordinates	[1 2]
Position vector based on 3D coordinates	[1 3]

Some examples of bus signals include:

Bus type	Elements
Sensor Bus	Force Vector [F <sub>x</sub> , F <sub>y</sub> , F <sub>z</sub> ]
	Position
	Wheel Speed Vector [ $\theta_{lf}$ , $\theta_{rf}$ , $\theta_{lr}$ , $\theta_{rr}$ ]
	Acceleration
	Pressure
Controller Bus	Sensor Bus
	Actuator Bus
Serial Data Bus	Coolant Temperature
	Engine Speed, Passenger Door Open

## Rationale

- Readability
- Workflow

## Last Changed

V2.0

## Model Advisor Check

“Check for proper use of signal buses and Mux block usage”

# db\_0110: Tunable parameters in basic blocks

---

**ID: Title** db\_0110: Tunable parameters in basic blocks

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** To ensure that a parameter is tunable, enter it in the basic block:

- Without any expression.
- Without a data type conversion.
- Without selection of rows or columns.

tunable\_parameter\_value } tunable\_parameter\_vector } tunable\_parameter\_array }

## Correct

tunable\_parameter\_value\*2 } tunable\_parameter\_vector\*3 } tunable\_parameter\_array\*3 }  
int16(tunable\_parameter\_value) } tunable\_parameter\_vector(2) } tunable\_parameter\_array(1,1) }

## Incorrect

- Rationale**
- Readability
  - Workflow
  - Code Generation

**Last Changed** V2.0

# db\_0110: Tunable parameters in basic blocks

---

## **Model Advisor Check**

“Check whether tunable parameters specify expressions, data type conversions, or indexing operations”

## Simulink Patterns

db\_0114: Simulink patterns for  
If-then-else-if constructs

db\_0115: Simulink patterns for  
case constructs

db\_0116: Simulink patterns for  
logical constructs with logical  
blocks

db\_0117: Simulink patterns for  
vector signals

jc\_0111: Direction of Subsystem

jc\_0351: Methods of initialization

na\_0012: Use of Switch vs.  
If-Then-Else Action Subsystem

The preceding guidelines illustrate sample patterns used in Simulink diagrams. As such, the patterns normally would be part of a much larger Simulink diagram.

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.



# na\_0012: Use of Switch vs. If-Then-Else Action Subsystem

**ID: Title** na\_0012: Use of Switch vs. If-Then-Else Action Subsystem

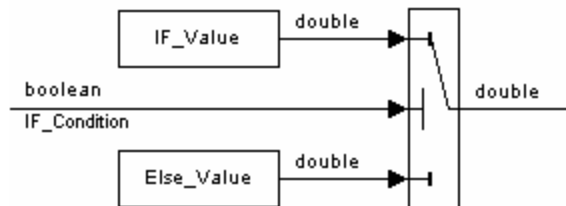
**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

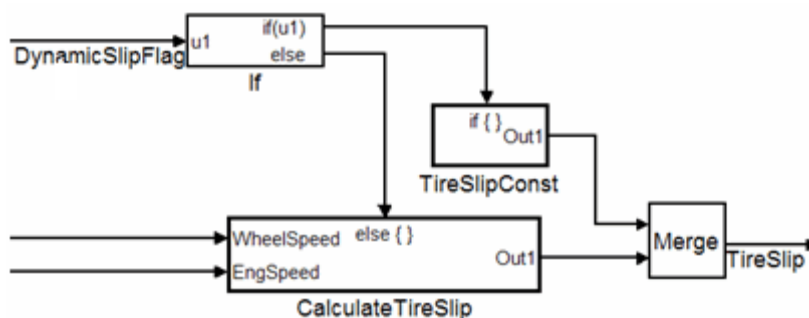
**Description** The **Switch** block should be used for modeling simple *if-then-else* structures, if the associated *then* and *else* actions involve only the assignment of constant values.



The **if-then-else** action subsystem construct:

- Should be used for modeling *if-then-else* structures, if the associated *then* and/or *else* actions require complicated computations. This maximizes simulation efficiency and the efficiency of generated code. (Note that even a basic block, for example a table lookup, may require fairly complicated computations.)

# na\_0012: Use of Switch vs. If-Then-Else Action Subsystem



- Must be used for modeling *if-then-else* structures, if the purpose of the construct is to avoid an undesirable numerical computation, such as division by zero.
- Should be used for modeling *if-then-else* structures, if the explicit or implied *then* or the *else* action is just to hold the associated output values.

In other cases, the degree of complexity of the *then* and/or *else* action computations and the intelligence of the Simulink simulation and code generation engines determine the appropriate construct.

These statements also apply to more complicated nested and cascaded *if-then-else* structures and *case* structure implementations.

## Rationale

- Readability
- Workflow

## Last Changed

V2.0

## Model Advisor Check

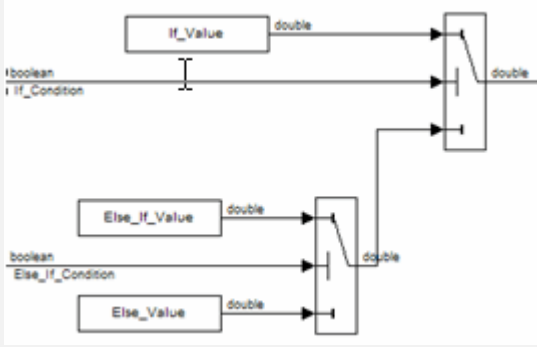
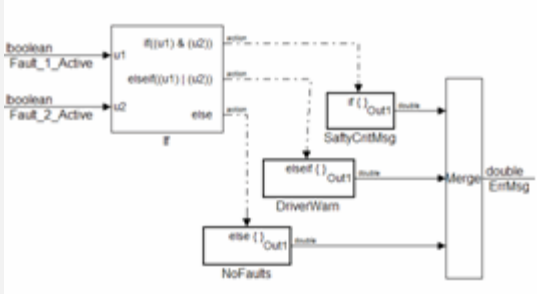
Not applicable

# db\_0114: Simulink patterns for If-then-else-if constructs

---

<b>ID: Title</b>	db_0114: Simulink patterns for If-then-else-if constructs
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	Use the following patterns for If-then-else-if constructs within a Simulink model:

# db\_0114: Simulink patterns for If-then-else-if constructs

Equivalent Functionality	Simulink Pattern
<p>if then else if with blocks</p> <pre> if (If_Condition) {   output_signal = If_Value; } else if (Else_If_Condition) {   output_signal =   Else_If_Value; } else {   output_signal =   Else_Value; } </pre>	
<p>if then else if with if/then/else subsystems</p> <pre> if(Fault_1_Active &amp;   Fault_2_Active) {   ErrMsg = SaftyCrit; } else if (Fault_1_Active     Fault_2_Active) {   ErrMsg = DriveWarn; } else {   ErrMsg = NoFaults; } </pre>	

## Rationale

- Readability
- Workflow

# db\_0114: Simulink patterns for If-then-else-if constructs

---

- Code Generation

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# db\_0115: Simulink patterns for case constructs

**ID: Title** db\_0115: Simulink patterns for case constructs

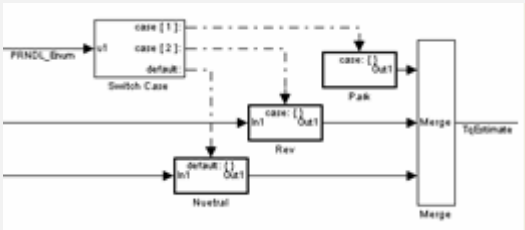
**Priority** Strongly recommended

**Scope** MAAB

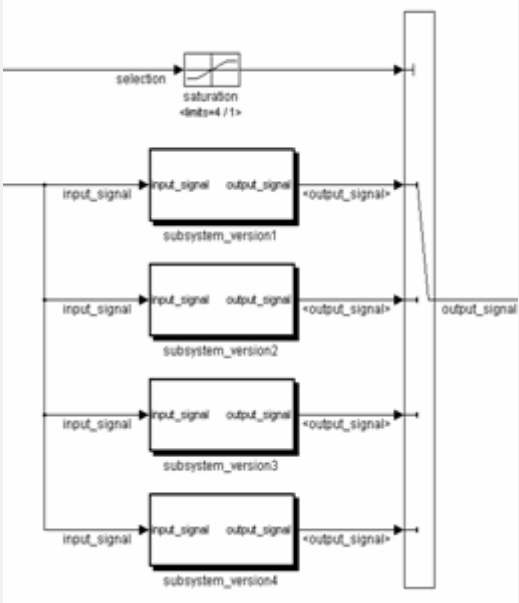
**MATLAB Versions** All

**Prerequisites** None

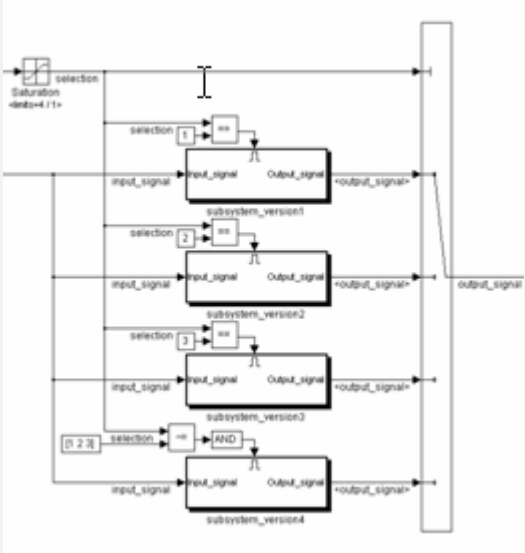
**Description** Use the following patterns for case constructs within a Simulink model:

Equivalent Functionality	Simulink Pattern
<p>case with Switch Case block</p> <pre> switch (PRNDL_Enum) { case 1   TqEstimate = ParkV;   break; case 2   TqEstimate = RevV;   break; default   TqEstimate = NeutralV;   break; }                     </pre>	 <p>The diagram illustrates the Simulink implementation of the provided MATLAB code. It features a 'Switch Case' block with three inputs: 'case [1]', 'case [2]', and 'default'. The 'case [1]' input is connected to a 'Park' block, which outputs 'Out1'. The 'case [2]' input is connected to a 'Rev' block, which outputs 'Out1'. The 'default' input is connected to a 'Neutral' block, which outputs 'Out1'. All three 'Out1' outputs are fed into a 'Merge' block, which produces the final 'TqEstimate' output.</p>

# db\_0115: Simulink patterns for case constructs

Equivalent Functionality	Simulink Pattern
<p>case with subsystems</p> <pre>output_version1 = function_version1(input_signal); output_version2 = function_version2(input_signal); output_version3 = function_version3(input_signal); output_version4 = function_version4(input_signal); switch (selection) { case 1: output_signal = output_version1; break; case 2: output_signal = output_version2; break; case 3: output_signal = output_version3; break; case 4: output_signal = output_version4;</pre>	 <p>The diagram illustrates a Simulink pattern for a case construct. It features a 'selection' input block that feeds into a 'saturation' block (labeled 'saturation &lt;ints=4 / 1&gt;'). The output of the saturation block is connected to a switch block. The switch block has four inputs, each labeled 'input_signal', which are connected to four separate subsystem blocks: 'subsystem_version1', 'subsystem_version2', 'subsystem_version3', and 'subsystem_version4'. Each subsystem block has an 'input_signal' input and an 'output_signal' output. The outputs of all four subsystems are connected to a single 'output_signal' block. The switch block selects between these four outputs based on the 'selection' input.</p>

# db\_0115: Simulink patterns for case constructs

Equivalent Functionality	Simulink Pattern
<pre> }  case with enabled subsystems  switch (selection) { case 1: output_version1 = function_version1(input_signal); output_signal = output_version1; break; case 2: output_version2 = function_version2(input_signal); output_signal = output_version2; break; case 3: output_version3 = function_version3(input_signal); output_signal = output_version3; break; default: output_version4 = function_version4(input_signal); output_signal = output_version4; } </pre>	

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V2.0



## db\_0115: Simulink patterns for case constructs

---

**Model  
Advisor  
Check**

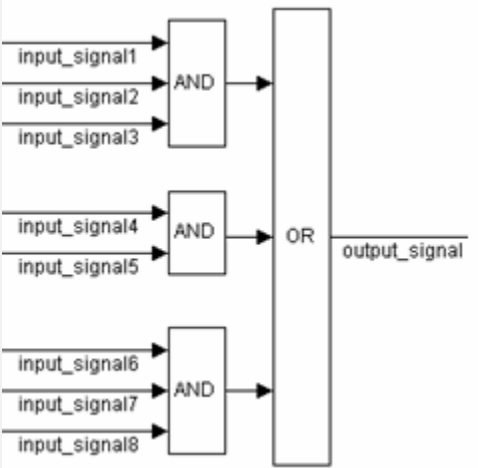
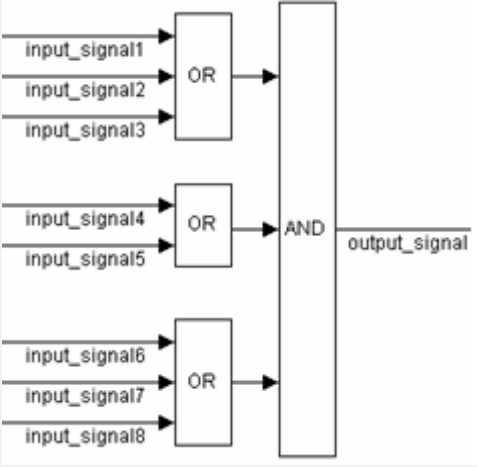
Not applicable

# db\_0116: Simulink patterns for logical constructs with logical blocks

---

<b>ID: Title</b>	db_0116: Simulink patterns for logical constructs with logical blocks
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	Use the following patterns for logical combinations within Simulink:

# db\_0116: Simulink patterns for logical constructs with logical blocks

Equivalent Functionality	Simulink Pattern
Combination of logical signals: conjunctive	 <p>The diagram illustrates a conjunctive combination of logical signals. It features three AND blocks in a vertical column. The top AND block has three inputs labeled 'input_signal1', 'input_signal2', and 'input_signal3'. The middle AND block has two inputs labeled 'input_signal4' and 'input_signal5'. The bottom AND block has three inputs labeled 'input_signal6', 'input_signal7', and 'input_signal8'. The outputs of these three AND blocks are connected to a single OR block. The output of the OR block is labeled 'output_signal'.</p>
Combination of logical signals: disjunctive	 <p>The diagram illustrates a disjunctive combination of logical signals. It features three OR blocks in a vertical column. The top OR block has three inputs labeled 'input_signal1', 'input_signal2', and 'input_signal3'. The middle OR block has two inputs labeled 'input_signal4' and 'input_signal5'. The bottom OR block has three inputs labeled 'input_signal6', 'input_signal7', and 'input_signal8'. The outputs of these three OR blocks are connected to a single AND block. The output of the AND block is labeled 'output_signal'.</p>

## Rationale

- Readability

# db\_0116: Simulink patterns for logical constructs with logical blocks

---

- Workflow
- Verification and Validation

**Last  
Changed**

V1.0

**Model  
Advisor  
Check**

Not applicable

# db\_0117: Simulink patterns for vector signals

**ID: Title** db\_0117: Simulink patterns for vector signals

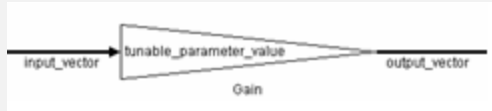
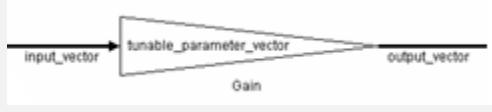
**Priority** Strongly recommended

**Scope** MAAB

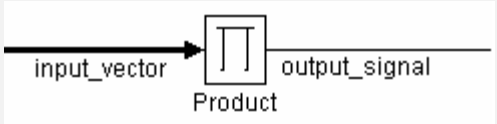
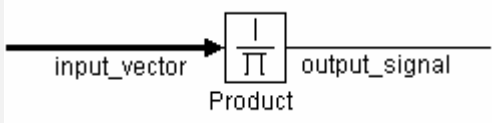
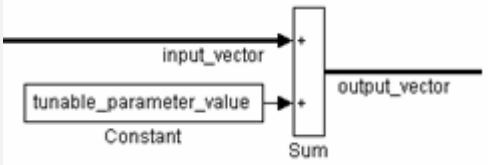
**MATLAB Versions** All

**Prerequisites** None

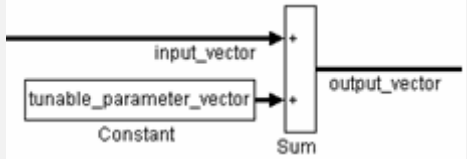
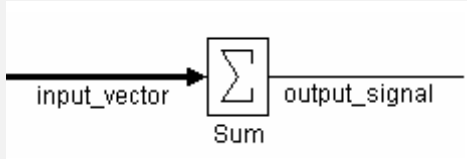
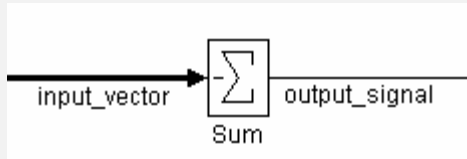
**Description** Use the following patterns for vector signals within a Simulink model:

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre>for (i=0; i&gt;input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_value; }</pre>	 A Simulink Gain block diagram. An input vector signal enters a trapezoidal block labeled 'tunable_parameter_value' with 'Gain' written below it. The output is an output vector signal.
<p>Vector loop</p> <pre>for (i=0; i&gt;input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_vector(i); }</pre>	 A Simulink Gain block diagram. An input vector signal enters a trapezoidal block labeled 'tunable_parameter_vector' with 'Gain' written below it. The output is an output vector signal.

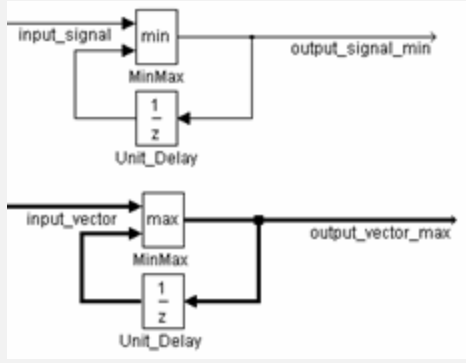
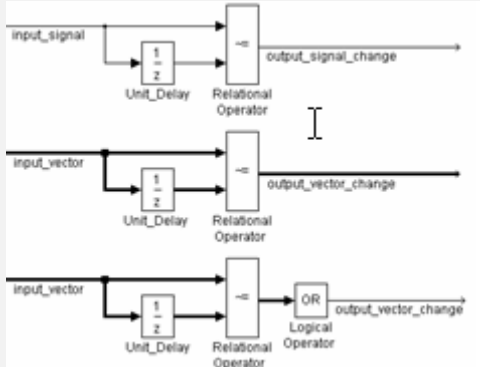
# db\_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i&gt;input_vector_size; i++) { output_signal = output_signal * input_vector(i); }                     </pre>	 <p>The diagram shows a Simulink Product block. An arrow labeled 'input_vector' enters the block from the left. An arrow labeled 'output_signal' exits the block to the right. The block is labeled 'Product' below it.</p>
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i&gt;input_vector_size; i++) { output_signal = output_signal / input_vector(i); }                     </pre>	 <p>The diagram shows a Simulink Product block with a division symbol (  over Π) inside. An arrow labeled 'input_vector' enters the block from the left. An arrow labeled 'output_signal' exits the block to the right. The block is labeled 'Product' below it.</p>
<p>Vector loop</p> <pre> for (i=0; i&gt;input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_value; }                     </pre>	 <p>The diagram shows a Simulink Sum block. Two arrows enter the block from the left: one labeled 'input_vector' and one labeled 'tunable_parameter_value' which is enclosed in a box labeled 'Constant'. An arrow labeled 'output_vector' exits the block to the right. The block is labeled 'Sum' below it.</p>

# db\_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre>for (i=0; i&gt;input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_vector(i); }</pre>	 <p>The diagram shows a Simulink Sum block with two inputs. The top input is labeled 'input_vector' and the bottom input is labeled 'tunable_parameter_vector' with 'Constant' written below it. The output of the block is labeled 'output_vector'.</p>
<p>Vector loop:</p> <pre>output_signal = 0; for (i=0; i&gt;input_vector_size; i++) { output_signal = output_signal + input_vector(i); }</pre>	 <p>The diagram shows a Simulink Sum block with a single input labeled 'input_vector'. The output of the block is labeled 'output_signal'.</p>
<p>Vector loop:</p> <pre>output_signal = 0; for (i=0; i&gt;input_vector_size; i++) { output_signal = output_signal - input_vector(i); }</pre>	 <p>The diagram shows a Simulink Sum block with a single input labeled 'input_vector'. The output of the block is labeled 'output_signal'.</p>

# db\_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
<p>Minimum or maximum of a signal or a vector over time:</p>	
<p>Change event of a signal or a vector:</p>	

## Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation



# db\_0117: Simulink patterns for vector signals

---

**Last  
Changed**

V1.0

**Model  
Advisor  
Check**

Not applicable

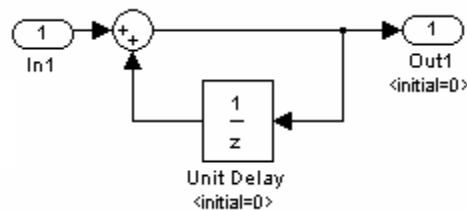
# jc\_0351: Methods of initialization

---

<b>ID: Title</b>	jc_0351: Methods of initialization
<b>Priority</b>	Recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	db_0140: Display of basic block parameters

## Description **Simple Initialization**

- You may use blocks, such as the Unit Delay block, which have an initial value field, to set simple initial values.
- To determine if the initial value should be displayed, see MAAB Guideline db\_0140: Display of basic block parameters.



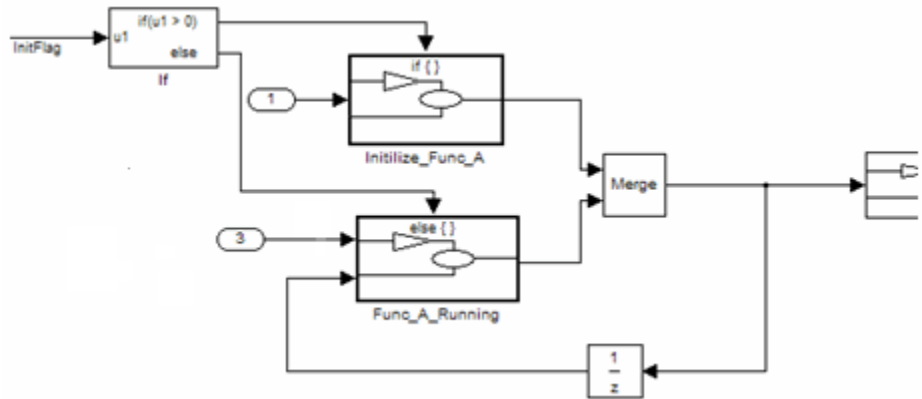
## Example

### Initialization that Requires Computation

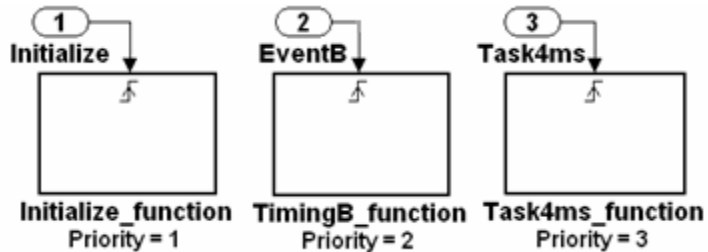
The following rules apply for complex initialization:

- The initialization should be performed in a separate subsystem.
- The initialization subsystem should have a name that indicates that initialization is performed by the subsystem.

Complex initialization may be done at a local level (Example A), at a global level (Example B), or a combination of local and global.



**Example A**



**Example B**

**Rationale**

Workflow

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# jc\_0111: Direction of Subsystem

**ID: Title** jc\_0111: Direction of Subsystem

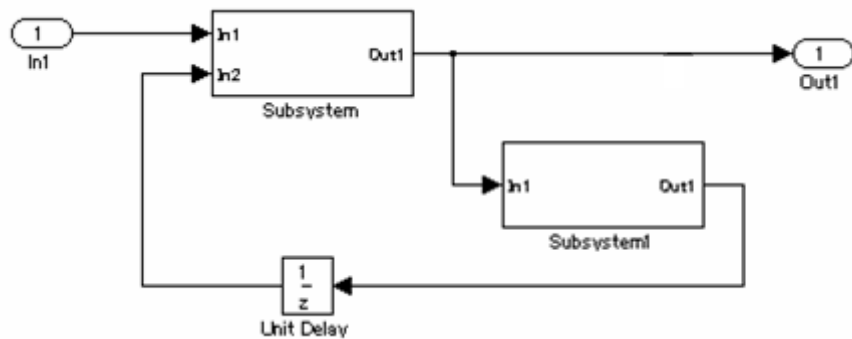
**Priority** Strongly recommended

**Scope** J-MAAB

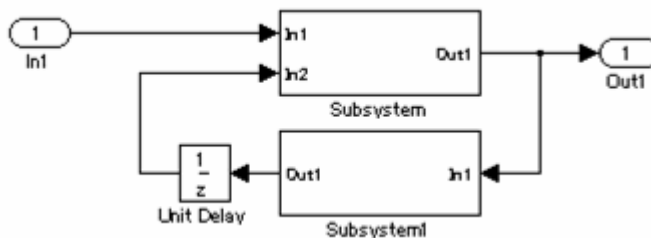
**MATLAB Versions** All

**Prerequisites** None

**Description** Subsystem must not be reversed.



**Correct**



**Incorrect**

<b>Rationale</b>	Readability
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for direction of subsystem blocks”

## **jc\_0111: Direction of Subsystem**

---

# Stateflow

---

- “Chart Appearance” on page 6-2
- “Stateflow Data and Operations” on page 6-20
- “Events” on page 6-39
- “Statechart Patterns” on page 6-43
- “Flowchart Patterns” on page 6-49

## **Chart Appearance**

db\_0123: Stateflow port names

db\_0129: Stateflow transition  
appearance

db\_0132: Transitions in Flowcharts

db\_0133: Use of patterns for  
Flowcharts

db\_0137: States in state machines

jc\_0501: Format of entries in a State  
block

jc\_0511: Setting the return value  
from a graphical function

jc\_0521: Use of the return value  
from graphical functions

jc\_0531: Placement of the default  
transition



## db\_0123: Stateflow port names

---

<b>ID: Title</b>	db_0123: Stateflow port names
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>The name of a Stateflow input or output should be the same as the corresponding signal.</p> <p><b>Exception:</b> Reusable Stateflow blocks may have different port names.</p>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li></ul>
<b>Last Changed</b>	V1.0
<b>Model Advisor Check</b>	“Check for mismatches between Stateflow ports and associated signal names”

# db\_0129: Stateflow transition appearance

---

**ID: Title** db\_0129: Stateflow transition appearance

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB  
Versions** All

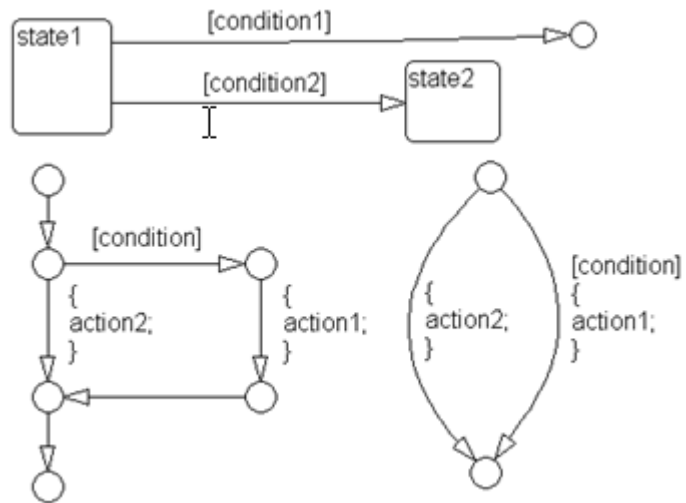
**Prerequisites** None

**Description** Transitions in Stateflow:

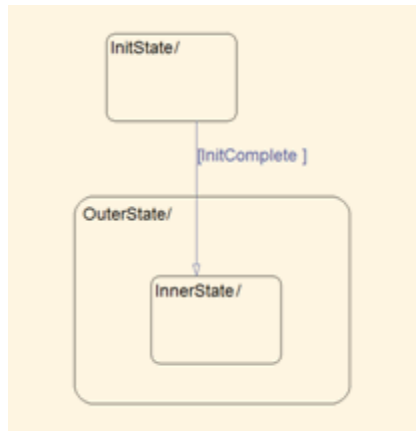
- Do not cross each other, if possible
- Are not drawn one upon the other
- Do not cross any states, junctions, or text fields
- Are allowed if transition is to an internal state

Transition labels may be visually associated to the corresponding transition.

# db\_0129: Stateflow transition appearance

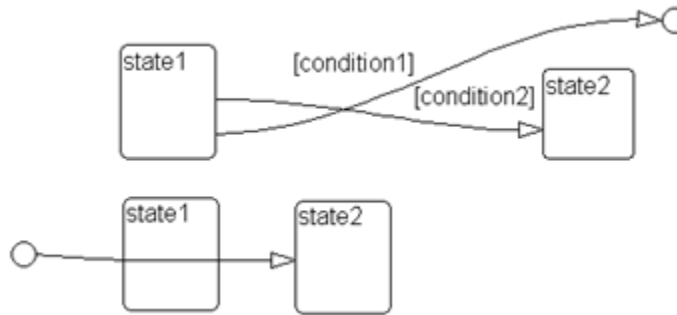


**Correct**



# db\_0129: Stateflow transition appearance

---



**Incorrect**

## Rationale

- Readability
- Workflow

## Last Changed

V2.0

## Model Advisor Check

Not applicable

<b>ID: Title</b>	db_0137: States in state machines
<b>Priority</b>	Mandatory
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	db_0149: Flowchart patterns for condition actions
<b>Description</b>	<p>In state machines:</p> <ul style="list-style-type: none"><li>• At least two exclusive states exist</li><li>• A state must have multiple substates</li><li>• The initial state of a hierarchical level with exclusive states is clearly defined by a default transition</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check for exclusive states, default states, and substate validity”

# db\_0133: Use of patterns for Flowcharts

---

<b>ID: Title</b>	db_0133: Use of patterns for Flowcharts
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>A Flowchart is built with the help of Flowchart patterns (for example, if-then-else, for loop, and so on):</p> <ul style="list-style-type: none"><li>• The data flow is oriented from the top to the bottom. •</li><li>• Patterns are connected with empty transitions.</li></ul>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V1.0
<b>Model Advisor Check</b>	Not applicable

**ID: Title** db\_0132: Transitions in Flowcharts

**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

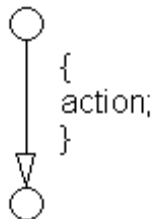
**Prerequisites** None

**Description** The following rules apply to transitions in Flowcharts:

- Conditions are drawn on the horizontal.
- Actions are drawn on the vertical.
- Loop constructs are intentional exceptions to this rule.
- Transitions have a condition, a condition action, or an empty transition.



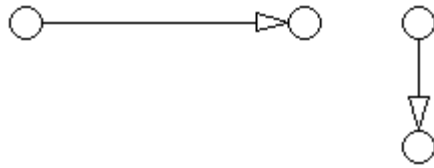
**Transition with Condition**



**Transition with Condition Action**

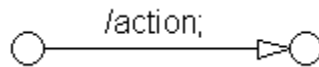
# db\_0132: Transitions in Flowcharts

---



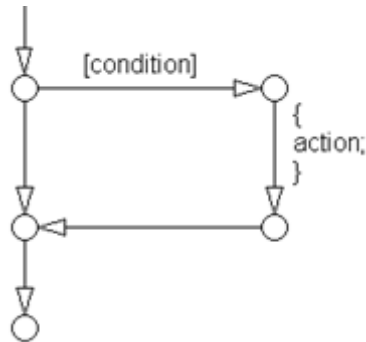
## Empty Transition

Transition actions are not used in Flowcharts. Transition actions are only valid when used in transitions between states in a state machine, otherwise they are not activated because of the inherent dependency on a valid state to state transition to activate them.

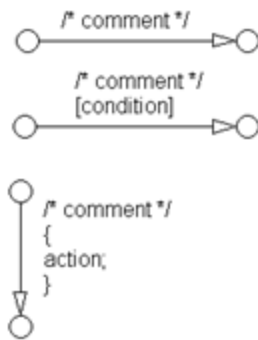


## Transition Action

At every junction, except for the last junction of a flow diagram, exactly one unconditional transition begins. Every decision point (junction) must have a default path.







## Transitions with Comments

### Rationale

- Readability
- Workflow
- Verification and Validation

### Last Changed

V2.0

### Model Advisor Check

“Check transition orientations in flow charts”

# jc\_0501: Format of entries in a State block

---

**ID: Title** jc\_0501: Format of entries in a State block

**Priority** Recommended

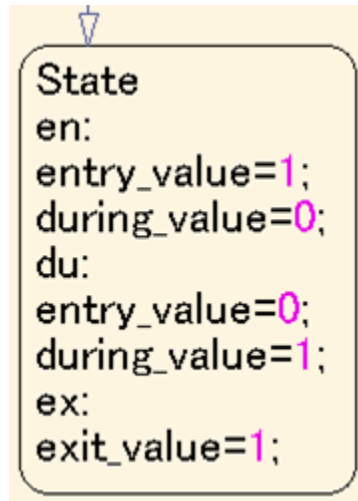
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** A new line should be:

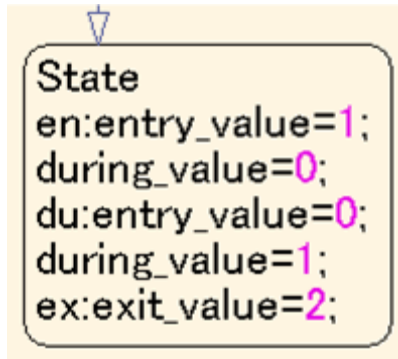
- Started after the entry (en), during (du), and exit (ex) statements.
- Started after the completion of an assignment statement “;”.



```
State
en:
entry_value=1;
during_value=0;
du:
entry_value=0;
during_value=1;
ex:
exit_value=1;
```

**Correct**

# jc\_0501: Format of entries in a State block

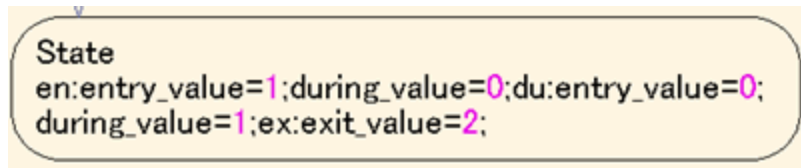


A diagram showing a state block with a yellow background and a black border. The text inside is: State, en:entry\_value=1;, during\_value=0;, du:entry\_value=0;, during\_value=1;, ex:exit\_value=2;. The text is not wrapped, and there are no new lines after the semicolons. A blue arrow points to the top of the block.

```
State
en:entry_value=1;
during_value=0;
du:entry_value=0;
during_value=1;
ex:exit_value=2;
```

## Incorrect

Failed to start a new line after en, du, and ex.



A diagram showing a state block with a yellow background and a black border. The text inside is: State, en:entry\_value=1;during\_value=0;du:entry\_value=0;, during\_value=1;ex:exit\_value=2;. The text is not wrapped, and there are no new lines after the semicolons. A blue arrow points to the top of the block.

```
State
en:entry_value=1;during_value=0;du:entry_value=0;
during_value=1;ex:exit_value=2;
```

## Incorrect

Failed to start a new line after the completion of an assignment statement “;”.

## Rationale

Readability

## Last Changed

V2.0

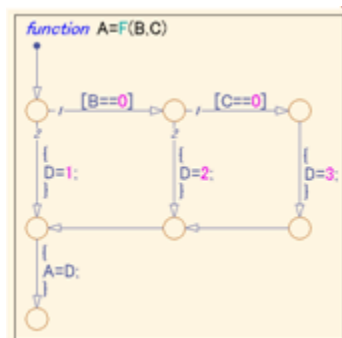
## Model Advisor Check

“Check for entry format in state blocks”

# jc\_0511: Setting the return value from a graphical function

---

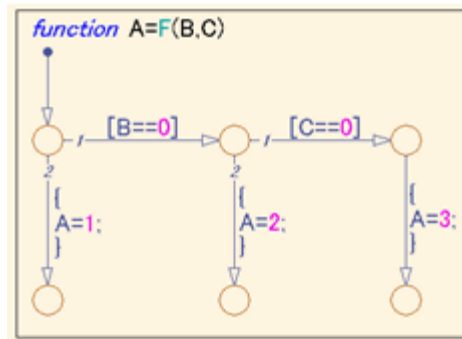
<b>ID: Title</b>	jc_0511: Setting the return value from a graphical function
<b>Priority</b>	Mandatory
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	The return value from a graphical function must be set in only one place.



## Correct

Return value A is set in one place.

# jc\_0511: Setting the return value from a graphical function



## Incorrect

Return value A is set in multiple places.

## Rationale

- Workflow
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

“Check setting Stateflow graphical function return value”

# jc\_0531: Placement of the default transition

---

**ID: Title** jc\_0531: Placement of the default transition

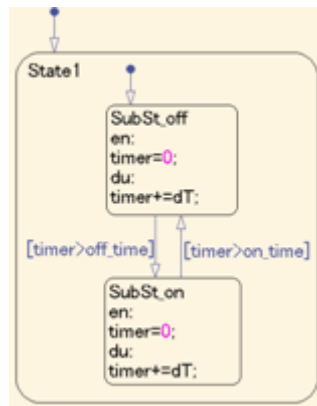
**Priority** Recommended

**Scope** J-MAAB

**MATLAB Versions** All

**Prerequisites** None

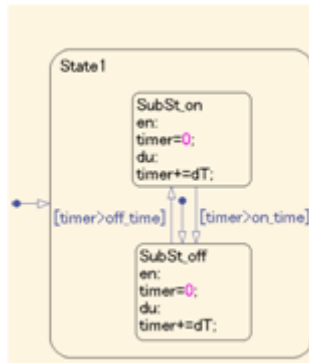
- Description**
- Default transition is connected at the top of the state.
  - The destination state of the default transition is put above the other states in the same hierarchy.



## Correct

- The default transition is connected at the top of the state.
- The destination state of the default transition is put above the other states in the same hierarchy.

# jc\_0531: Placement of the default transition



## Incorrect

- Default transition is connected at the side of the state (State 1).
- The destination state of the default transition is lower than the other states in the same hierarchy (SubSt\_off).

## Rationale

Readability

## Last Changed

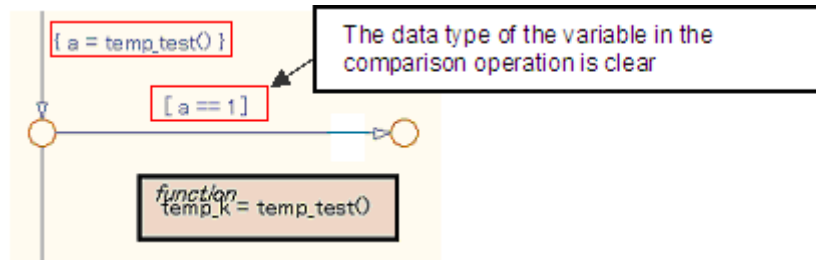
V2.0

## Model Advisor Check

“Check default transition placement in Stateflow charts”

# jc\_0521: Use of the return value from graphical functions

<b>ID: Title</b>	jc_0521: Use of the return value from graphical functions
<b>Priority</b>	Recommended
<b>Scope</b>	J-MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	The return value from a graphical function should not be used directly in a comparison operation.

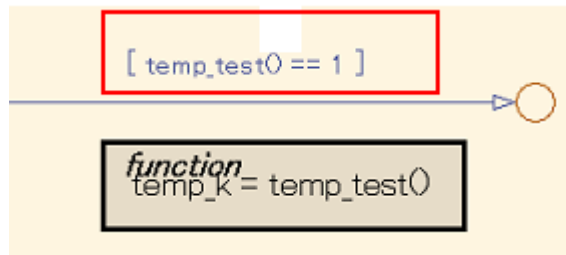


## Correct

An intermediate variable is used in the conditional expression after the assignment of the return value from the function `temp_test` to the intermediate variable `a`.



# jc\_0521: Use of the return value from graphical functions



## Incorrect

Return value of the function `temp_test` is used in the conditional expression.

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

Not applicable

# jc\_0521: Use of the return value from graphical functions

---

## Stateflow Data and Operations

na\_0001: Bitwise Stateflow operators

# na\_0001: Bitwise Stateflow operators

---

<b>ID: Title</b>	na_0001: Bitwise Stateflow operators
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>The bitwise Stateflow operators (&amp;,  , and ^) should not be used in Stateflow charts unless you want bitwise operations. If you do not want bitwise operations, enable <b>Enable C-bit operations</b>.</p> <p>To enable Enable C bit operations:</p> <ol style="list-style-type: none"><li>1 Select <b>File &gt; Chart Properties</b> .</li><li>2 Select <b>Enable C-bit operations</b>.</li></ol> <p><b>Correct</b></p> <p>Use &amp;&amp; and II for Boolean operation. Use &amp; and I for bit operation.</p> <p><b>Incorrect</b></p> <p>Use &amp; and I for Boolean operation.</p>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Simulation</li><li>• Code Generation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	Not applicable

# jc\_0451: Use of unary minus on unsigned integers in Stateflow

---

**ID: Title** jc\_0451: Use of unary minus on unsigned integers in Stateflow

**Priority** Recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Do not perform unary minus on unsigned integers.

```
si16_var1=-si16_var2;
```

Name	Data Type
si_var2	int16

**Correct**

```
ui16_var1=-ui16_var2;
```

Name	Data Type
ui_var2	uint16

**Incorrect**

- Rationale**
- Readability
  - Workflow
  - Code Generation

**Last Changed** V2.0

**Model Advisor Check** “Check for use of tunable parameters in Stateflow”

# na\_0013: Comparison operation in Stateflow

**ID: Title** na\_0013: Comparison operation in Stateflow

**Priority** Recommended

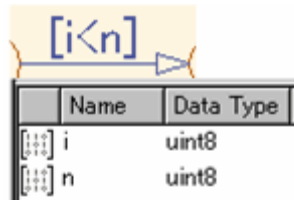
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

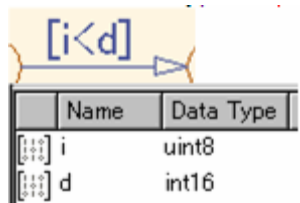
**Description**

- Comparisons should be made only between variables of the same data type.
- If comparisons are made between variables of different data types, the variables need to be explicitly type cast to matching data types.



**Correct**

Same data type in “i” and “n”



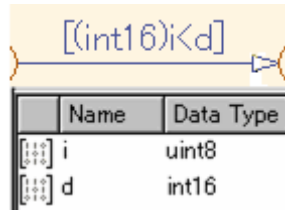
**Incorrect**

Different data type in “i” and “d”

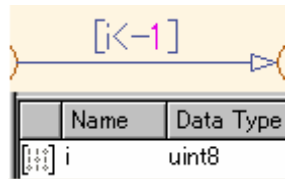
# na\_0013: Comparison operation in Stateflow

---

Do not make comparisons between unsigned integers and negative numbers.



**Correct**



**Incorrect**

## Rationale

- Workflow
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

Not applicable

# db\_0122: Stateflow and Simulink interface signals and parameters

---

<b>ID: Title</b>	db_0122: Stateflow and Simulink interface signals and parameters
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	A Chart uses strong data typing with Simulink and requires that you select the <b>Use Strong Data Typing with Simulink I/O</b> parameter.
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check interface signals and parameters”

# db\_0125: Scope of internal signals and local auxiliary variables

**ID: Title** db\_0125: Scope of internal signals and local auxiliary variables

**Priority** Strongly recommended

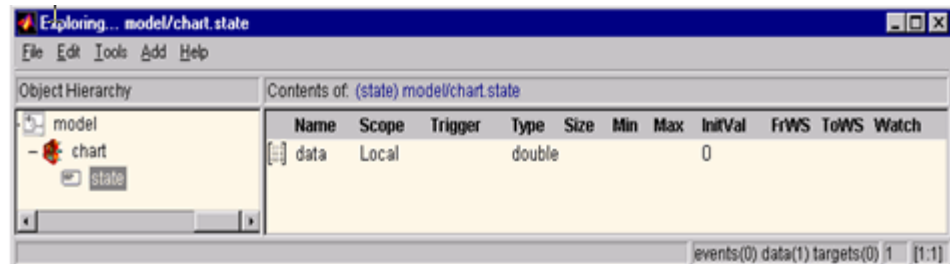
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** Internal signals and local auxiliary variables are "Local data" in Stateflow:

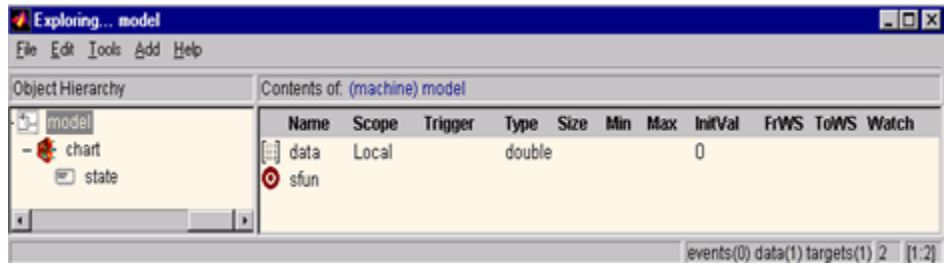
- All local data of a Stateflow block must be defined on the chart level or below the Object Hierarchy.
- No local variables may exist on the machine level (that is, no interaction should occur between local data in different charts).
- Parameters and constants are allowed at the machine level.



**Correct**



# db\_0125: Scope of internal signals and local auxiliary variables



**Incorrect**

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V2.0

## Model Advisor Check

“Check interface signals and parameters”

# jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow

**ID: Title** jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow

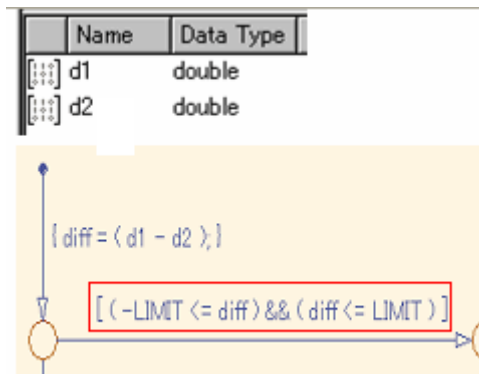
**Priority** Recommended

**Scope** MAAB

**MATLAB Versions** All

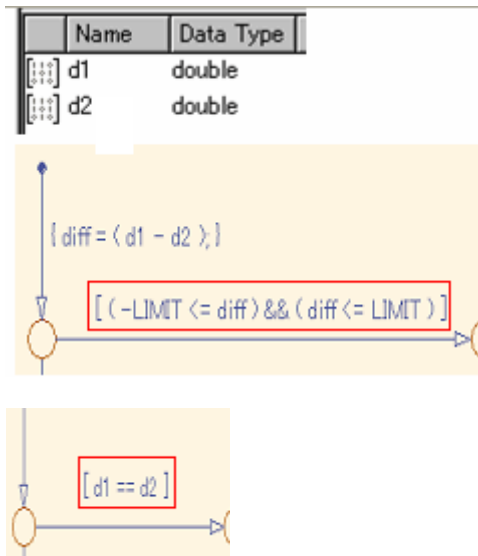
**Prerequisites** None

- Description**
- Do not use hard equality comparisons (`Var1 == Var2`) with two floating-point numbers.
  - If a hard comparison is required, a margin of error should be defined and used in the comparison (`LIMIT`, in the example).
  - Hard equality comparisons may be done between two integer data types.



**Correct**

# jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow



**Incorrect**

**Rationale**

- Workflow
- Verification and Validation
- Code Generation

**Last Changed**

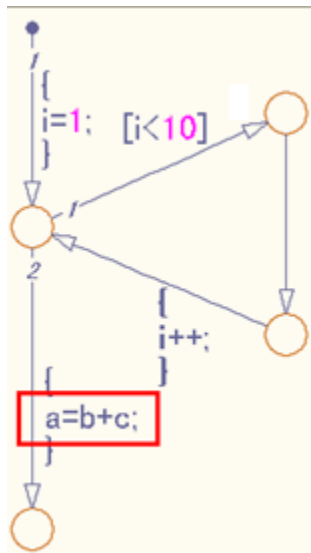
V2.0

**Model Advisor Check**

Not applicable

# jc\_0491: Reuse of variables within a single Stateflow scope

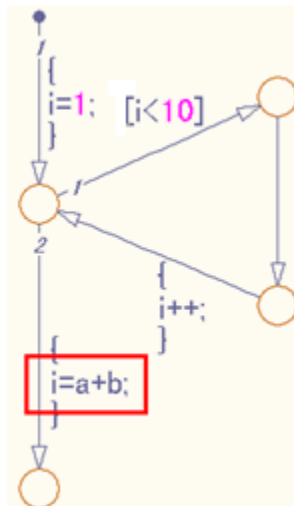
<b>ID: Title</b>	jc_0491: Reuse of variables within a single Stateflow scope
<b>Priority</b>	Recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	The same variable should not have multiple meanings (usages) within a single Stateflow scope.



## Correct

Variable of loop counter must not be used other than loop counter.

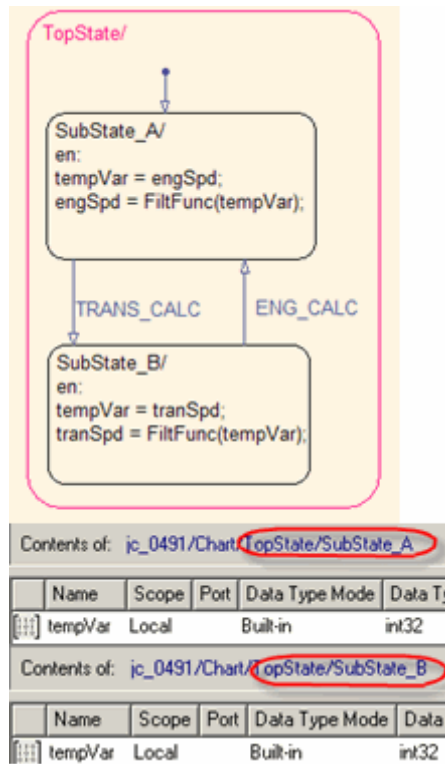
# jc\_0491: Reuse of variables within a single Stateflow scope



## Incorrect

The meaning of the variable `i` changes from the index of the loop counter to the sum of `a+b`

# jc\_0491: Reuse of variables within a single Stateflow scope



## Correct

tempVar is defined as local scope in both SubState\_A and SubState\_B.

## Rationale

- Readability
- Workflow
- Code Generation

## Last Changed

V2.0

# jc\_0491: Reuse of variables within a single Stateflow scope

---

**Model  
Advisor  
Check**

Not applicable

# jc\_0541: Use of tunable parameters in Stateflow

**ID: Title** jc\_0541: Use of tunable parameters in Stateflow

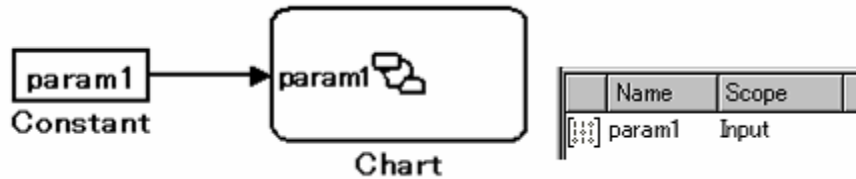
**Priority** Strongly recommended

**Scope** MAAB

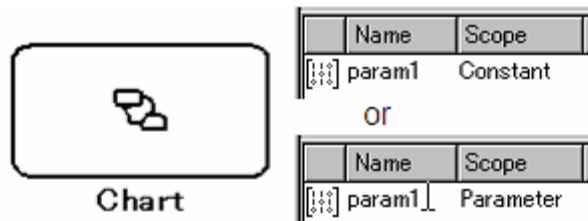
**MATLAB Versions** All

**Prerequisites** None

**Description** Tunable parameters should be included in a Chart as inputs from the Simulink model.



### Correct



### Incorrect

### Rationale

- Readability
- Workflow
- Code Generation



# jc\_0541: Use of tunable parameters in Stateflow

---

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

“Check for use of tunable parameters in Stateflow”

# db\_0127: MATLAB commands in Stateflow

---

**ID: Title** db\_0127: MATLAB commands in Stateflow

**Priority** Mandatory

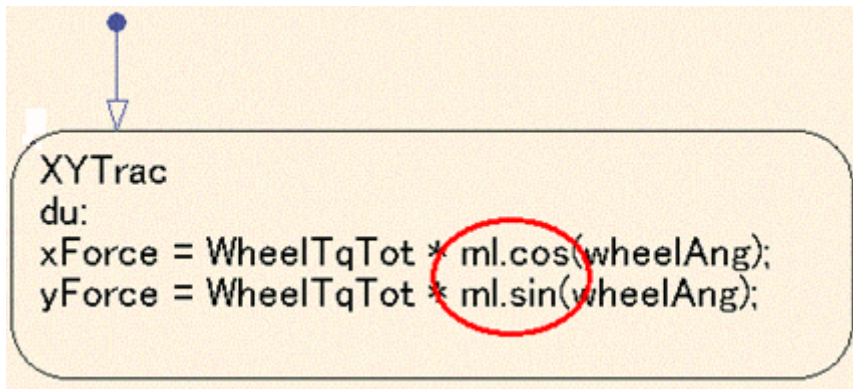
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

**Description** The following rules apply to logic in Stateflow:

- MATLAB functions are not used.
- MATLAB instructions are not used.
- MATLAB operators are not used.
- Project-specific MATLAB functions are not used.



**Incorrect**

- Rationale**
- Readability
  - Workflow

- Verification and Validation
- Code Generation

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# jm\_0011: Pointers in Stateflow

---

<b>ID: Title</b>	jm_0011: Pointers in Stateflow
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	In a Stateflow diagram, pointers to custom code variables are not allowed.
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li><li>• Code Generation</li></ul>
<b>Last Changed</b>	V1.0
<b>Model Advisor Check</b>	Not applicable

## Events

db\_0126: Scope of events

jm\_0012: Event broadcasts

## db\_0126: Scope of events

---

<b>ID: Title</b>	db_0126: Scope of events
<b>Priority</b>	Mandatory
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	<p>The following rules apply to events in Stateflow:</p> <ul style="list-style-type: none"><li>• All events of a Chart must be defined on the chart level or lower.</li><li>• There is no event on the machine level (i.e. there is no interaction with local events between different charts).</li></ul>
	<b>Specifics</b>
<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V2.0
<b>Model Advisor Check</b>	“Check whether Stateflow events are defined at the chart level or below”

**ID: Title** jm\_0012: Event broadcasts

**Priority** Strongly recommended

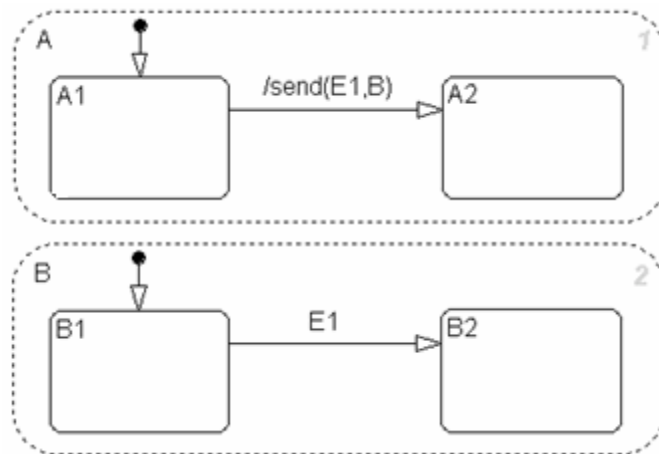
**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** db\_0126: Scope of events

**Description** The following rules apply to event broadcasts in Stateflow:

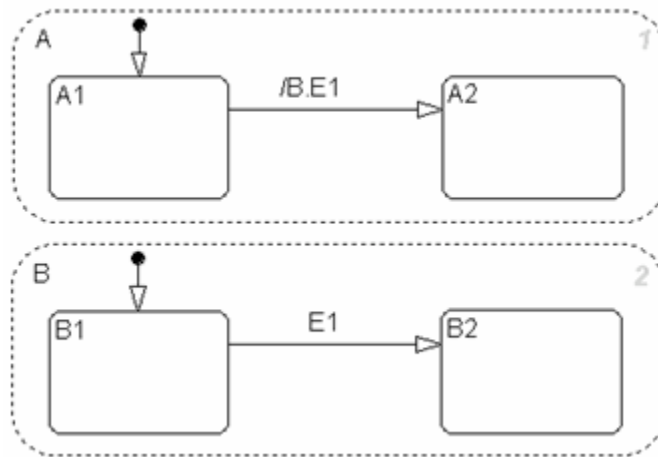
- Directed event broadcasts are the only type of event broadcasts allowed.
- The send syntax or qualified event names are used to direct the event to a particular state.
- Multiple send statements should be used to direct an event to more than one state.



**Example Using Send Syntax**

# jm\_0012: Event broadcasts

---



**Example Using Qualified Event Names**

**Rationale**

- Readability
- Workflow
- Verification and Validation
- Code Generation

**Last Changed**

V1.0

**Model Advisor Check**

Not applicable



## Statechart Patterns

db\_0150: State machine patterns  
for conditions

db\_0151: State machine patterns  
for transition actions

# db\_0150: State machine patterns for conditions

---

**ID: Title** db\_0150: State machine patterns for conditions

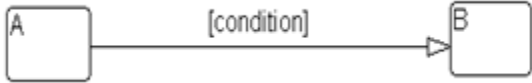
**Priority** Strongly recommended

**Scope** MAAB

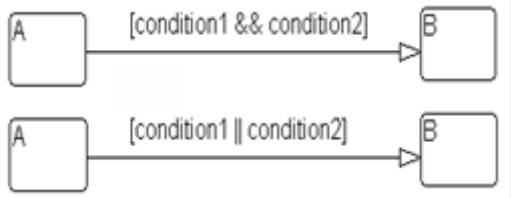
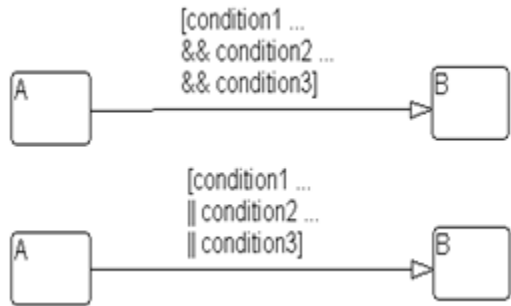
**MATLAB Versions** All

**Prerequisites** None

**Description** The following patterns are used for conditions within Stateflow state machines:

<b>Equivalent Functionality</b>	<b>State Machine Pattern</b>
One condition:  (condition)	 <p>The diagram shows a state machine pattern with two states, A and B, represented by rounded rectangles. A horizontal arrow points from state A to state B. Above the arrow is the text "[condition]". The arrow ends in a small triangle pointing towards state B.</p>

# db\_0150: State machine patterns for conditions

Equivalent Functionality	State Machine Pattern
<p>Up to three conditions, short form:</p> <p>(The use of different logical operators in this form is not allowed. Use subconditions instead.)</p> <pre>(condition1 &amp;&amp; condition2) (condition1    condition2)</pre>	
<p>Two or more conditions, multiline form:</p> <p>A subcondition is a set of logical operations, all of the same type, enclosed in parentheses.</p> <p>(The use of different operators in this form is not allowed. Use subconditions instead.)</p> <pre>(condition1 ... &amp;&amp; condition2 ... &amp;&amp; condition3) (condition1 ...    condition2 ...    condition3)</pre>	

## Rationale

- Readability
- Workflow

## db\_0150: State machine patterns for conditions

---

- Verification and Validation

**Last  
Changed**

V2.0

**Model  
Advisor  
Check**

Not applicable

# db\_0151: State machine patterns for transition actions

**ID: Title** db\_0151: State machine patterns for transition actions

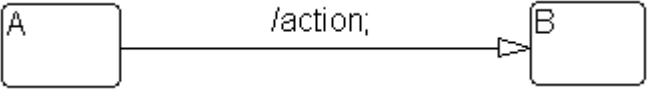
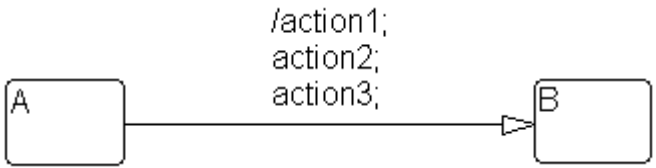
**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

**Prerequisites** None

## Description

Equivalent Functionality	State Machine Pattern
One transition action:  action;	
Two or more transition actions, multiline form: (Two or more transition actions in one line are not allowed.)  action1; action2; action3;	

For more information about Stateflow actions, see “Using Actions in Stateflow Charts” in the Stateflow documentation.

# db\_0151: State machine patterns for transition actions

---

**Rationale**

- Readability
- Workflow
- Verification and Validation

**Last  
Changed**

V1.0

**Model  
Advisor  
Check**

Not applicable

## Flowchart Patterns

db\_0134: Flowchart patterns for  
If constructs

db\_0135: Flowchart patterns for  
loop constructs

db\_0148: Flowchart patterns for  
conditions

db\_0149: Flowchart patterns for  
condition actions

db\_0159: Flowchart patterns for  
case constructs

The preceding guidelines illustrate sample patterns used in flow charts. As such, they would normally be part of a much larger Stateflow diagram.

# db\_0148: Flowchart patterns for conditions

**ID: Title** db\_0148: Flowchart patterns for conditions

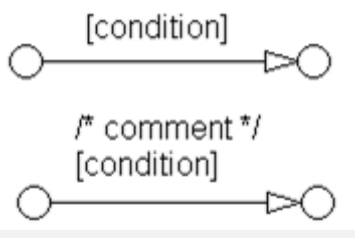
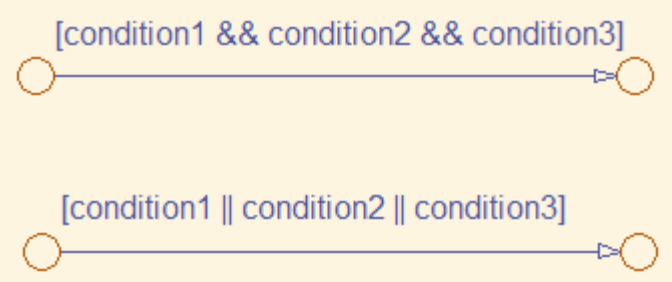
**Priority** Strongly recommended

**Scope** MAAB

**MATLAB Versions** All

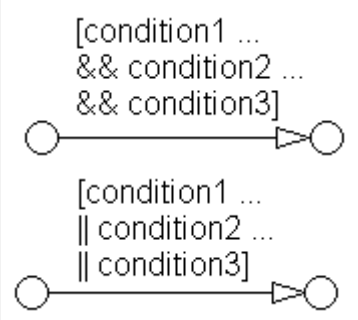
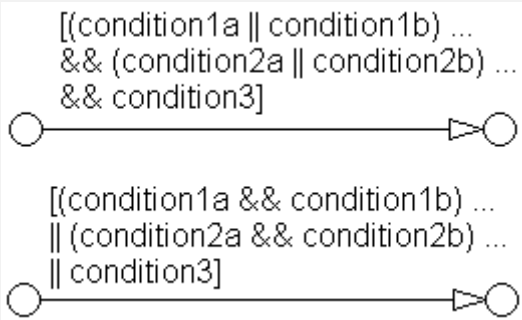
**Prerequisites** None

**Description** Use the following patterns for conditions within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
One condition:  [condition]	
Up to three conditions, short form:  (The use of different logical operators in this form is not allowed. Use subconditions instead.)  [condition1 && condition2 && condition3] [condition1    condition2    condition3]	



# db\_0148: Flowchart patterns for conditions

Equivalent Functionality	Flowchart Pattern
<p>Two or more conditions, multiline form: (The use of different logical operators in this form is not allowed. Use subconditions instead.)</p> <pre>[condition1 ... &amp;&amp; condition2 ... &amp;&amp; condition3] [condition1 ...    condition2 ...    condition3]</pre>	
<p>Conditions with subconditions: (The use of different logical operators to connect subconditions is not allowed. The use of brackets is mandatory.)</p> <pre>[(condition1a    condition1b) ... &amp;&amp; (condition2a    condition2b) ... &amp;&amp; (condition3)] [(condition1a &amp;&amp; condition1b) ...    (condition2a &amp;&amp; condition2b) ...    (condition3)]</pre>	

# db\_0148: Flowchart patterns for conditions

Equivalent Functionality	Flowchart Pattern
<p>Conditions that are visually separated: (This form may be combined with the preceding patterns.)</p> <pre>[condition1 &amp;&amp; condition2] [condition1    condition2]</pre>	

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V2.0

## Model Advisor Check

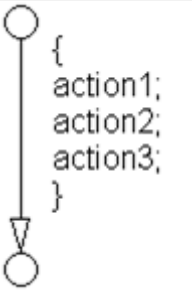
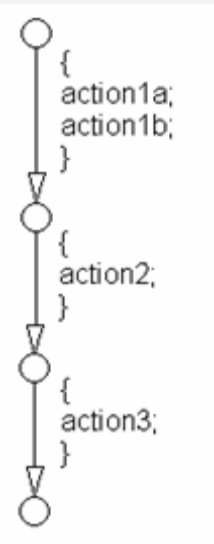
Not applicable

# db\_0149: Flowchart patterns for condition actions

<b>ID: Title</b>	db_0149: Flowchart patterns for condition actions
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	None
<b>Description</b>	Use the following patterns for condition actions within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
One condition action:  <code>action;</code>	<p>The image shows two flowchart patterns for condition actions. Each pattern consists of a top circle connected by a vertical line to a downward-pointing triangle, which is then connected to a bottom circle. The left pattern has a curly brace on the left side of the line containing the text 'action;'. The right pattern has a curly brace on the left side of the line containing the text '/* comment */' followed by 'action;' on the next line.</p>

# db\_0149: Flowchart patterns for condition actions

Equivalent Functionality	Flowchart Pattern
<p>Two or more condition actions, multiline form:</p> <p>(Two or more condition actions in one line are not allowed.)</p> <pre>action1; ... action2; ... action3; ...</pre>	 <p>The flowchart shows a vertical line starting from a circle at the top, going down to a downward-pointing triangle, and then continuing down to another circle at the bottom. To the right of the line, between the top circle and the first triangle, is a block of code: { action1; action2; action3; }. This represents a single condition block containing multiple actions.</p>
<p>Condition actions, that are visually separated:</p> <p>(This form may be combined with the preceding patterns.)</p> <pre>action1a; action1b; action2; action3;</pre>	 <p>The flowchart shows a vertical line starting from a circle at the top, going down to a downward-pointing triangle, then to a second circle, then to a second downward-pointing triangle, then to a third circle, and finally to a fourth circle at the bottom. To the right of the line, between the first and second triangles is a block of code: { action1a; action1b; }. Between the second and third triangles is a block of code: { action2; }. Between the third and fourth triangles is a block of code: { action3; }. This represents three separate condition blocks, each containing one or more actions.</p>

## Rationale

- Readability
- Workflow
- Verification and Validation

# db\_0149: Flowchart patterns for condition actions

---

**Last  
Changed**

V1.0

**Model  
Advisor  
Check**

Not applicable

# db\_0134: Flowchart patterns for If constructs

**ID: Title** db\_0134: Flowchart patterns for If constructs

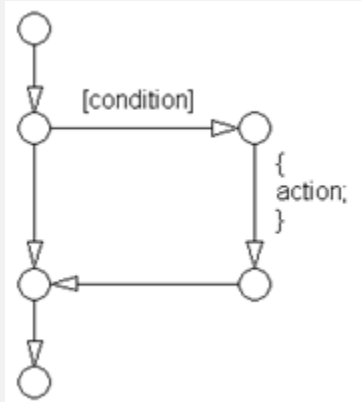
**Priority** Strongly recommended

**Scope** MAAB

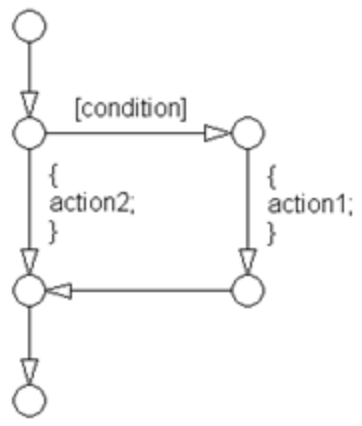
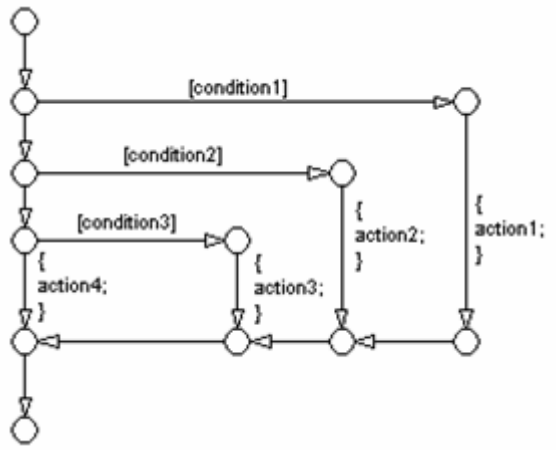
**MATLAB Versions** All

**Prerequisites** db\_0148: Flowchart patterns for conditions  
db\_0149: Flowchart patterns for condition actions

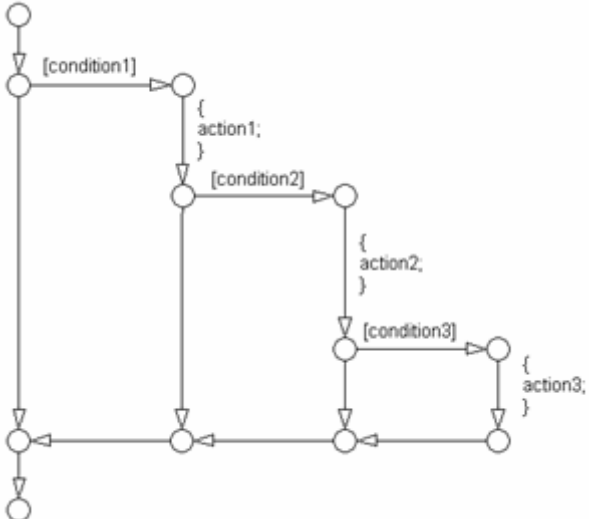
**Description** Use the following patterns for If constructs within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
<pre>if then    if (condition)   {   action;   }  endif</pre>	

# db\_0134: Flowchart patterns for If constructs

Equivalent Functionality	Flowchart Pattern
<pre> if then else     if (condition)     {     action1;     }     else     {     action2;     }         </pre>	
<pre> if then else if     if (condition1)     {     action1;     }     else if (condition2)     {     action2;     }     else if (condition3)     {     action3;     }     else     {     action4;     }         </pre>	

# db\_0134: Flowchart patterns for If constructs

Equivalent Functionality	Flowchart Pattern
<p>Cascade of if then</p> <pre>if (condition1) { action1; if (condition2) { action2; if (condition3) { action3; } } }</pre>	 <pre>graph TD     Start(( )) --&gt; C1{[condition1]}     C1 --&gt; A1["{action1;}"]     C1 --&gt; J1(( ))     A1 --&gt; C2{[condition2]}     C2 --&gt; A2["{action2;}"]     C2 --&gt; J2(( ))     A2 --&gt; C3{[condition3]}     C3 --&gt; A3["{action3;}"]     C3 --&gt; J3(( ))     J1 --&gt; J2     J2 --&gt; J3     J3 --&gt; End(( ))</pre>

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V1.0

## Model Advisor Check

Not applicable



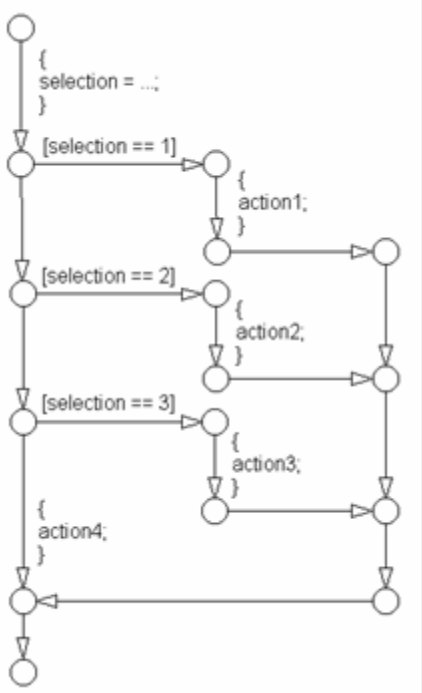
# db\_0159: Flowchart patterns for case constructs

---

<b>ID: Title</b>	db_0159: Flowchart patterns for case constructs
<b>Priority</b>	Strongly recommended
<b>Scope</b>	MAAB
<b>MATLAB Versions</b>	All
<b>Prerequisites</b>	db_0148: Flowchart patterns for conditions db_0149: Flowchart patterns for condition actions

# db\_0159: Flowchart patterns for case constructs

**Description** Use the following patterns must be used for case constructs within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
<pre>case with exclusive selection  selection = ...; switch (selection) { case 1:     action1; break; case 2:     action2; break; case 3:     action3; break; default:     action4; }</pre>	 <p>The flowchart diagram illustrates the execution of an exclusive selection case construct. It begins with a start node leading to an initialization block: { selection = ...; }. This is followed by a series of decision nodes, each with a guard condition: [selection == 1], [selection == 2], and [selection == 3]. Each guard node has a corresponding action block: { action1; }, { action2; }, and { action3; }. The flowchart uses a vertical spine of nodes. From the top, the flow goes down to the first guard node. If the guard is true, the flow branches right to the action block and then down to the next guard node. If the guard is false, the flow goes down to the next guard node. This pattern repeats for the second and third guard nodes. After the third guard node, the flow goes down to a final action block: { action4; }. From this final action block, the flow branches left to the spine and then down to the end node. This structure ensures that only one of the case actions is executed, or the default action is executed if no case guard is true.</p>

# db\_0159: Flowchart patterns for case constructs

Equivalent Functionality	Flowchart Pattern
<p>case with exclusive conditions</p> <pre>c1 = condition1; c2 = condition2; c3 = condition3; if (c1 &amp;&amp; !c2 &amp;&amp; !c3) { action1; } elseif (!c1 &amp;&amp; c2 &amp;&amp; !c3) { action2; } elseif (!c1 &amp;&amp; !c2 &amp;&amp; c3) { action3; } else { action4; }</pre>	<pre>graph TD     Start(( )) --&gt; Init["{ c1 = condition1; c2 = condition2; c3 = condition3; }"]     Init --&gt; Cond1["[c1 &amp;&amp; !c2 &amp;&amp; !c3]"]     Init --&gt; Cond2["[!c1 &amp;&amp; c2 &amp;&amp; !c3]"]     Init --&gt; Cond3["[!c1 &amp;&amp; !c2 &amp;&amp; c3]"]     Init --&gt; Cond4["{ action4; }"]     Cond1 --&gt; Act1["{ action1; }"]     Cond2 --&gt; Act2["{ action2; }"]     Cond3 --&gt; Act3["{ action3; }"]     Act1 --&gt; Merge(( ))     Act2 --&gt; Merge     Act3 --&gt; Merge     Cond4 --&gt; Merge     Merge --&gt; End(( ))</pre>

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V1.0

# db\_0159: Flowchart patterns for case constructs

---

**Model  
Advisor  
Check**

Not applicable

# db\_0135: Flowchart patterns for loop constructs

**ID: Title** db\_0135: Flowchart patterns for loop constructs

**Priority** Recommended

**Scope** MAAB

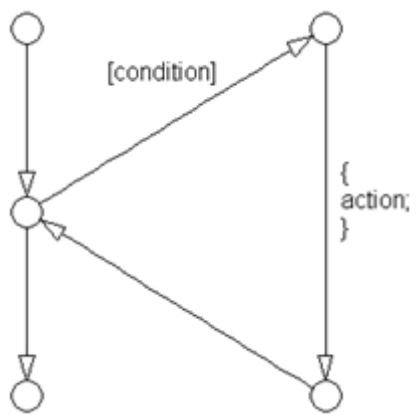
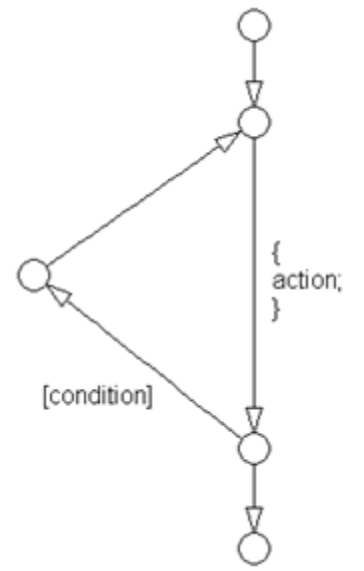
**MATLAB Versions** All

**Prerequisites** db\_0148: Flowchart patterns for conditions  
db\_0149: Flowchart patterns for condition actions

**Description** Use the following patterns to create Loops within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
<pre>for loop  for (index=0; index&lt;number_of_loops; index++) { action; }</pre>	

# db\_0135: Flowchart patterns for loop constructs

Equivalent Functionality	Flowchart Pattern
<pre>while loop  while (condition) { action; }</pre>	 <p>The flowchart for a while loop starts with an entry node at the top. A vertical arrow points down to a decision node. From the decision node, an arrow labeled "[condition]" points up and right to an action node. From the action node, a vertical arrow points down to another decision node. From this second decision node, an arrow labeled "{ action; }" points up and left back to the first decision node, forming a loop. A final vertical arrow points down from the second decision node to an exit node at the bottom.</p>
<pre>do while loop  do { action; } while (condition);</pre>	 <p>The flowchart for a do while loop starts with an entry node at the top. A vertical arrow points down to an action node. From the action node, a vertical arrow points down to a decision node. From the decision node, an arrow labeled "[condition]" points up and left to another action node. From this second action node, a vertical arrow points down back to the first decision node, forming a loop. A final vertical arrow points down from the first decision node to an exit node at the bottom.</p>

# db\_0135: Flowchart patterns for loop constructs

---

<b>Rationale</b>	<ul style="list-style-type: none"><li>• Readability</li><li>• Workflow</li><li>• Verification and Validation</li></ul>
<b>Last Changed</b>	V1.0
<b>Model Advisor Check</b>	Not applicable

## **db\_0135: Flowchart patterns for loop constructs**

---



# Recommendations for Automation Tools

---

These recommendations are for companies who develop tools that automate checking of the style guidelines. The MathWorks Automotive Advisory Board (MAAB) developed these recommendations for tool vendors who create tools developed with MathWorks tools that check models against these guidelines. To provide maximum information to potential users of the tools, the MAAB strongly recommends that tool vendors provide a compliance matrix that is easily accessible while the tool is running. This information should be available without a need to purchase the tool.

The compliance matrix should include the following information:

- Version of the guidelines that are checked – shall include the complete title, as found on the title page of this document.

Include the MAAB Style Guidelines Title and Version document number.

- Table consisting of the following information for each guideline:
  - Guideline ID
  - Guideline title
  - Level of compliance
  - Detail

The guideline ID and title shall be exactly as included in this document. The level of compliance shall be one of the following:

Correction	The tool checks and automatically or semiautomatically corrects the noncompliance.
Check	The tool checks and flags noncompliance. It is the developer's responsibility to make the correction.
Partial	The tool checks part of the guideline. The detail section should clearly identify what is and what is not checked.
None	The tool does not check the guideline. The MAAB recommends that the vendor provide a recommendation of how to manually check guidelines that the tool does not check.

# Guideline Writing

---

Guidelines with the following characteristics are easier to understand and use. At a minimum, when writing a new guideline, the guideline should be

Understandable and unambiguous	A guideline's description should be precise, clearly worded, concise, and should define a characteristic of a model (or part of a model) that the checking tool can evaluate. Use the words "must," "shall," "should," and "may" carefully; they have distinct meanings that are important for model developers and model checkers (human and automated). It is helpful to the reader if the guideline author describes how the conforming state can be reached (for example, by selecting particular options or clicking a certain button). Examples, counterexamples, pictures, diagrams, and screen shots are also helpful and are encouraged.
Easy to find	Minimize the allowable exceptions to a guideline; exceptions blur a guideline and make it harder to apply. If a guideline has many allowable exceptions, you may be trying to cover too many characteristics with one guideline. (See Minimal, following, for some solutions.)
Minimal	A guideline should have a clear, stable title and be properly located among all the other guidelines. The title should describe the topic covered but not the specific evaluation criteria. This makes the title less likely to change over time and, therefore, easier to find. Specific evaluation criteria should be included in the

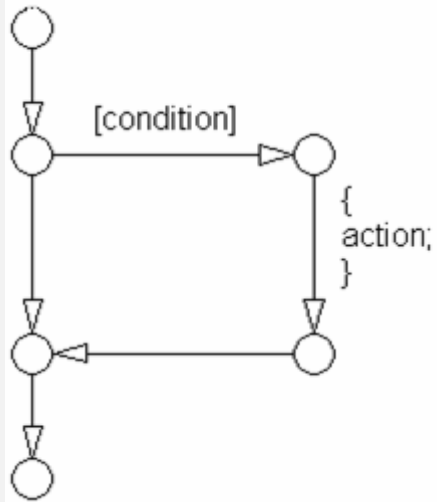
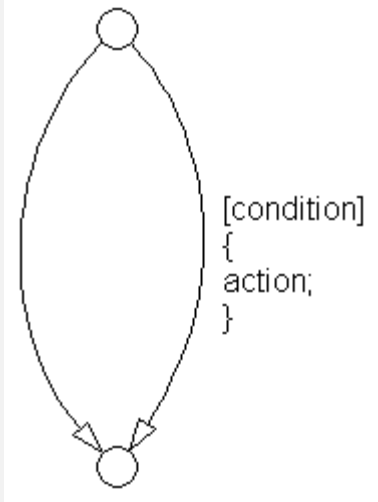
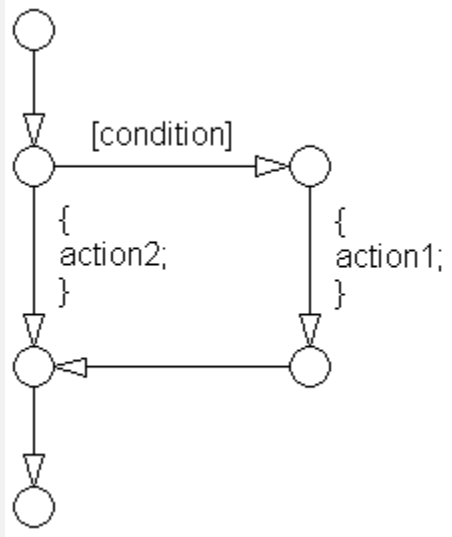
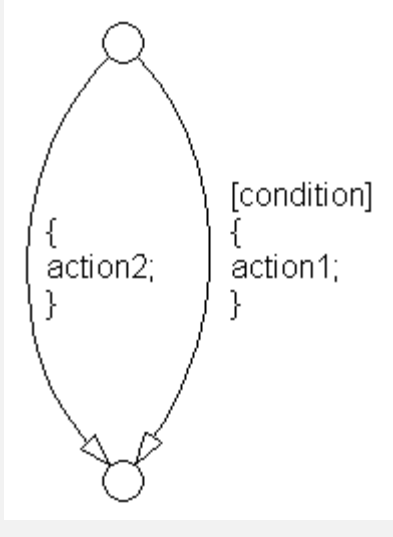
guideline description. For example, if a guideline addresses the characters allowed in names, the guideline title should be something like "Allowed characters in names," and the guideline description should indicate specifically what characters are or are not to be used. If a guideline has prerequisites, they should appear before the dependent guideline. (This may not always be possible if the prerequisite is in a different section.)

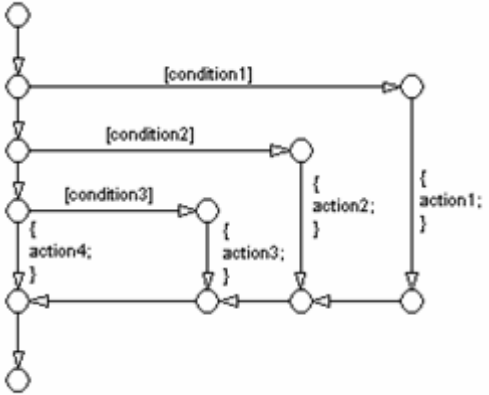
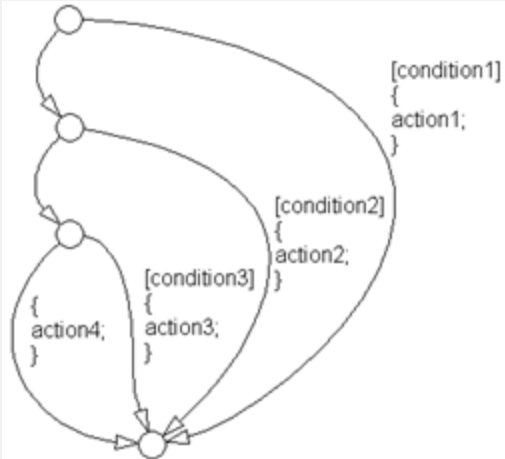
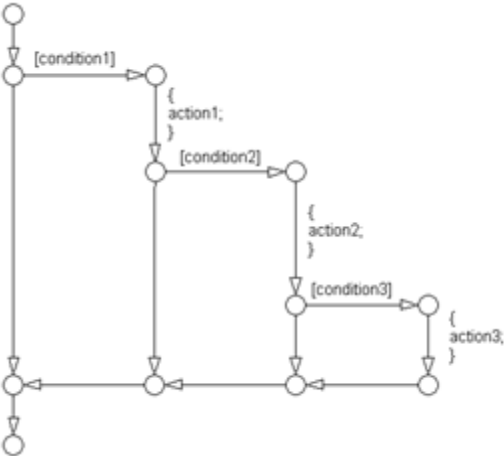
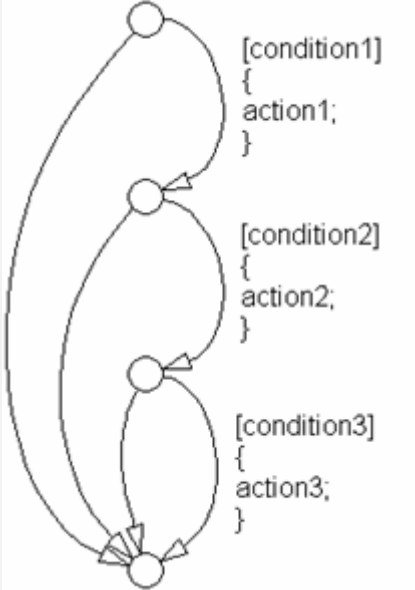
A guideline should address only one model characteristic at a time. Guidelines should be atomic. For example, instead of writing a big guideline that addresses error prevention and readability at the same time, make two guidelines, one that addresses error prevention and one that addresses readability. If appropriate, make one guideline a prerequisite of the other. Also, big guidelines are more likely than small guidelines to require compromises for wide acceptance. Big guidelines may end up being weaker, less specific, and less beneficial. Small, focused guidelines are less likely to change due to compromise and easier adoption.

# Flowchart Reference

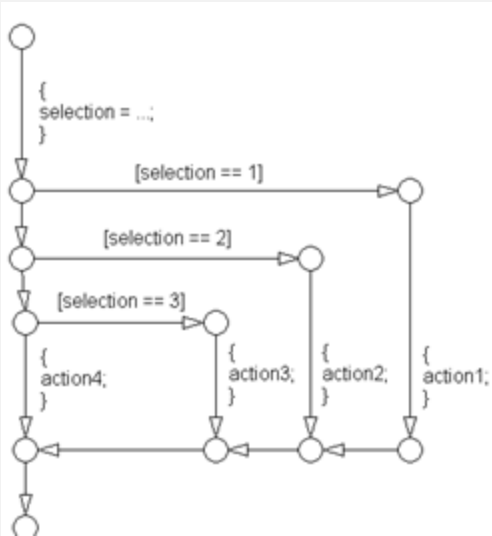
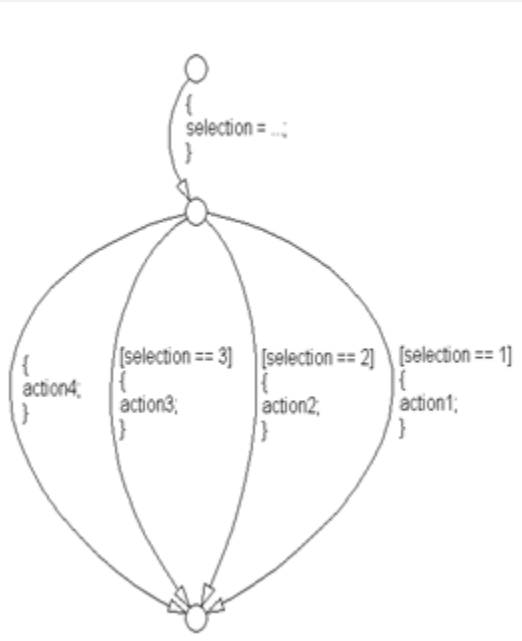
---

Use the patterns that appear in this appendix for if-then-else-if constructs within Stateflow Flowcharts.

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="136 300 244 326">if then</p> 	
<p data-bbox="136 894 319 920">if then else</p> 	

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="138 300 360 326">if then else if</p> 	
<p data-bbox="138 847 382 873">Cascade of if then</p> 	

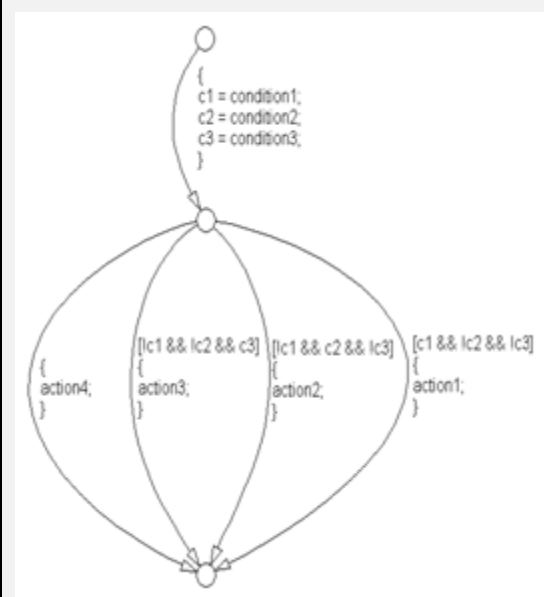
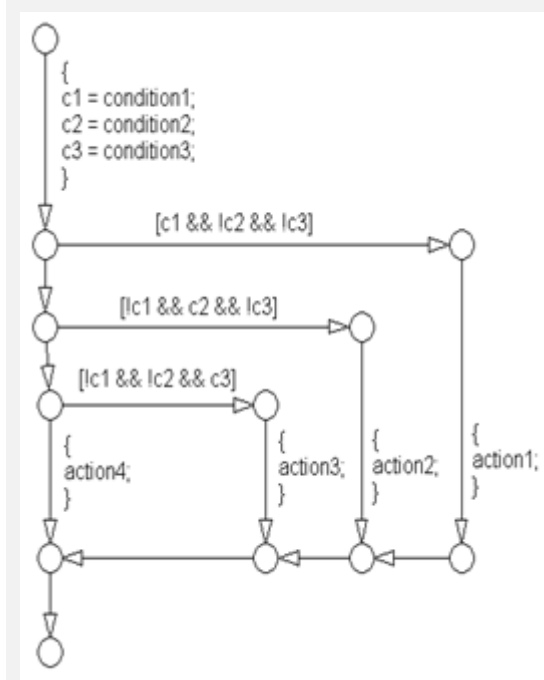
The following patterns are used for case constructs within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
case with exclusive selection	
 <p>The diagram shows a vertical flowchart with five nodes. The top node is the start node. A downward arrow leads to a node containing the code block <code>{ selection = ...; }</code>. From this node, three horizontal arrows branch out to the right, each labeled with a selection condition: <code>[selection == 1]</code>, <code>[selection == 2]</code>, and <code>[selection == 3]</code>. These arrows lead to three separate nodes, each containing a code block: <code>{ action1; }</code>, <code>{ action2; }</code>, and <code>{ action3; }</code>. From these three nodes, three horizontal arrows branch out to the left, leading to a fourth node containing the code block <code>{ action4; }</code>. Finally, a downward arrow leads from this node to the end node.</p>	 <p>The diagram shows a curved flowchart with two main nodes. The top node is the start node, with a curved arrow pointing down to a node containing the code block <code>{ selection = ...; }</code>. From this node, three curved arrows branch out downwards and to the right, each labeled with a selection condition: <code>[selection == 1]</code>, <code>[selection == 2]</code>, and <code>[selection == 3]</code>. These arrows lead to three separate nodes, each containing a code block: <code>{ action1; }</code>, <code>{ action2; }</code>, and <code>{ action3; }</code>. From these three nodes, three curved arrows branch out downwards and to the left, leading to a fourth node containing the code block <code>{ action4; }</code>. Finally, a curved arrow leads from this node back up to the start node.</p>

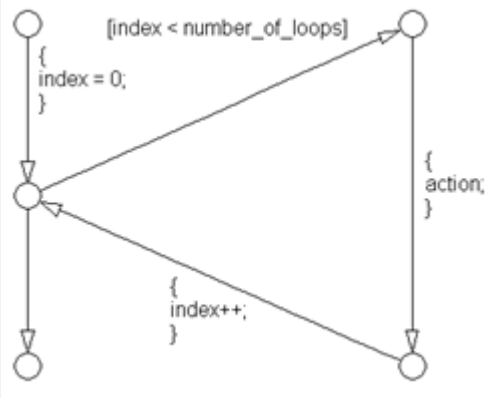
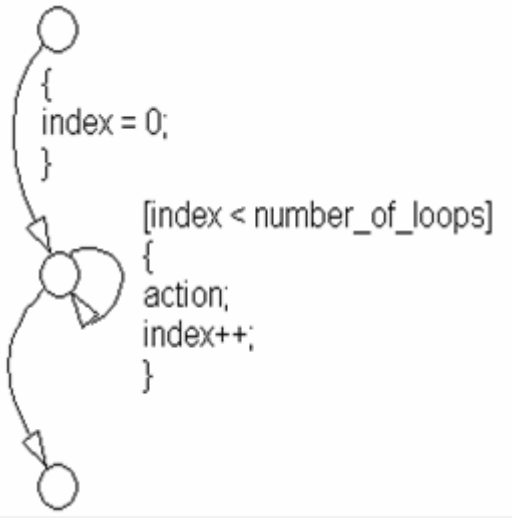


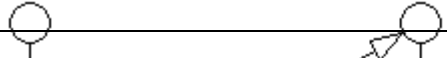

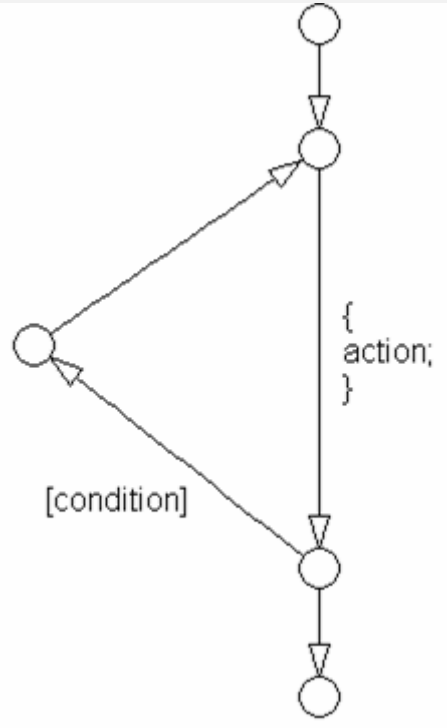
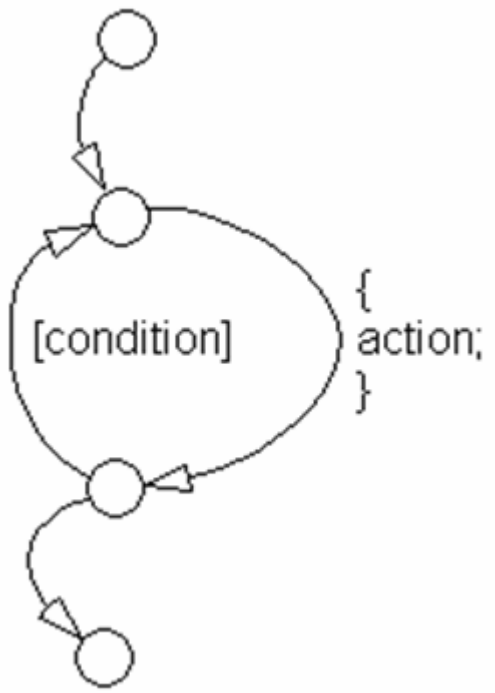
**Straight Line Flow Chart Pattern****Curved Line Flow Chart Pattern**

case with exclusive conditions

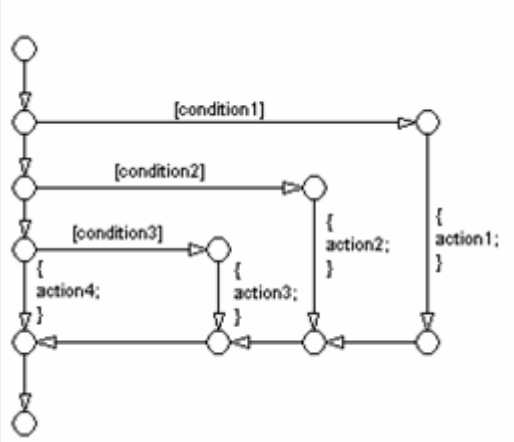
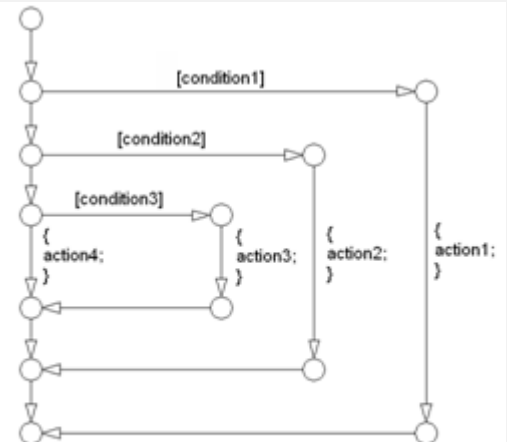


The following patterns are used for for loops within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
for loop	
 <p>The diagram shows a flowchart with four nodes. The top node is the start of the loop, with a guard condition <code>[index &lt; number_of_loops]</code>. An arrow points down to a second node containing the initialization <code>{ index = 0; }</code>. From this second node, an arrow points right to a third node containing the loop body <code>{ action; }</code>. From the third node, an arrow points down to a fourth node containing the increment <code>{ index++; }</code>. From the fourth node, an arrow points back up to the top node, completing the loop.</p>	 <p>The diagram shows a flowchart with three nodes. The top node is the start of the loop, with a guard condition <code>[index &lt; number_of_loops]</code>. An arrow points down to a second node containing the initialization <code>{ index = 0; }</code>. From this second node, an arrow points down to a third node containing the loop body <code>{ action; index++; }</code>. From the third node, a curved arrow loops back up to the top node, completing the loop.</p>

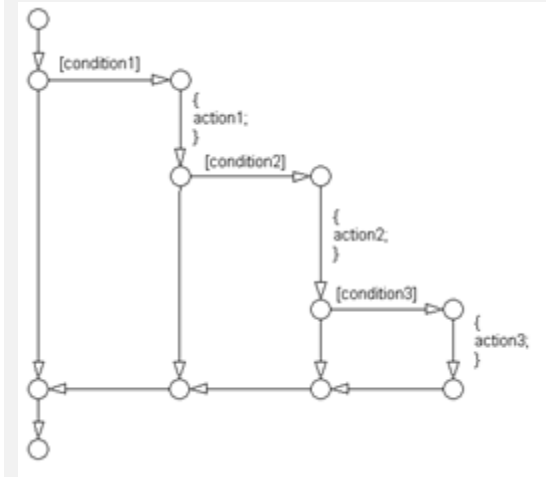
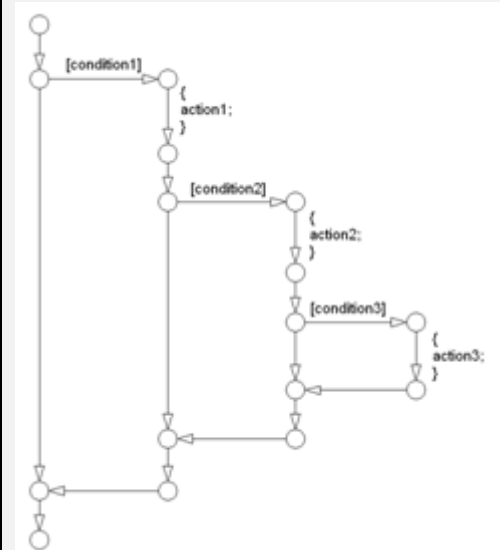
Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
while loop	
	
Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
do while loop	
	

The following patterns are alternately used for If-then-else-if constructs within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
if then else if	
 <p>The diagram illustrates the 'Straight Line Flow Chart Pattern' for an 'if then else if' construct. It features a vertical sequence of nodes. The flow starts at the top node and proceeds downwards. Three horizontal transitions branch off to the right, labeled [condition1], [condition2], and [condition3]. Each transition leads to a node containing an action block: {action1;}, {action2;}, and {action3;}. From each of these nodes, a horizontal transition leads back to the main vertical line. The final node on the vertical line contains {action4;}. The flow ends at the bottom node.</p>	 <p>The diagram illustrates the 'Alternate Straight Line Flow Chart Pattern' for an 'if then else if' construct. It features a vertical sequence of nodes. Three horizontal transitions branch off to the right, labeled [condition1], [condition2], and [condition3]. Each transition leads to a node containing an action block: {action1;}, {action2;}, and {action3;}. From each of these nodes, a horizontal transition leads back to the main vertical line. The final node on the vertical line contains {action4;}. The flow ends at the bottom node.</p>

**Straight Line Flow Chart Pattern**

Cascade of if then

**Alternate Straight Line Flow Chart Pattern**



# Background Information on Basic Blocks and Signals








---

- “Basic Blocks” on page D-2
- “Signals and Signal Labels” on page D-3

## Basic Blocks

This document uses the term *basic blocks* to refer to blocks built into the Simulink block libraries. The following table lists some examples of basic blocks.

### Basic Blocks

Block	Example
Inport	
Constant	
Gain	
Sum	
Switch	
Saturation	
Abs	



## Signals and Signal Labels

Signals may be scalars, vectors, or busses. They may carry data or control flows.

You use signal labels to make model functionality more understandable from the Simulink diagram. You can also use them to control the variable names used in simulation and code generation. Enter signal labels only once (at the point of signal origination). Often, you may want to also display the signal name elsewhere in the model. In these cases, the signal name should be inherited until the signal is functionally transformed. (Passing a signal through an integrator is functionally transforming. Passing a signal through an Inport into a nested subsystem is not.) Once a named signal is functionally transformed, associate a new name with it.

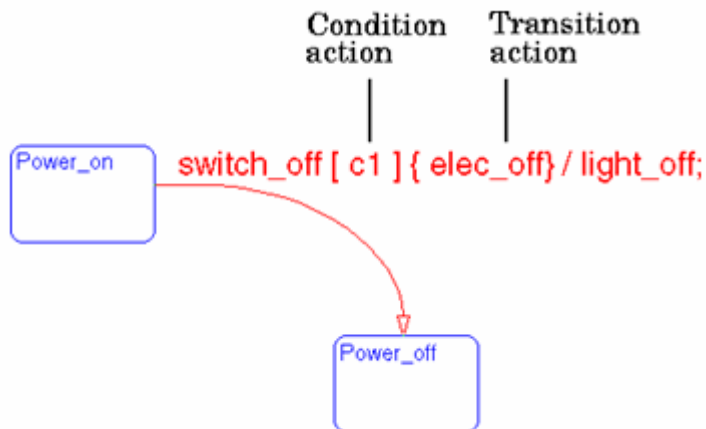
Unless explicitly stated otherwise, the guidelines in “Signals” on page 5-27 apply to all types of signals.

For more information about the representation of signals in Simulink models, see “Working with Signals” in the Simulink documentation.

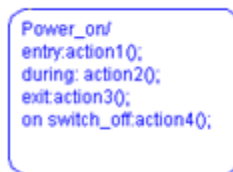


## Actions

Actions are part of Stateflow diagram execution. The action can be executed as part of a transition from one state to another, or depending on the activity status of a state. Transitions can have condition actions and transition actions. For example,



States can have entry, during, exit, and, on *event\_name* actions. For example,



If you enter the name and backslash followed directly by an action or actions (without the entry keyword), the actions are interpreted as entry actions. This shorthand is useful if you are specifying only entry actions.

The action language defines the categories of actions you can specify and their associated notations. An action can be a function call, an event to be broadcast, a variable to be assigned a value, and so on.

## Action Language

Sometimes you want actions to take place as part of Stateflow diagram execution. The action can be executed as part of a transition from one state to another, or it can depend on the activity status of a state. Transitions can have condition actions and transition actions. States can have entry, during, exit, and, on *event\_name* actions. An action can be a function call, an event to be broadcast, a variable to be assigned a value, etc.

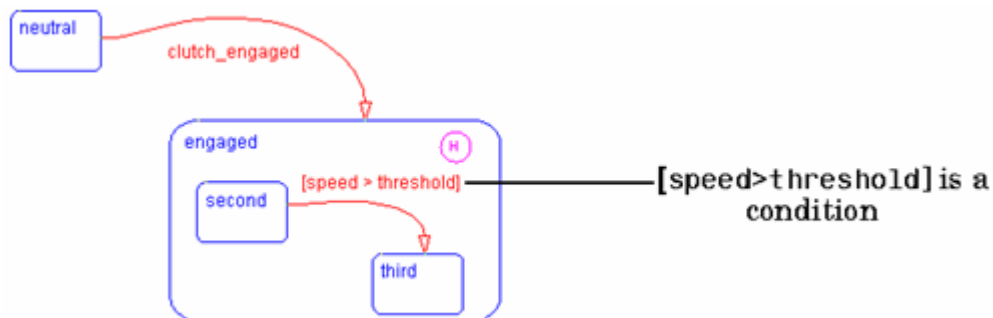
The action language defines the categories of actions you can specify and their associated notations. Violations of the action language notation are flagged as errors by the parser. This section describes the action language notation rules.

## Chart Instance

A chart instance is a link from a Stateflow model to a chart stored in a Simulink library. A chart in a library can have many chart instances. Updating the chart in the library automatically updates all the instances of that chart.

## Condition

A condition is a Boolean expression to specify that a transition occur, given that the specified expression is true. For example,

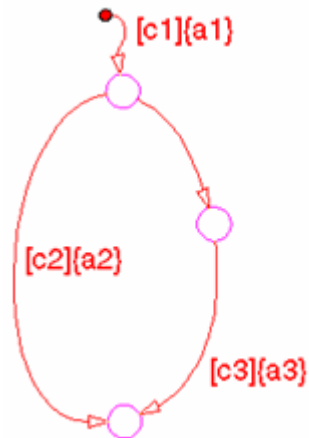


The action language defines the notation to define conditions associated with transitions.

## Connective Junction

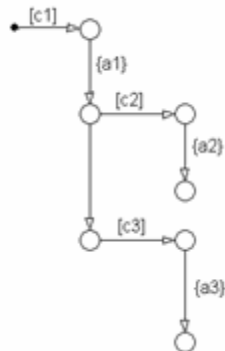
Connective junctions are decision points in the system. A connective junction is a graphical object that simplifies Stateflow diagram

representations and facilitates generation of efficient code. Connective junctions provide alternative ways to represent the system behavior you want. This example shows how connective junctions (displayed as small circles) are used to represent the flow of an if code structure.




```
if [c1]{
    a1
    if [c2]{
        a2
    }else if [c3]{
        a3
    }
}
```

Or the equivalent squared style



```
if [c1]{
    a1
    if [c2]{
        a2
    }else if [c3]{
        a3
    }
}
```

Name	Button Icon	Description
Connective junction		One use of a Connective junction is to handle situations where transitions out of one state into two or more states are taken based on the same event but guarded by different conditions.

### Data

Data objects store numerical values for reference in the Stateflow diagram.

### Defining Data

A state machine can store and retrieve data that resides internally in its own workspace. It can also access data that resides externally in the Simulink model or application that embeds the state machine. When creating a Stateflow model, you must define any internal or external data referenced by the state machine's actions.

### Data Dictionary

The data dictionary is a database where Stateflow diagram information is stored. When you create Stateflow diagram objects, the information about those objects is stored in the data dictionary, once you save the Stateflow diagram.

### Decomposition


A state has decomposition when it consists of one or more substates. A Stateflow diagram that contains at least one state also has decomposition. Representing hierarchy necessitates some rules around how states can be grouped in the hierarchy. A superstate has either parallel (AND) or exclusive (OR) decomposition. All substates at a particular level in the hierarchy must be of the same decomposition.

**Parallel (AND) State Decomposition.** Parallel (AND) state decomposition is indicated when states have dashed borders. This representation is appropriate if all states at that same level in the hierarchy are active at the same time. The activity within parallel states is essentially independent.

**Exclusive (OR) State Decomposition.** Exclusive (OR) state decomposition is represented by states with solid borders. Exclusive (OR) decomposition is used to describe system modes that are mutually exclusive. Only one state, at the same level in the hierarchy, can be active at a time.

### Default Transition

Default transitions are primarily used to specify which exclusive (OR) state is to be entered when there is ambiguity among two or more neighboring exclusive (OR) states. For example, default transitions specify which substate of a superstate with exclusive (OR) decomposition the system enters by default in the absence of any other information. Default transitions are also used to specify that a junction should be entered by default. A default transition is represented by selecting the default transition object from the toolbar and then dropping it to attach to a destination object. The default transition object is a transition with a destination but no source object.

Name	Button Icon	Description
Default transition		Use a Default transition to indicate, when entering this level in the hierarchy, which state becomes active by default.

### Events

Events drive the Stateflow diagram execution. Define all events that affect the Stateflow diagram. The occurrence of an event causes the status of the states in the Stateflow diagram to be evaluated. The broadcast of an event can trigger a transition to occur and/or can trigger an action to be executed. Events are broadcast in a top-down manner starting from the event's parent in the hierarchy.

### Finite State Machine

A finite state machine (FSM) is a representation of an event-driven system. FSMs are also used to describe reactive systems. In an event-driven or reactive system, the system transitions from one mode or state, to another prescribed mode or state, provided that the condition defining the change is true.

**Flow Graph**

A flow graph is the set of Flowcharts that start from a transition segment that, in turn, starts from a state or a default transition segment.

**Flowchart (also known as Flow Path)**

A Flowchart is an ordered sequence of transition segments and junctions where each succeeding segment starts on the junction that terminated the previous segment.

**Flow Subgraph**


A flow subgraph is the set of Flowcharts that start on the same transition segment.

**Hierarchy**

Using hierarchy you can organize complex systems by placing states within other higher-level states. A hierarchical design usually reduces the number of transitions and produces neat, more manageable diagrams.

**History Junction**

A History Junction specifies the destination substate of a transition based on historical information. If a superstate has a History Junction, the transition to the destination substate is defined to be the substate that was most recently visited. The History Junction applies to the level of the hierarchy in which it appears.

Name	Button Icon	Description
History Junction		Use a History Junction to indicate, when entering this level in the hierarchy, that the last state that was active becomes the next state to be active.

**Inner Transitions**

An inner transition is a transition that does not exit the source state. Inner transitions are most powerful when defined for superstates with XOR decomposition. Use of inner transitions can greatly simplify a Stateflow diagram.



**Library Link**

A library link is a link to a chart that is stored in a library model in a Simulink block library.

**Library Model**

A Stateflow library model is a Stateflow model that is stored in a Simulink library. You can include charts from a library in your model by copying them. When you copy a chart from a library into your model, Stateflow does not physically include the chart in your model. Instead, it creates a link to the library chart. You can create multiple links to a single chart. Each link is called a chart instance. When you include a chart from a library in your model, you also include its state machine. A Stateflow model that includes links to library charts has multiple state machines. When Stateflow simulates a model that includes charts from a library model, it includes all charts from the library model even if there are links to only some of its models. However, when Stateflow generates a stand-alone or Real-Time Workshop® target, it includes only those charts for which there are links. A model that includes links to a library model can be simulated only if all charts in the library model are free of parse and compile errors.

**Machine**

A machine is the collection of all Stateflow blocks defined by a Simulink model exclusive of chart instances (library links). If a model includes any library links, it also includes the state machines defined by the models from which the links originate.

**Nonvirtual Block**

Blocks that perform a calculation, such as a Gain block.

**Notation**

A notation defines a set of objects and the rules that govern the relationships between those objects. Stateflow notation provides a common language to communicate the design information conveyed by a Stateflow diagram. Stateflow notation consists of:

- A set of graphical objects
- A set of nongraphical text-based objects
- Defined relationships between those objects

**Parallelism**

A system with parallelism can have two or more states that can be active at the same time. The activity of parallel states is independent. Parallelism is represented with a parallel (AND) state decomposition.

**Real-Time System**

A system that uses actual hardware to implement algorithms, for example, digital signal processing or control applications.

**Real-Time Workshop**

Real-Time Workshop software includes an automatic C language code generator for Simulink. It produces C code directly from Simulink block diagram models and automatically builds programs that can be run in real-time in a variety of environments.

**Real-Time Workshop Target**

An executable built from code generated by the Real-Time Workshop product.

**S-Function**

A customized Simulink block written in C or M-code. S-functions written in C can be inlined in the Real-Time Workshop software. When using Simulink together with Stateflow for simulation, Stateflow generates an S-function (MEX-file) for each Stateflow machine to support model simulation. This generated code is a simulation target and is called the S-Fun target within Stateflow.

**Signal propagation**

Process used by Simulink to determine attributes of signals and blocks, such as data types, labels, sample time, dimensionality, and so on, that are determined by connectivity.

**Signal source**

The signal source is the block of origin for a signal. The signal source may or may not be the true source.

**Simulink**

Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can

also be multirate, that is, have different parts that are sampled or updated at different rates.


Simulink allows you to represent systems as block diagrams that you build using your mouse to connect blocks and your keyboard to edit block parameters. Stateflow is part of this environment. The Stateflow block is a masked Simulink model. Stateflow builds an S-function that corresponds to each Stateflow machine. This S-function is the agent Simulink interacts with for simulation and analysis.

The control behavior that Stateflow models complements the algorithmic behavior modeled in Simulink block diagrams. By incorporating Stateflow diagrams into Simulink models, you can add event-driven behavior to Simulink simulations. You create models that represent both data and control flow by combining Stateflow blocks with the standard Simulink blockset. These combined models are simulated using Simulink.

### State

A state describes a mode of a reactive system. A reactive system has many possible states. States in a Stateflow diagram represent these modes. The activity or inactivity of the states dynamically changes based on events and conditions.

Every state has hierarchy. In a Stateflow diagram consisting of a single state, that state's parent is the Stateflow diagram itself. A state also has history that applies to its level of hierarchy in the Stateflow diagram. States can have actions that are executed in a sequence based upon action type. The action types are: entry, during, exit, or on event\_ *name* actions.

Name	Button Icon	Description
State		Use a state to depict a mode of the system.

### Stateflow Block

The Stateflow block is a masked Simulink model and is equivalent to an empty, untitled Stateflow diagram. Use the Stateflow block to include a Stateflow diagram in a Simulink model.

The control behavior that Stateflow models complements the algorithmic behavior modeled in Simulink block diagrams. By incorporating Stateflow blocks into Simulink models, you can add complex event-driven behavior to Simulink simulations. You create models that represent both data and control flow by combining Stateflow blocks with the standard Simulink and toolbox block libraries. These combined models are simulated using Simulink.

**Stateflow Debugger**

Use the Stateflow Debugger to debug and animate your Stateflow diagrams. Each state in the Stateflow diagram simulation is evaluated for overall code coverage. This coverage analysis is done automatically when the target is compiled and built with the debug options. The Debugger can also be used to perform dynamic checking. The Debugger operates on the Stateflow machine.

**Stateflow Diagram**

Using Stateflow, you create Stateflow diagrams. A Stateflow diagram is also a graphical representation of a finite state machine where states and transitions form the basic building blocks of the system.

**Stateflow Explorer**

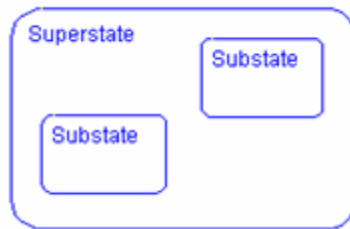
Use the Stateflow Explorer to add, remove, and modify data, event, and target objects.

**Stateflow Finder**

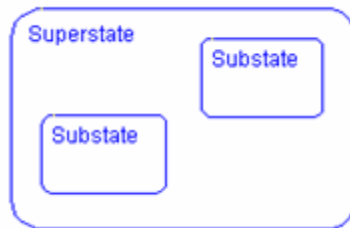
Use the Finder to display a list of objects based on search criteria that you specify. You can directly access the properties dialog box of any object in the search output display by clicking on that object.

**Substate**

A state is a substate if it is contained by a superstate.

**Superstate**

A state is a superstate if it contains other states, called substates.

**Target**

An executable program built from code generated by Stateflow or Real-Time Workshop software.

**Top-down Processing**

Top-down processing refers to the way in which Stateflow processes states. In particular, Stateflow processes superstates before states. Stateflow processes a state only if its superstate is activated first.

**Transition**

A transition describes the circumstances under which the system moves from one state to another. Either end of a transition can be attached to a source and a destination object. The source is where the transition begins and the destination is where the transition ends. It is often the occurrence of some event that causes a transition to take place.

**Transition Path**

A transition path is a Flowchart that starts and ends on a state.

**Transition Segment**

A transition segment is a single directed edge on a Stateflow diagram. Transition segments are sometimes loosely referred to as transitions.

**Tunable parameters**

A tunable parameter is a parameter that can be adjusted in the model and in generated code.

**True Source**

The true source is the block which creates a signal. The true source is different from the signal source because the signal source may be a simple routing block such as a Demux block.

**Virtual Block**

When creating models, be aware that Simulink blocks fall into two basic categories: nonvirtual and virtual blocks. Nonvirtual blocks play an active role in the simulation of a system. If you add or remove a nonvirtual block, you change the model's behavior. Virtual blocks, by contrast, play no active role in the simulation. They help to organize a model graphically. Some Simulink blocks can be virtual in some circumstances and nonvirtual in others. Such blocks are called conditionally virtual blocks. The following table lists Simulinks virtual and conditionally virtual blocks.

<b>Block Name</b>	<b>Condition Under Which Block Is Virtual</b>
Bus Selector	Virtual if input bus is virtual
Demux	Always virtual
Enable	Virtual unless connected directly to an Output block
From	Always virtual
Goto	Always virtual
Goto Tag Visibility	Always virtual
Ground	Always virtual
Inport	Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.

<b>Block Name</b>	<b>Condition Under Which Block Is Virtual</b>
Mux	Always virtual
Outport	Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.
Selector	Virtual except in matrix mode
Signal Specification	Always virtual
Subsystem	Virtual unless the block is conditionally executed and/or the block's Treat as Atomic Unit option is selected.
Terminator	Always virtual
Trigger	Virtual if the Outport port is not present.

### **Virtual Scrollbar**

Using a virtual scrollbar, you can set a value by scrolling through a list of choices. When you move the mouse over a menu item with a virtual scrollbar, the cursor changes to a line with a double arrowhead. Virtual scrollbars are either vertical or horizontal. The direction is indicated by the positioning of the arrowheads. Drag the mouse either horizontally or vertically to change the value.