

**MathWorks<sup>®</sup> Automotive Advisory Board**  
Control Algorithm Modeling Guidelines Using  
MATLAB<sup>®</sup>, Simulink<sup>®</sup>, and Stateflow<sup>®</sup>



**MATLAB<sup>®</sup>&SIMULINK<sup>®</sup>**

R2015b



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*MathWorks® Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow®*

© COPYRIGHT 2007–2015 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

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	New for Version 2.0 (Release 2009a)
September 2009	Online only	Revised for Version 2.1 (Release 2009b)
March 2010	Online only	Rereleased for Version 2.1 (Release 2010a)
September 2010	Online only	Rereleased for Version 2.1 (Release 2010b)
April 2011	Online only	Rereleased for Version 2.1 (Release 2011a)
September 2011	Online only	Rereleased for Version 2.1 (Release 2011b)
March 2012	Online only	Rereleased for Version 2.2 (Release 2012a)
September 2012	Online only	Rereleased for Version 2.2 (Release 2012b)
March 2013	Online only	Revised for Version 3.0 (Release 2013a)
September 2013	Online only	Rereleased for Version 3.0 (Release 2013b)
March 2014	Online only	Revised for Version 3.1 (Release 2014a)
October 2014	Online only	Rereleased for Version 3.1 (Release 2014b)
March 2015	Online only	Rereleased for Version 3.1 (Release 2015a)
September 2015	Online only	Rereleased for Version 3.1 (Release 2015b)



## Introduction

**1**

<b>Presentation of Guidelines Hosted by MathWorks</b> .....	<b>1-2</b>
<b>Motivation</b> .....	<b>1-3</b>
<b>Notes on Version 3.0</b> .....	<b>1-4</b>
<b>Guideline Template</b> .....	<b>1-5</b>
Guideline ID .....	<b>1-6</b>
Guideline Title .....	<b>1-6</b>
Priority .....	<b>1-6</b>
Scope .....	<b>1-7</b>
MATLAB Versions .....	<b>1-8</b>
Prerequisites .....	<b>1-8</b>
Description .....	<b>1-9</b>
Rationale .....	<b>1-9</b>
Last Change .....	<b>1-10</b>
Model Advisor Check .....	<b>1-10</b>
<b>Document Usage</b> .....	<b>1-11</b>
<b>Model Advisor Checks for MAAB Guidelines</b> .....	<b>1-12</b>

## Software Environment

**2**

<b>General Guidelines</b> .....	<b>2-2</b>
---------------------------------	------------

## **Naming Conventions**

### **3**

<b>General Guidelines</b> .....	<b>3-2</b>
<b>Model Content</b> .....	<b>3-10</b>

## **Model Architecture**

### **4**

<b>Simulink and Stateflow Partitioning</b> .....	<b>4-2</b>
<b>Subsystem Hierarchies</b> .....	<b>4-14</b>
<b>J-MAAB Model Architecture Decomposition</b> .....	<b>4-31</b>

## **Model Configuration Options**

### **5**

<b>Model Configuration Options</b> .....	<b>5-2</b>
--	------------

## **Simulink**

### **6**

<b>Diagram Appearance</b> .....	<b>6-2</b>
<b>Signals</b> .....	<b>6-42</b>
<b>Block Usage</b> .....	<b>6-53</b>
<b>Block Parameters</b> .....	<b>6-80</b>
<b>Simulink Patterns</b> .....	<b>6-88</b>

**7**

<b>Chart Appearance</b> .....	<b>7-2</b>
<b>Stateflow Data and Operations</b> .....	<b>7-28</b>
<b>Events</b> .....	<b>7-57</b>
<b>State Chart Patterns</b> .....	<b>7-64</b>
<b>Flow Chart Patterns</b> .....	<b>7-72</b>
<b>State Chart Architecture</b> .....	<b>7-91</b>

**Enumerated Data**

**8**

<b>General Guidelines</b> .....	<b>8-2</b>
---------------------------------	------------

**MATLAB Functions**

**9**

<b>MATLAB Function Appearance</b> .....	<b>9-2</b>
<b>MATLAB Function Data and Operations</b> .....	<b>9-9</b>
<b>MATLAB Function Patterns</b> .....	<b>9-15</b>
<b>MATLAB Function Usage</b> .....	<b>9-19</b>

**Recommendations for Automation Tools**

**A**

**Guideline Writing**

**B**

**Flow Chart Reference**

**C**

**Background Information on Basic Blocks and Signals**

**D**

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

**MAAB Glossary**



# Introduction

---

- “Presentation of Guidelines Hosted by MathWorks” on page 1-2
- “Motivation” on page 1-3
- “Notes on Version 3.0” on page 1-4
- “Guideline Template” on page 1-5
- “Document Usage” on page 1-11
- “Model Advisor Checks for MAAB Guidelines” on page 1-12

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

This presentation of the MathWorks® Automotive Advisory Board (MAAB) guidelines, Version 3.0, 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 one key prerequisite to achieving:

- System integration without problems
- Well-defined interfaces
- Uniform appearance of models, code, and documentation
- Reusable models
- 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

## Notes on Version 3.0

The current version of this document, 3.0, supports MATLAB® releases R2007b through R2011b. Version 3.0 references rules from the NASA Orion style guidelines (NASA - Orion GN&C: MATLAB and Simulink Standards). Rules that are referenced from the NASA Orion guideline are noted with a “See also” field that provides the original rule number.

## Guideline Template

In this section...
“Guideline ID” on page 1-6
“Guideline Title” on page 1-6
“Priority” on page 1-6
“Scope” on page 1-7
“MATLAB Versions” on page 1-8
“Prerequisites” on page 1-8
“Description” on page 1-9
“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.

Mandatory	Strongly Recommended	Recommended
<b>Definition</b>		
<p>Guidelines that all companies agree to that are absolutely essential</p> <p>Guidelines that all companies conform to 100%</p>	<p>Guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming to the guideline 100%</p> <p>Models should conform to these guidelines to the greatest extent possible; however, 100% compliance is not required</p>	<p>Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model</p> <p>Guidelines where conformance is preferred, but not required</p>
<b>Consequences:</b> If the guideline is violated,		
<p>Essential items are missing</p> <p>The model might not work properly</p>	<p>The quality and appearance deteriorates</p> <p>There may be an adverse effect on maintainability, portability, and reusability</p>	<p>The appearance does not conform with other projects</p>
<b>Waiver Policy:</b> If the guideline is intentionally ignored,		
<p>The reasons must be documented</p>		

## Scope

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

Scope	Description
<p>MAAB (MathWorks Automotive Advisory Board)</p>	<p>A group of automotive manufacturers and suppliers that work closely together with</p>

Scope	Description
	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 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 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>

<b>Rationale</b>	<b>Description</b>
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 Simulink Model Advisor MAAB checks, which correspond to a subset of MAAB guidelines, that you can select and run with the Simulink Model Advisor. In this presentation of the MAAB guidelines, MathWorks includes a Model Advisor check field in guideline descriptions, which contains the title of and a link to the corresponding Model Advisor check, if a check exists. Although this information is included, note that the MAAB working group takes a neutral stance on recommendations for style guide checkers.

For a list of available Model Advisor checks for the MAAB guidelines, see “Model Advisor Checks for MAAB Guidelines” on page 1-12. For information on using the Model Advisor, see “Run Model Checks” in the Simulink documentation.

## Document Usage

- *Name Conventions* and *Model Architecture* provide basic guidelines that apply to all types of models.
- *Simulink* and *Stateflow*<sup>®</sup> provide specific rules for those environments.
- Some guidelines are dependent on other guidelines and are explicitly listed throughout the document.
- If users do not view the content of masked subsystems with a model, the guidelines for readability are not applicable.

For information on automated checking of the guidelines, see Appendix A.

## Model Advisor Checks for MAAB Guidelines

Simulink Verification and Validation provides Model Advisor MAAB checks which correspond to a subset of MAAB guidelines. You can run the checks using the Model Advisor.

The MAAB guidelines and corresponding Model Advisor checks are summarized in the following table. Not all guidelines have Model Advisor checks. For some of the guidelines without Model Advisor checks, it is not possible to automate checking of the guideline. Guidelines without a corresponding check are noted as not applicable. For information on using the Model Advisor, see “Run Model Checks” in the Simulink documentation.

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0026: Consistent software environment	Not applicable	
na_0027: Use of only standard library blocks	Not applicable	
ar_0001: Filenames	<b>Naming Conventions</b>	“Check file names”
ar_0002: Directory names	<b>Naming Conventions</b>	“Check folder names”
na_0035: Adoption of naming conventions	Not applicable	
jc_0201: Usable characters for Subsystem names	<b>Naming Conventions</b>	“Check subsystem names”
jc_0211: Usable characters for Inport blocks and Outport blocks	<b>Naming Conventions</b>	“Check port block names”
jc_0221: Usable characters for signal line names	<b>Naming Conventions</b>	“Check character usage in signal labels”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0030: Usable characters for Simulink Bus names	Not applicable	
jc_0231: Usable characters for block names	<b>Naming Conventions</b>	“Check character usage in block names”
na_0014: Use of local language in Simulink and Stateflow	Not applicable	
na_0006: Guidelines for mixed use of Simulink and Stateflow	Not applicable	
na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines	Not applicable	
db_0143: Similar block types on the model levels	<b>Model Architecture</b>	“Check for mixing basic blocks and subsystems”
db_0144: Use of Subsystems	Not applicable	
db_0040: Model hierarchy	Not applicable	
na_0037: Use of single variable variant conditionals	Not applicable	
na_0020: Number of inputs to variant subsystems	Not applicable	
na_0036: Default variant	Not applicable	

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
jc_0301: Controller model	Not applicable	
jc_0311: Top layer/root level	Not applicable	
jc_0321: Trigger layer	Not applicable	
jc_0331: Structure layer	Not applicable	
jc_0341: Data flow layer	Not applicable	
jc_0011: Optimization parameters for Boolean data types	<b>Model Configuration Options</b>	“Check Implement logic signals as Boolean data (vs. double)”
jc_0021: Model diagnostic settings	<b>Model Configuration Options</b>	“Check model diagnostic parameters”
na_0004: Simulink model appearance	<b>Simulink</b>	“Check for Simulink diagrams using nonstandard display attributes”
db_0043: Simulink font and font size	<b>Simulink</b>	“Check font formatting”
db_0042: Port block in Simulink models	<b>Simulink</b>	“Check positioning and configuration of ports”
na_0005: Port block name visibility in Simulink models	<b>Simulink</b>	“Check visibility of block port names”
jc_0081: Icon display for Port block	<b>Simulink</b>	“Check display for port blocks”
jm_0002: Block resizing	Not applicable	
db_0142: Position of block names	<b>Simulink</b>	“Check whether block names appear below blocks”
jc_0061: Display of block names	<b>Simulink</b>	“Check the display attributes of block names”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
db_0146: Triggered, enabled, conditional Subsystems	<b>Simulink</b>	“Check position of Trigger and Enable blocks”
db_0140: Display of basic block parameters	<b>Simulink</b>	“Check for nondefault block attributes”
db_0032: Simulink signal appearance	Not applicable	
db_0141: Signal flow in Simulink models	Not applicable	
jc_0171: Maintaining signal flow when using Goto and From blocks	Not applicable	
na_0032: Use of merge blocks	Not applicable	
jm_0010: Port block names in Simulink models	<b>Simulink</b>	“Check for matching port and signal names”
jc_0281: Naming of Trigger Port block and Enable Port block	<b>Simulink</b>	“Check Trigger and Enable block names”
na_0008: Display of labels on signals	<b>Simulink</b>	“Check signal line labels”
na_0009: Entry versus propagation of signal labels	<b>Simulink</b>	“Check for propagated signal labels”
db_0097: Position of labels for signals and busses	Not applicable	
db_0081: Unconnected signals, block inputs and block outputs	<b>Simulink</b>	“Check for unconnected ports and signal lines”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0003: Simple logical expressions in If Condition block	Not applicable	
na_0002: Appropriate implementation of fundamental logical and numerical operations	Not applicable	
jm_0001: Prohibited Simulink standard blocks inside controllers	<b>Simulink</b>	“Check for prohibited blocks in discrete controllers”
hd_0001: Prohibited Simulink sinks	<b>Simulink</b>	“Check for prohibited sink blocks”
na_0011: Scope of Goto and From blocks	<b>Simulink</b>	“Check scope of From and Goto blocks”
jc_0141: Use of the Switch block	<b>Simulink</b>	“Check usage of Switch blocks”
jc_0121: Use of the Sum block	Not applicable	
jc_0131: Use of Relational Operator block	<b>Simulink</b>	“Check usage of Relational Operator blocks”
jc_0161: Use of Data Store Read/Write/Memory blocks	Not applicable	
db_0112: Indexing	<b>Simulink</b>	“Check for indexing in blocks”
na_0010: Grouping data flows into signals	<b>Simulink</b>	“Check usage of buses and Mux blocks”
db_0110: Tunable parameters in basic blocks	<b>Simulink</b>	“Check usage of tunable parameters in blocks”



<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0012: Use of Switch vs. If-Then-Else Action Subsystem	Not applicable	
db_0114: Simulink patterns for If-then-else-if constructs	Not applicable	
db_0115: Simulink patterns for case constructs	Not applicable	
na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches	Not applicable	
db_0116: Simulink patterns for logical constructs with logical blocks	Not applicable	
db_0117: Simulink patterns for vector signals	Not applicable	
jc_0351: Methods of initialization	Not applicable	
jc_0111: Direction of Subsystem	<b>Simulink</b>	“Check orientation of Subsystem blocks”
db_0123: Stateflow port names	<b>Stateflow</b>	“Check for mismatches between names of Stateflow ports and associated signals”
db_0129: Stateflow transition appearance	Not applicable	
db_0137: States in state machines	<b>Stateflow</b>	“Check usage of exclusive and default states in state machines”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
db_0133: Use of patterns for flow charts	Not applicable	
db_0132: Transitions in flow charts	<b>Stateflow</b>	“Check Transition orientations in flow charts”
jc_0501: Format of entries in a State block	<b>Stateflow</b>	“Check entry formatting in State blocks in Stateflow charts”
jc_0511: Setting the return value from a graphical function	<b>Stateflow</b>	“Check return value assignments of graphical functions in Stateflow charts”
jc_0531: Placement of the default transition	<b>Stateflow</b>	“Check default transition placement in Stateflow charts”
jc_0521: Use of the return value from graphical functions	<b>Stateflow</b>	“Check usage of return values from a graphical function in Stateflow charts”
na_0001: Bitwise Stateflow operators	<b>Stateflow</b>	“Check for bitwise operations in Stateflow charts”
jc_0451: Use of unary minus on unsigned integers in Stateflow	<b>Stateflow</b>	“Check for unary minus operations on unsigned integers in Stateflow charts”
na_0013: Comparison operation in Stateflow	<b>Stateflow</b>	“Check for comparison operations in Stateflow charts”
db_0122: Stateflow and Simulink interface signals and parameters	<b>Stateflow</b>	“Check for Strong Data Typing with Simulink I/O”
db_0125: Scope of internal signals and local auxiliary variables	<b>Stateflow</b>	“Check Stateflow data objects with local scope”
jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow	<b>Stateflow</b>	“Check for equality operations between floating-point expressions in Stateflow charts”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
jc_0491: Reuse of variables within a single Stateflow scope	Not applicable	
jc_0541: Use of tunable parameters in Stateflow	Not applicable	
db_0127: MATLAB commands in Stateflow	<b>Stateflow</b>	“Check for MATLAB expressions in Stateflow charts”
jm_0011: Pointers in Stateflow	<b>Stateflow</b>	“Check for pointers in Stateflow charts”
db_0126: Scope of events	Not applicable	
jm_0012: Event broadcasts	<b>Stateflow</b>	“Check for event broadcasts in Stateflow charts”
db_0150: State machine patterns for conditions	Not applicable	
db_0151: State machine patterns for transition actions	<b>Stateflow</b>	“Check transition actions in Stateflow charts”
db_0148: Flow chart patterns for conditions	Not applicable	
db_0149: Flow chart patterns for condition actions	Not applicable	
db_0134: Flow chart patterns for If constructs	Not applicable	
db_0159: Flow chart patterns for case constructs	Not applicable	
db_0135: Flow chart patterns for loop constructs	Not applicable	

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0038: Levels in Stateflow charts	Not applicable	
na_0039: Use of Simulink in Stateflow charts	Not applicable	
na_0040: Number of states per container	Not applicable	
na_0041: Selection of function type	Not applicable	
na_0042: Location of Simulink functions	Not applicable	
na_0033: Enumerated Types Usage	Not applicable	
na_0031: Definition of default enumerated value	Not applicable	
na_0018: Number of nested if/else and case statement	<b>MATLAB Functions</b>	“Check MATLAB Function metrics”
na_0019: Restricted Variable Names	Not applicable	
na_0025: MATLAB Function Header	Not applicable	
na_0034: MATLAB Function block input/output settings	<b>MATLAB Functions</b>	“Check input and output settings of MATLAB Functions”
na_0024: Global Variables	<b>MATLAB Functions</b>	“Check MATLAB code for global variables”

<b>MAAB Guideline - Version 3.0</b>	<b>By Task &gt; Modeling Standards for MAAB subfolder</b>	<b>Model Advisor check</b>
na_0022: Recommended patterns for Switch/Case statements	Not applicable	
na_0016: Source lines of MATLAB Functions	<b>MATLAB Functions</b>	"Check MATLAB Function metrics"
na_0017: Number of called function levels	Not applicable	
na_0021: Strings	Not applicable	



# Software Environment

---

### General Guidelines

- na\_0026: Consistent software environment
- na\_0027: Use of only standard library blocks



# na\_0026: Consistent software environment

## ID: Title

na\_0026: Consistent software environment

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

See description

## Prerequisites

None

## Description

During software development, it is recommended that a consistent software environment is used across the project. Software includes, but is not limited, to:

- MATLAB
- Simulink
- C Compiler (for simulation)

- C Compiler (for target hardware)

Consistent software environment implies that the same version of the software is used across the full project. The version number applies to any patches or extensions to the software used by a group.

### **Rationale**

- Readability
- Code Generation

### **See Also**

- NASA Orion style guideline jh\_0042: Required software

### **Last Changed**

V3.0

**Introduced in R2013a**

# na\_0027: Use of only standard library blocks

## ID: Title

na\_0027: Use of only standard library blocks

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

Companies should specify a subset of Simulink blocks for use when developing models. The block list can include custom block libraries developed by the company or third parties. Models should be built only from these blocks.

Non-compliant blocks can be used during development. If non-compliant blocks are used, they should be marked either with a color, icon and / or annotation. These blocks must be removed prior to use in production code generation.

### **Rationale**

- Readability
- Verification and Validation
- Code Generation
- Simulation

### **See Also**

- NASA Orion style guideline hyl\_0201: Use of standard library blocks only

### **Last Changed**

V3.0

**Introduced in R2013a**

# Naming Conventions

---

- “General Guidelines” on page 3-2
- “Model Content” on page 3-10

### General Guidelines

- ar\_0001: Filenames
- ar\_0002: Directory names
- na\_0035: Adoption of naming conventions

## ar\_0001: Filenames

### ID: Title

ar\_0001: Filenames

### Priority

Mandatory

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

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*:

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 \_

*extension*:

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

*extension*:

Should not use underscores

### Rationale

- Readability
- Workflow
- Code Generation
- Simulation



## **Last Changed**

V3.0

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Naming Conventions > Check file names**

For check details, see “Check file names”.

**Introduced in R2010a**

## ar\_0002: Directory 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
- Code Generation
- Simulation

## Last Changed

V1.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Naming Conventions > Check for invalid model folder names**

For check details, see “Check folder names”.

**Introduced in R2010a**

# na\_0035: Adoption of naming conventions

## ID: Title

na\_0035: Adoption of naming conventions

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

Adoption of a naming convention is recommended. A naming convention provides guidance for naming blocks, signals, parameters and data types. Naming conventions frequently cover issues such as:

- Compliance with the programming language and downstream tools
- Length

- Use of symbols
- Readability
  - Use of underscores
  - Use of capitalization
- Encoding information
  - Use of “meaningful” names
  - Standard abbreviations and acronyms
  - Data type
  - Engineering units
  - Data ownership
  - Memory type

## **Rationale**

- Readability
- Workflow
- Code Generation
- Simulation

## **Last Changed**

V3.0

**Introduced in R2013a**

### 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
- na\_0030: Usable characters for Simulink Bus 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

## ID: Title

jc\_0201: Usable characters for Subsystem

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

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
- Should not include carriage returns

### 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

### Last Changed

V2.2

### Model Advisor Check

**By Task > Modeling Standards for MAAB > Naming Conventions > Check whether subsystem block names include invalid characters**

For check details, see “Check subsystem names”.

**Introduced in R2010a**



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

## ID: Title

jc\_0211: Usable characters for Inport blocks and Outport 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
- Should not include carriage returns

### 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

### Last Changed

V2.2

### Model Advisor Check

**By Task > Modeling Standards for MAAB > Naming Conventions > Check whether Inport and Outport block names include invalid characters**

For check details, see “Check port block names”.

**Introduced in R2010a**

## **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
- Should not include carriage returns

## 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

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Naming Conventions > Check character usage in signal labels**

For check details, see “Check character usage in signal labels”.

**Introduced in R2010a**

# na\_0030: Usable characters for Simulink Bus names

## ID: Title

na\_0030: Usable characters for Simulink Bus names

## Priority

Strongly recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

All Simulink Bus names should conform to the following constraints:

## Form

*name*:

- Should not start with a number

- Should not have blank spaces
- Carriage returns are not 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 \_

### 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

### See Also

- NASA Orion style guideline jh\_0040: Usable characters for Simulink Bus Names

### Last Changed

V3.0

Introduced in R2013a



# 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 spaces at the beginning of a block name
- 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

### Last Changed

V2.0

### Model Advisor Check

**By Task > Modeling Standards for MAAB > Naming Conventions > Check character usage in block names**

For check details, see “Check character usage in block names”.

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Strongly recommended

## Scope

J-MAAB

## MATLAB Versions

All

## Prerequisites

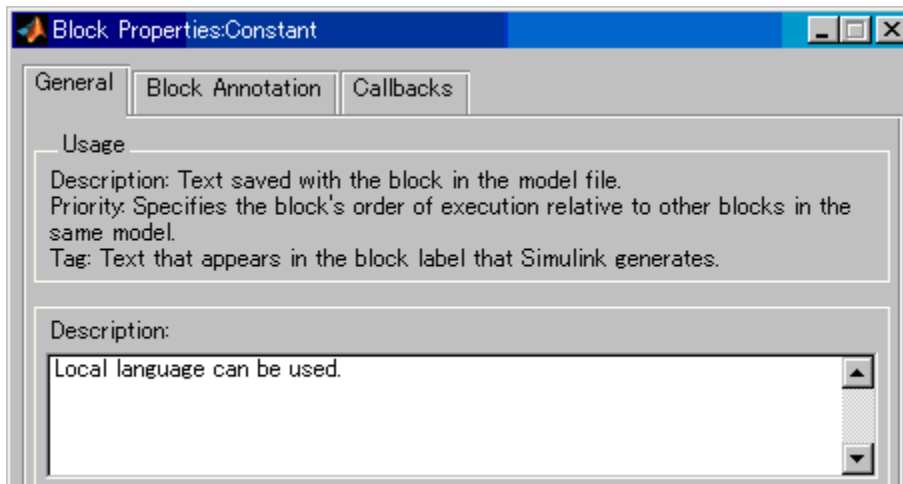
None

## Description

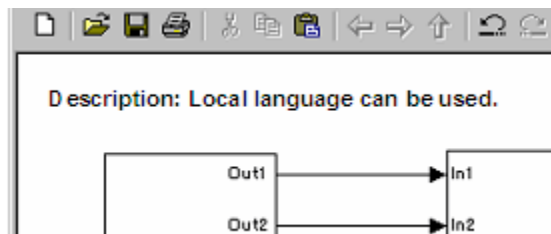
The local language should be used in descriptive fields only. Descriptive fields are text entry points that do not affect code generation or simulation. Examples of descriptive fields include the **Description** field in the Block Properties dialog box.

## Simulink Examples

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

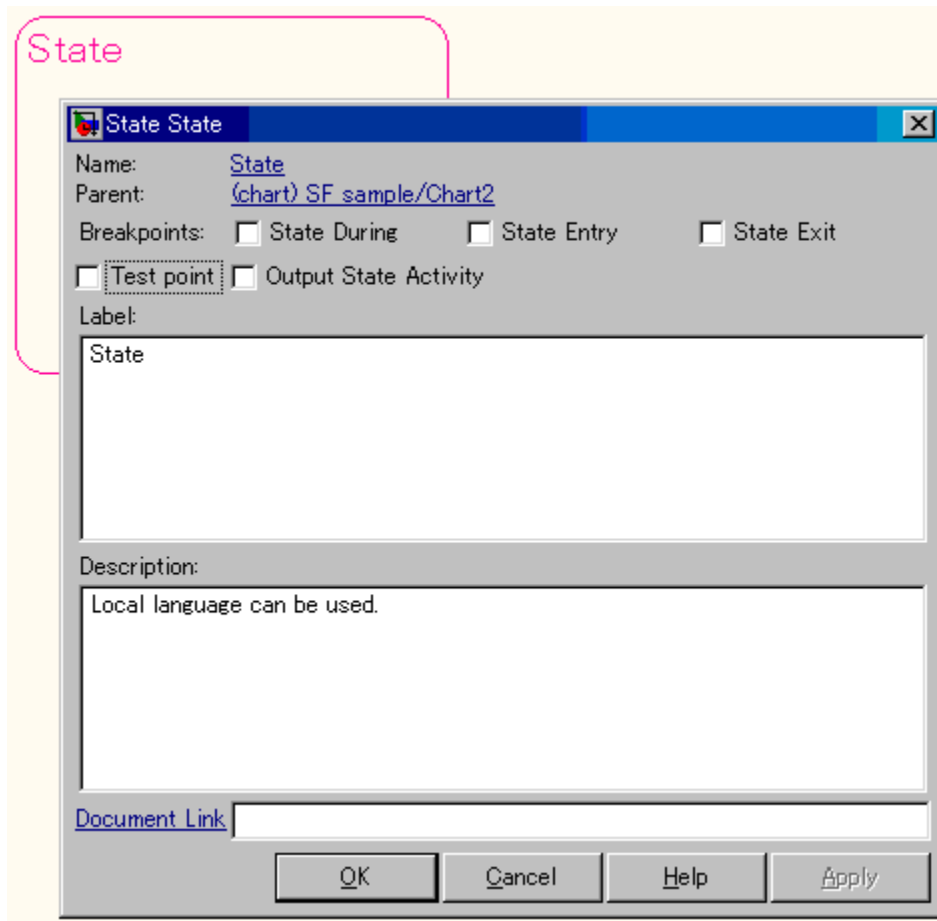


- Text annotation entered directly in the model

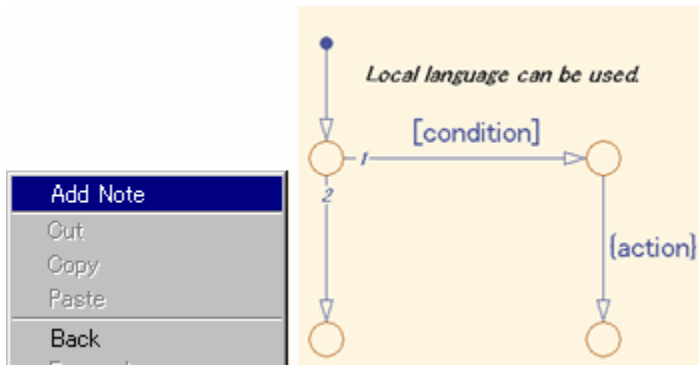


## Stateflow Examples

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



- Annotation description added using **Add Note**



---

**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

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

# Model Architecture

---

- “Simulink and Stateflow Partitioning” on page 4-2
- “Subsystem Hierarchies” on page 4-14
- “J-MAAB Model Architecture Decomposition” on page 4-31

This document uses the term *basic blocks* to refer to blocks built into the Simulink block libraries. “Basic Blocks” on page D-2 in Appendix D lists some examples of basic blocks.

### **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

## ID: Title

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

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

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.

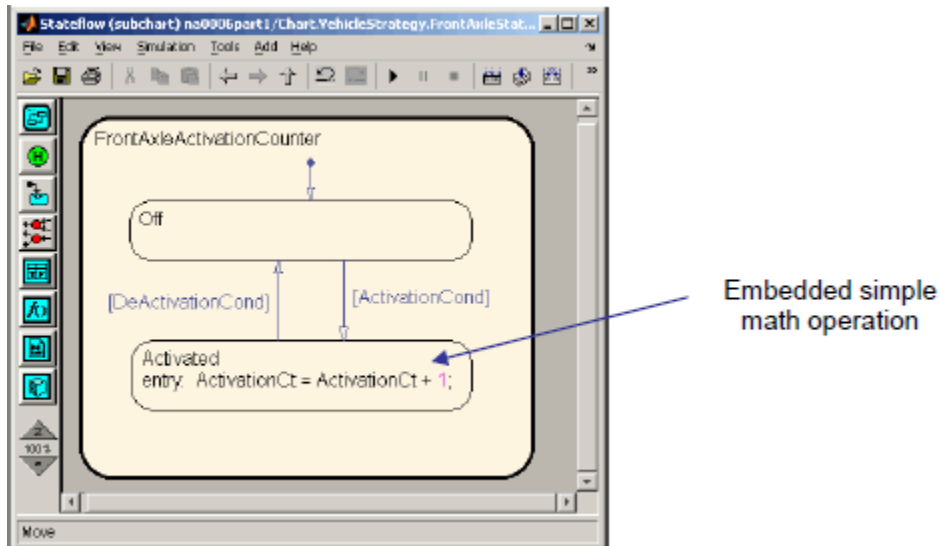
- If the function primarily involves complicated logical operations, use Stateflow diagrams.

Use Stateflow diagrams to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*.

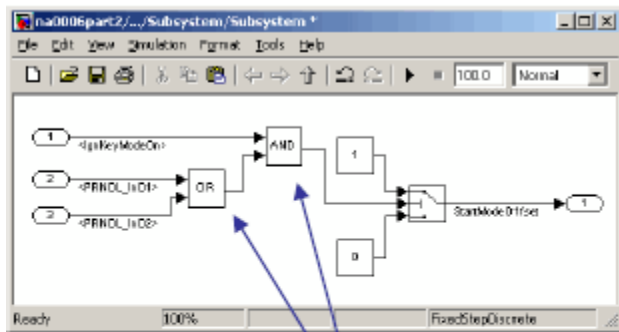
- If the function primarily involves numerical operations, use Simulink features.

### Specifics

- 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.

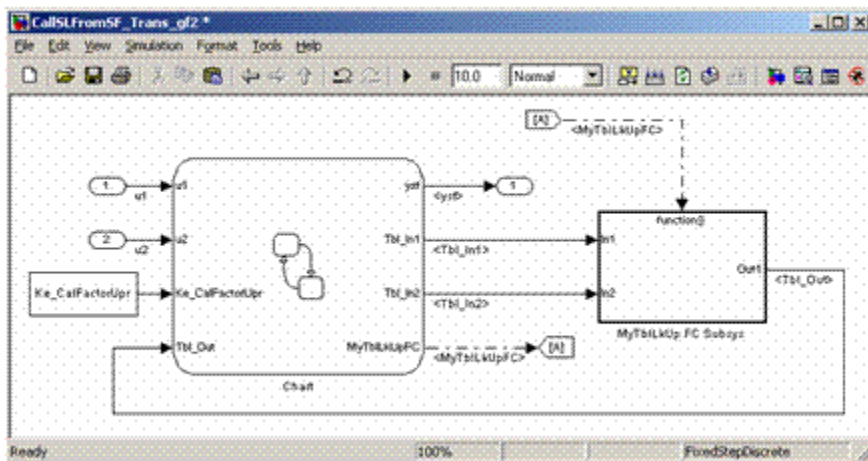
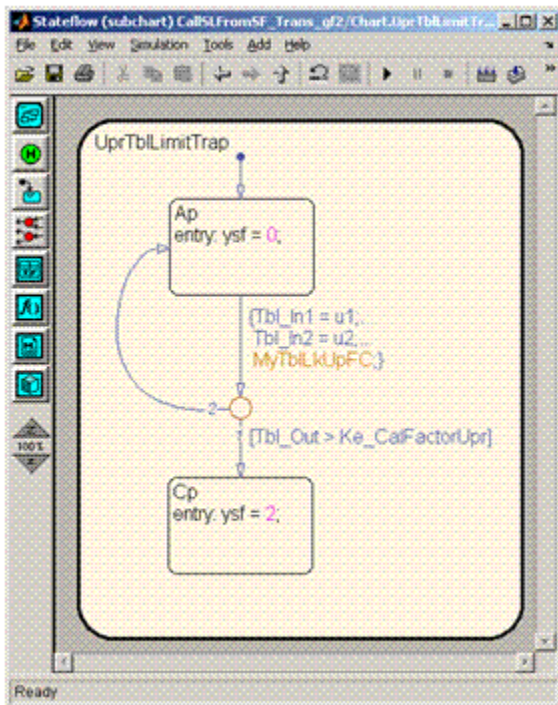


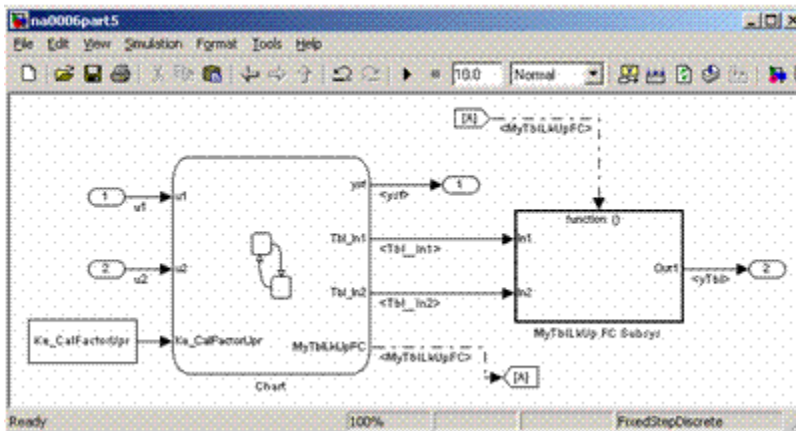
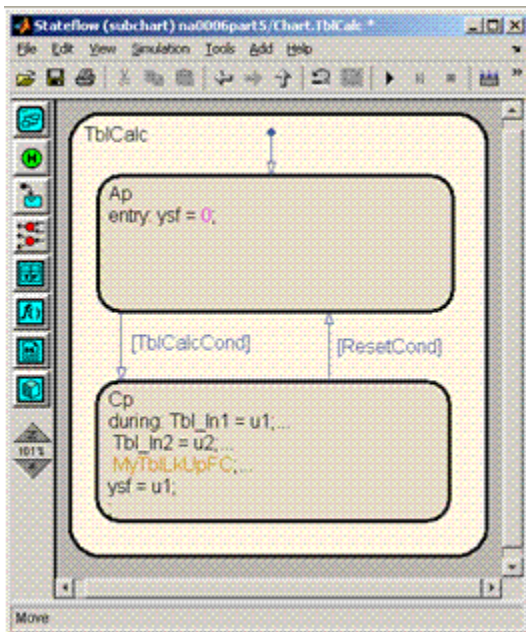
- If the primary nature of the function is numeric, but some simple logical operations are done to support the arithmetic, implement the simple logical functions with Simulink blocks.



Embedded simple  
logic operations

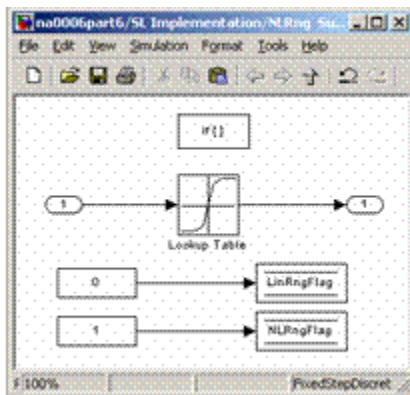
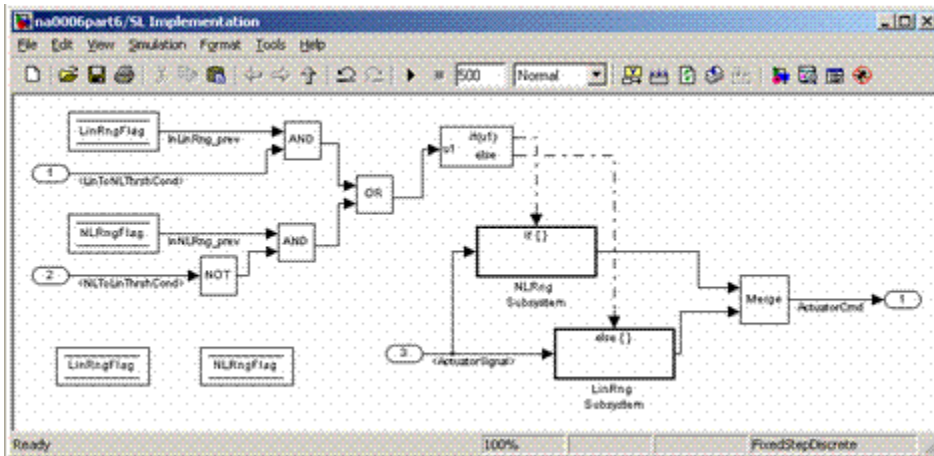
- If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, use a Simulink subsystem to implement the numerical calculations. The Stateflow software should invoke the execution of the subsystem, using a function call.



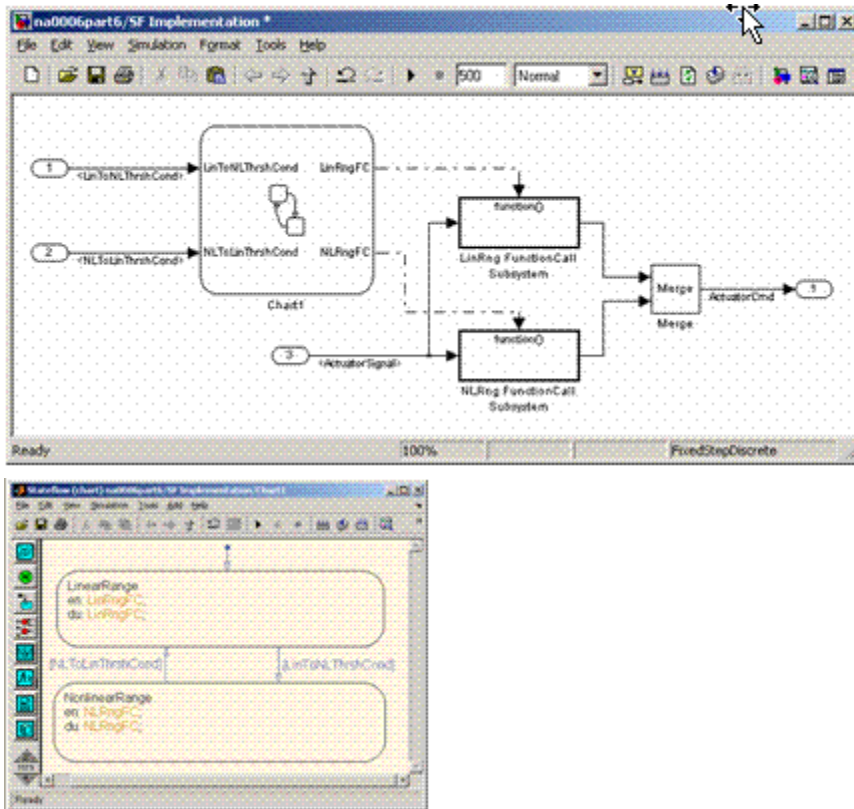


- Use the Stateflow product to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*. (If there is a need to store the result of a logical condition test in a Simulink model, for example, by storing a flag, this is an indicator of the presence of modal logic, which should be modeled with Stateflow software.)

## 4 Model Architecture



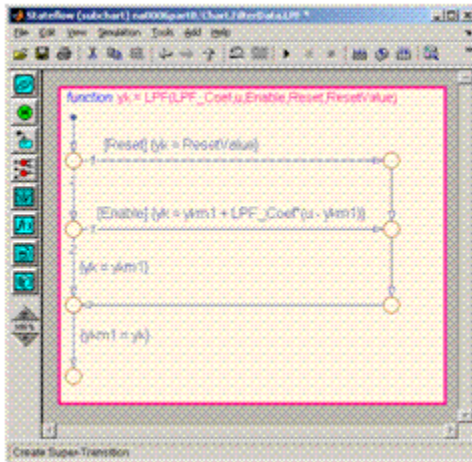
Incorrect



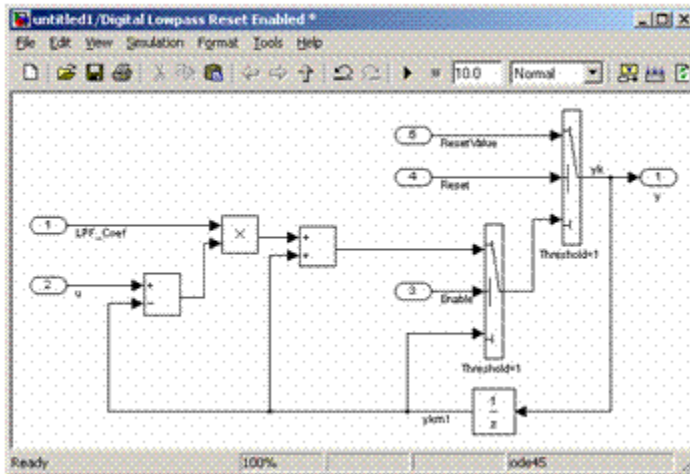
### Correct

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

## 4 Model Architecture



Incorrect



Correct



## Rationale

- Readability
- Workflow
- Simulation
- Verification and Validation
- Code Generation

## See Also

- “Driving Function Call Subsystems and Charts from Stateflow® Using Function Call Outputs”

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

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

## Description

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.

- If the primary nature of the function segment is to calculate modes of operation or discrete-valued states, use state charts. Some examples are:

- Diagnostic models with pass, fail, abort, and conflict states
- Model that calculates different modes of operation for a control algorithm
- If the primary nature of the function segment involves **if-then-else** statements, use flow charts or truth tables.

## Specifics

If the primary nature of a function segment is to calculate modes or states, but **if-then-else** statements are required, add a flow chart to a state within the state chart. (See “Flow Chart Patterns” on page 7-72.)

## Rationale

- Readability
- Workflow
- Simulation
- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

### Subsystem Hierarchies

- db\_0143: Similar block types on the model levels
- db\_0144: Use of Subsystems
- db\_0040: Model hierarchy
- na\_0037: Use of single variable variant conditionals
- na\_0020: Number of inputs to variant subsystems
- na\_0036: Default variant

# 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

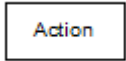


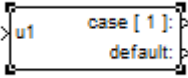

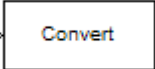


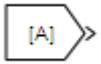

## Prerequisites




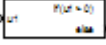







None



## Description

To allow partitioning of the model into discrete units, every level of a model must be designed with building blocks of the same type (i.e. only Subsystems or only “Basic Blocks”). The blocks listed in this guideline are used for signal routing. You can place them at any level of the model.

### Blocks that You Can Place at any Model Level

Block	Example
Action port <sup>1</sup>	
Bus Creator	
Bus Selector	
Case	
Data Store Memory	
Data Type Conversion	
Demux	
Enable <sup>2</sup>	
From	
Function-Call Generator	

Block	Example
Function-Call Split	
Goto	
Ground	
If	
Inport	
Merge	
Mux	
Outport	
Rate Transition	
Selector	
Terminator	

Block	Example
Trigger <sup>3</sup>	
Unit Delay	
<p><sup>1</sup>Action ports are not allowed at the root level of a model.</p> <p><sup>2</sup>Starting in R2011b, the Enable block is allowed at the root level of the model.</p> <p><sup>3</sup>Starting in R2009a, the Trigger block is allowed at the root level of the model.</p> <p><b>Note:</b> If the Trigger or Enable blocks are placed at the root level of the model, then the model will not simulate in a standalone mode. The model must be referenced using the Model block.</p>	

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V2.2

## Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > Check for systems that mix primitive blocks and subsystems

For check details, see “Check for mixing basic blocks and subsystems”.



**Introduced in R2010a**

## db\_0144: Use of Subsystems

### ID: Title

db\_0144: Use of Subsystems

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

Blocks in a Simulink diagram should be grouped together into subsystems based on functional decomposition of the algorithm, or portion thereof, represented in the diagram.

Grouping blocks into subsystems primarily for the purpose of saving space in the diagram should be avoided. Each subsystem in the diagram should represent a unit of functionality required to accomplish the purpose of the model or submodel. Blocks can also be grouped together based on behavioral variants or timing.

If creation of a subsystem is required for readability issues, then a virtual subsystem should be used.

## **Rationale**

- Readability
- Workflow
- Verification and Validation
- Code Generation

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## db\_0040: Model hierarchy

### ID: Title

db\_0040: Model hierarchy

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

The model hierarchy should correspond to the functional structure of the control system.

### Rationale

- Readability

- Workflow
- Verification and Validation
- Code Generation

## **Last Changed**

V2.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## na\_0037: Use of single variable variant conditionals

### ID: Title

na\_0037: Use of single variable variant conditionals

### Priority

Recommended

### Scope

NA-MAAB

### MATLAB Versions

All




### Prerequisites

None




### Description

Variant conditional expressions should be composed using either a single variable with compound conditions or multiple variables with a single condition. The default variant is an exception to the second rule.




**Correct:** Multiple variables (INLINE / FUNCTION with single condition per line

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	Default_F_of_A	DefaultVar	(INLINE==0) && (FUNC==0)
	Function_F_of_A	FunctionVar	(FUNC==1)
	Inline_F_of_A	InLineVar	(INLINE==1)

### Correct: Single variable compound conditions

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	autoTrans	autoTrans	(transType==3)  ((transType==4)  ((transType==5)
	defaultTrans	defaultTrans	(transType~3)&&(transType~4)&&(transType~5)&&(transType~0)
	manualTrans	manualTrans	(transType==0)

### Incorrect: Multiple variables, compound conditions

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	autoTrans	incorrect_1	(INLINE==0)&&(transType==3)
	defaultTrans	incorrect_Default	((INLINE==0)&&(transType==3)==0) &&(FUNC==0) && (transType~2)
	manualTrans	incorrect_2	(FUNC==1)&&(transType==2)

## Note

Use of enumerated variables is preferred in the Condition expressions. To improve the readability of the screenshots used in the examples, numerical values were used.

## Rationale

- Readability
- Code Generation
- Simulation

## See Also

- na\_0036: Default variant

## **Last Changed**

V3.0

**Introduced in R2013a**



# na\_0020: Number of inputs to variant subsystems

## ID: Title

na\_0020: Number of inputs to variant subsystems

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

Simulink requires variant subsystems to have the same number of inputs. However, the variant subsystem might not use all of the inputs. In these instances, terminate the unused inputs with the Terminator block.

## Rationale

- Readability

- Code Generation
- Simulation

### **Last Changed**

V3.0

**Introduced in R2013a**

# na\_0036: Default variant

## ID: Title

na\_0036: Default variant

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites




na\_0037: Use of single variable variant conditionals

## Description




All Variant subsystems and models should be configured so that one subsystem is always selected. This can be achieved by doing one of the following:

- Using a default variant.
- Defining conditions that exhaustively cover all possible values of the conditional variables. For example, defining conditions for true and false values of a Boolean.



## Correct

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	Default_F_of_A	defaultVar	(FUNC~=1)&&(FUNC~=2)
	Function_F_of_A	functionVar	(FUNC==1)
	Inline_F_of_A	inLineVar	(FUNC==2)

## Correct: Assumes FUNC and INLINE are Boolean

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	Default_F_of_A	DefaultVar	(INLINE==0) && (FUNC==0)
	Function_F_of_A	FunctionVar	(FUNC==1)
	Inline_F_of_A	InLineVar	(INLINE==1)

## Incorrect: No active subsystem iff FUNC not equal to 1 or 2.

Variant choices (list of child subsystems)			
	Name (read-only)	Variant object	Condition (read-only)
	Function_F_of_A	functionVar	(FUNC==1)
	Inline_F_of_A	inLineVar	(FUNC==2)

## Rationale

- Readability
- Code Generation
- Simulation

## Last Changed

V3.0

Introduced in R2013a

## **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

### ID: Title

jc\_0301: Controller model

### Priority

Mandatory

### Scope

J-MAAB

### MATLAB Versions

All

### Prerequisites

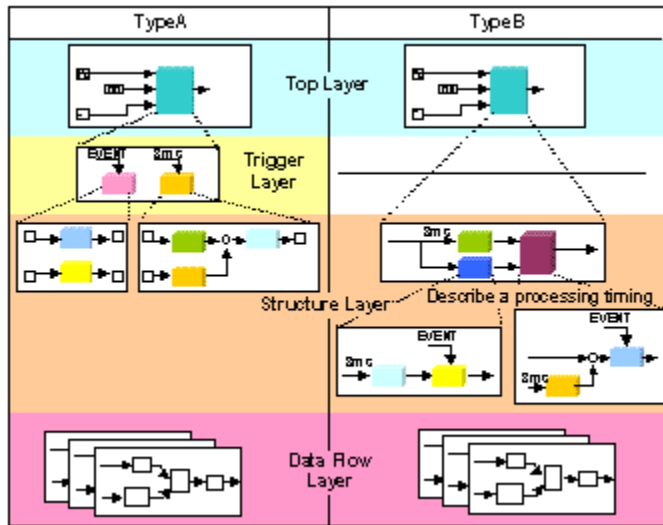
None

### Description

Control models are organized using the following hierarchical structure. Details on each layer are provided in corresponding rules.

- Top layer (root level), jc\_0311: Top layer/root level
- Trigger layer, jc\_0321: Trigger layer
- Structure layer, jc\_0331: Structure layer
- Data flow layer, jc\_0341: Data flow layer

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.



## Controller Model

## Rationale

Workflow

## Last Changed

V2.0

## Model Advisor Check

Not applicable

Introduced in R2010a

## jc\_0311: Top layer/root level

### ID: Title

jc\_0311: Top layer/root level

### Priority

Mandatory

### Scope

J-MAAB

### MATLAB Versions

All

### Prerequisites

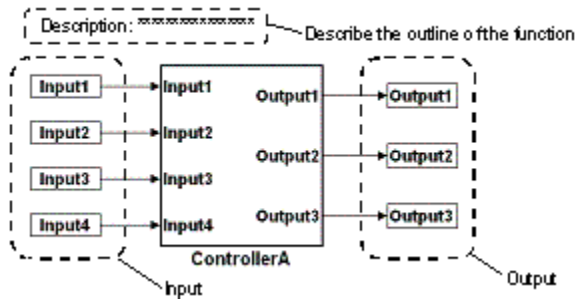
None

### 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

Introduced in R2010a

## jc\_0321: Trigger layer

### ID: Title

jc\_0321: Trigger layer

### Priority

Mandatory

### Scope

J-MAAB

### MATLAB Versions

All

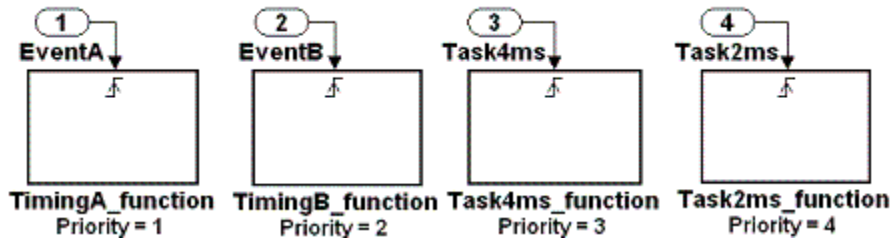
### Prerequisites

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

Introduced in R2010a

## jc\_0331: Structure layer

### ID: Title

jc\_0331: Structure layer

### Priority

Mandatory

### Scope

J-MAAB

### MATLAB Versions

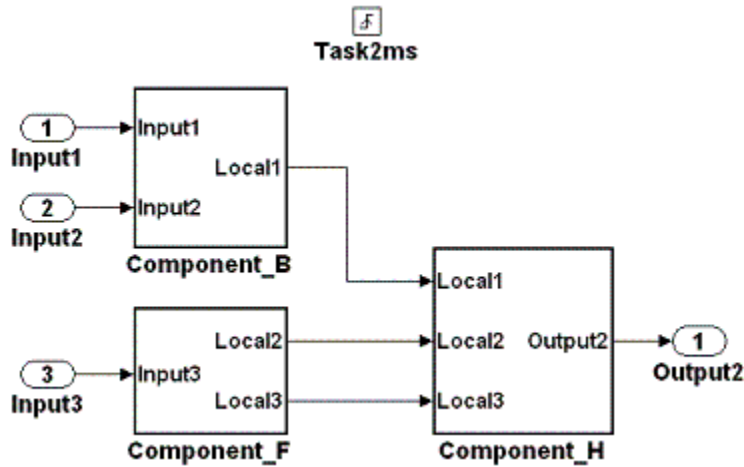
All

### Prerequisites

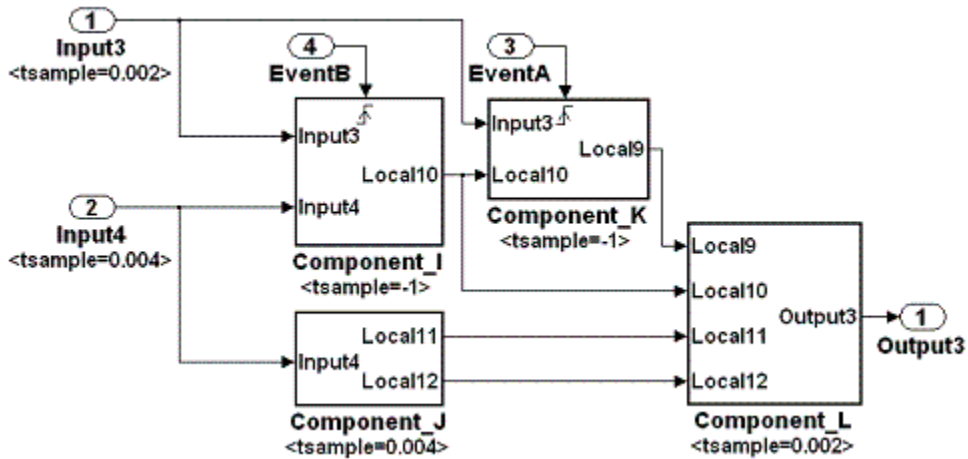
None

### 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

- Readability

- Workflow
- Code Generation

### **Last Changed**

V2.0

### **Model Advisor Check**

Not applicable

**Introduced in R2010a**

# jc\_0341: Data flow layer

## ID: Title

jc\_0341: Data flow layer

## Priority

Mandatory

## Scope

J-MAAB

## MATLAB Versions

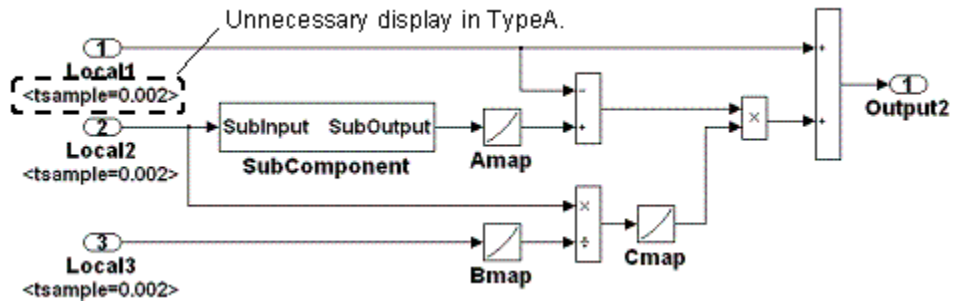
All

## 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

Introduced in R2010a



# 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

## ID: Title

jc\_0011: Optimization parameters for Boolean data types

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

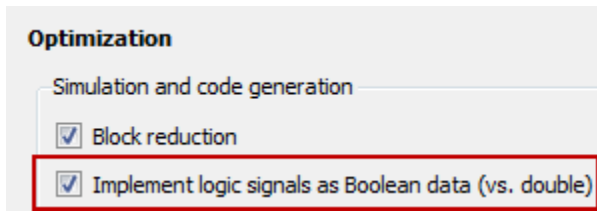
## Prerequisites

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

## Description

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

In the Configuration Parameters dialog box, on the **Optimization** pane, under **Simulation and code generation**, select **Implement logic signals as Boolean data (vs. double)**.



### Rationale

- Workflow
- Code Generation

### Last Changed

V2.2

### Model Advisor Check

**By Task > Modeling Standards for MAAB > Model Configuration Options > Check Implement logic signals as Boolean data (vs. double)**

For check details, see “Check Implement logic signals as Boolean data (vs. double)”.

**Introduced in R2010a**

# 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 Output 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**

**By Task > Modeling Standards for MAAB > Model Configuration Options > Check model diagnostic settings**

For check details, see “Check model diagnostic parameters”.

**Introduced in R2010a**





# Simulink

---

- “Diagram Appearance” on page 6-2
- “Signals” on page 6-42
- “Block Usage” on page 6-53
- “Block Parameters” on page 6-80
- “Simulink Patterns” on page 6-88

## Diagram Appearance

- na\_0004: Simulink model appearance
- db\_0043: Simulink font and font size
- db\_0042: Port block in Simulink models
- na\_0005: Port block name visibility in Simulink models
- jc\_0081: Icon display for Port block
- jm\_0002: Block resizing
- db\_0142: Position of block names
- jc\_0061: Display of block names
- db\_0146: Triggered, enabled, conditional Subsystems
- db\_0140: Display of basic block parameters
- db\_0032: Simulink signal appearance
- db\_0141: Signal flow in Simulink models
- jc\_0171: Maintaining signal flow when using Goto and From blocks
- na\_0032: Use of merge blocks
- jm\_0010: Port block names in Simulink models
- jc\_0281: Naming of Trigger Port block and Enable Port block

# 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.

View Options	Setting
Model Browser	Unchecked
Screen color	White
Status Bar	Checked

<b>View Options</b>	<b>Setting</b>
Toolbar	Checked
Zoom factor	Normal (100%)

<b>Block Display Options</b>	<b>Setting</b>
Background Color	White
Foreground Color	Black
Execution Context Indicator	Unchecked
Library Link Display	None
Linearization Indicators	Checked
Model/Block I/O Mismatch	Unchecked
Model Block Version	Unchecked
Sample Time Colors	Unchecked
Sorted Order	Unchecked

<b>Signal Display Options</b>	<b>Setting</b>
Port Data Types	Unchecked
Signal Dimensions	Unchecked
Storage Class	Unchecked
Test point Indicators	Checked
Viewer Indicators	Checked
Wide Nonscalar Lines	Checked

## Rationale

- Readability

## Last Changed

V2.0

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check for Simulink diagrams that have nonstandard appearance attributes**

For check details, see “Check for Simulink diagrams using nonstandard display attributes”.

**Introduced in R2010a**

## db\_0043: Simulink font and font size

### ID: Title

db\_0043: Simulink font and font size

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

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).

---

## Rationale

- Readability

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for difference in font and font sizes**

For check details, see “Check font formatting”.

**Introduced in R2010a**

## db\_0042: Port block in Simulink models

### ID: Title

db\_0042: Port block in Simulink models

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

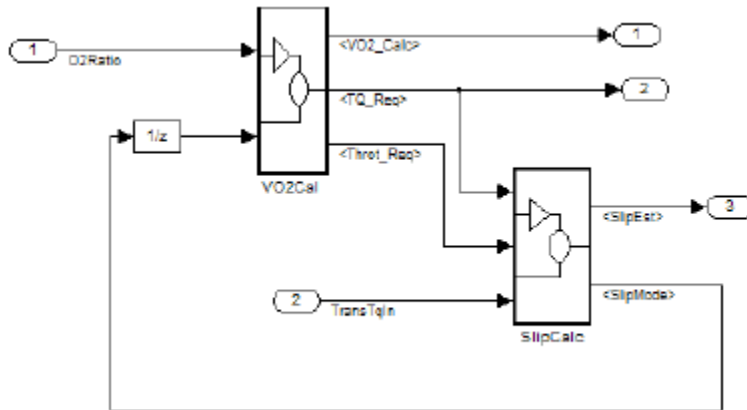
### Description

In a Simulink model, ports must comply with the following rules:

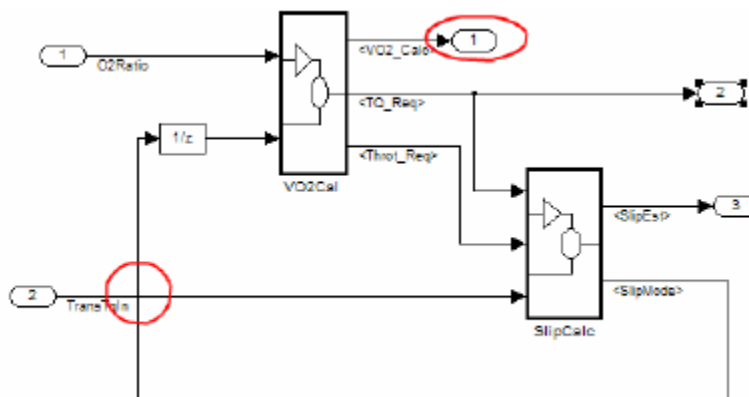
- Place Inport blocks on the left side of the diagram; you may move them to prevent signal crossings.
- Place Outport blocks on the right side of the diagram; you may move them to prevent signal crossings.



- You may use duplicate Inport blocks at the subsystem level, if required, but avoid doing so, if possible.
- Do not use duplicate Inport blocks at the root level.



**Correct**



**Incorrect**

Notes on the incorrect model

- Inport 2 should be moved in so it does not cross the feedback loop lines.
- Output 1 should be moved to the right side of the diagram.

## **Rationale**

Readability

## **Last Changed**

V2.0

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check for invalid port positioning and configuration**

For check details, see “Check positioning and configuration of ports”.

**Introduced in R2010a**

# 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

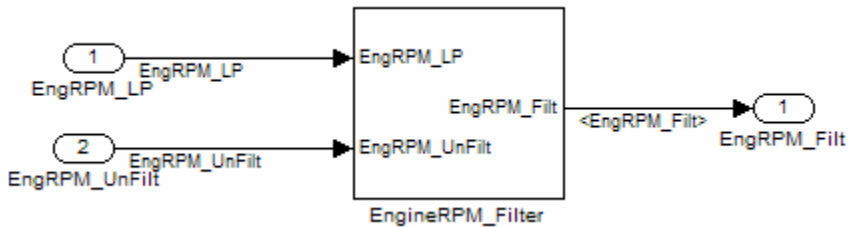
None

## Description

For some items it is not possible to define a single approach that is applicable to all organizations' internal processes. However, it is important that within a given organization, a single consistent approach is followed. An organization applying the guidelines must enforce one of the following alternatives.

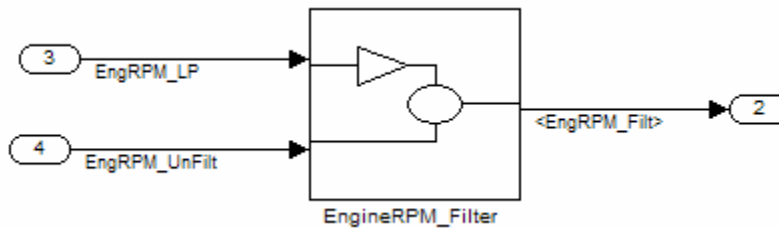
Apply one of the following practices:

- 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.



**Correct: Use of signal label**

## Rationale

Readability

## Last Changed

V2.0

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check visibility of port block names**

For check details, see “Check visibility of block port names”.

**Introduced in R2010a**

## 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

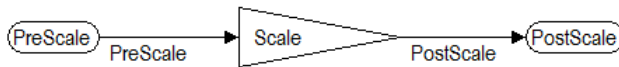
The Icon display setting should be set to Port number for Inport and Outport blocks.



**Correct**



**Incorrect**



**Incorrect**

## Rationale

Readability

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check display for port blocks**

For check details, see “Check display for port blocks”.

**Introduced in R2010a**

## **jm\_0002: Block resizing**

### **ID: Title**

jm\_0002: Block resizing

### **Priority**

Mandatory

### **Scope**

MAAB

### **MATLAB Versions**

All

### **Prerequisites**

None

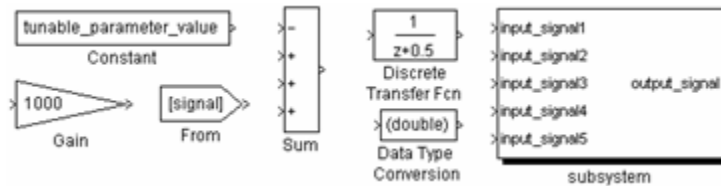
### **Description**

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.

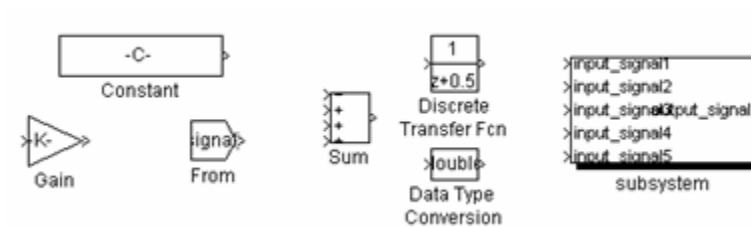
This guideline requires that you resize blocks with variable icons or blocks with a variable number of inputs and outputs. In some cases, it may not be practical or desirable to resize the icon of a subsystem block so that all of the input and output names within



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**



**Incorrect**

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

# db\_0142: Position of block names

## ID: Title

db\_0142: Position of block names

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

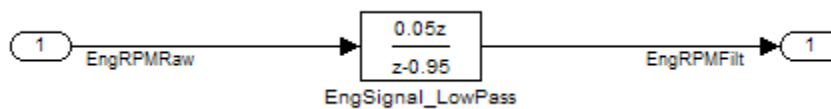
All

## Prerequisites

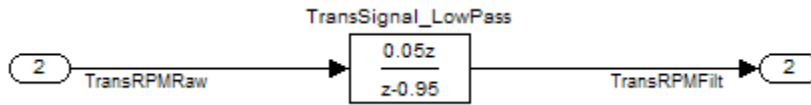
None

## Description

If shown, place the name of a block below the block.



**Correct**



**Incorrect**

## Rationale

- Readability

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check whether block names do not appear below blocks**

For check details, see “Check whether block names appear below blocks”.

**Introduced in R2010a**

# jc\_0061: Display of block names

## ID: Title

jc\_0061: Display of block names

## Priority

Recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

- Display a block name when it provides descriptive information.
- Do not display a block name if the block function is known and understood from the block appearance.

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check the display attributes of block names**

For check details, see “Check the display attributes of block names”.

**Introduced in R2010a**

# db\_0146: Triggered, enabled, conditional Subsystems

## ID: Title

db\_0146: Triggered, enabled, conditional Subsystems

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

The blocks that define subsystems as either conditional or iterative should be located at a consistent location at the top of the subsystem diagram. These blocks are:

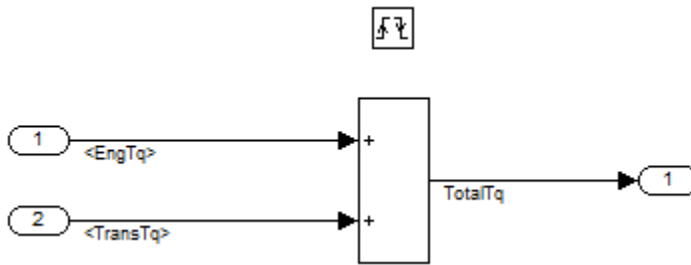
- Action Port
- Enable
- For Iterator

- Switch Case Action
- Trigger
- While Iterator

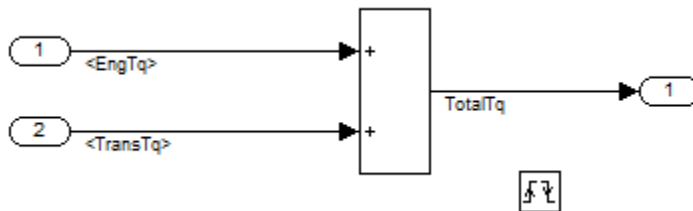
---

**Note:** The Action Port is associated with the If and Case blocks. The Trigger port is also the function-call block.

---



**Correct**



**Incorrect**

## Rationale

- Readability

## Last Changed

V2.2



## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check position of Trigger and Enable blocks**

For check details, see “Check position of Trigger and Enable blocks”.

**Introduced in R2010a**

## db\_0140: Display of basic block parameters

### ID: Title

db\_0140: Display of basic block parameters

### Priority

Recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

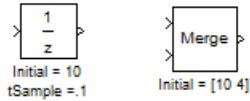
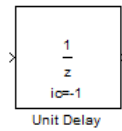
### Description

Important block parameters modified from the default values should be displayed.

---

**Note:** The attribute string is one method to support the display of block parameters. The block annotation tab allows you to add the desired attribute information. As of R2011b, masking basic blocks is a supported method for displaying the information. This method is allowed if the base icon is distinguishable.

---

**Correct****Correct: Masked block****Rationale**

- Readability

**Last Changed**

V2.2

**Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check for display of nondefault block attributes**

For check details, see “Check for nondefault block attributes”.

**Introduced in R2010a**

## db\_0032: Simulink signal appearance

### ID: Title

db\_0032: Simulink signal appearance

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

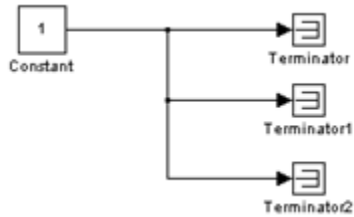
None

### Description

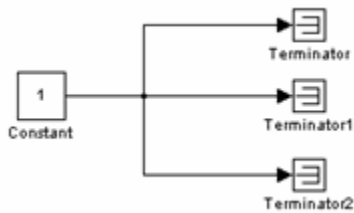
Signal lines

- 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 sublines at a single branching point



**Correct**



**Incorrect**

## Rationale

- Readability

## Last Changed

V2.0

## Model Advisor Check

Not applicable

Introduced in R2010a

## 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

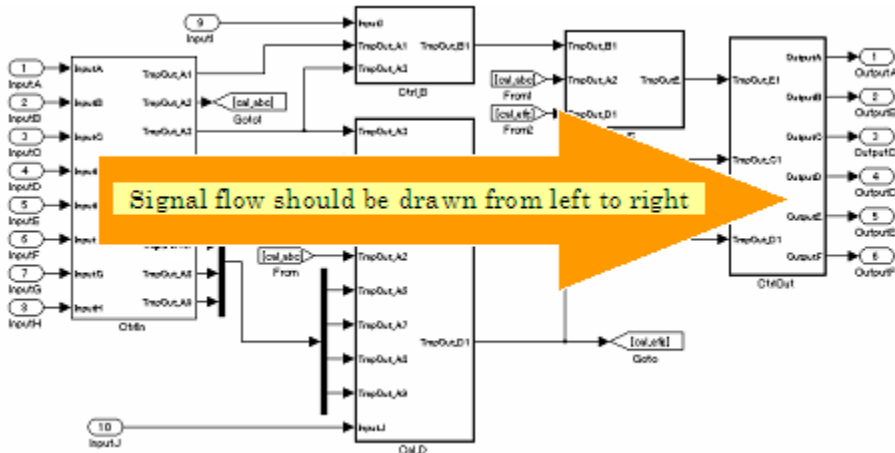
The signal flow in a model is from left to right.

**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

## Last Changed

V2.0

## Model Advisor Check

Not applicable

Introduced in R2010a

# 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

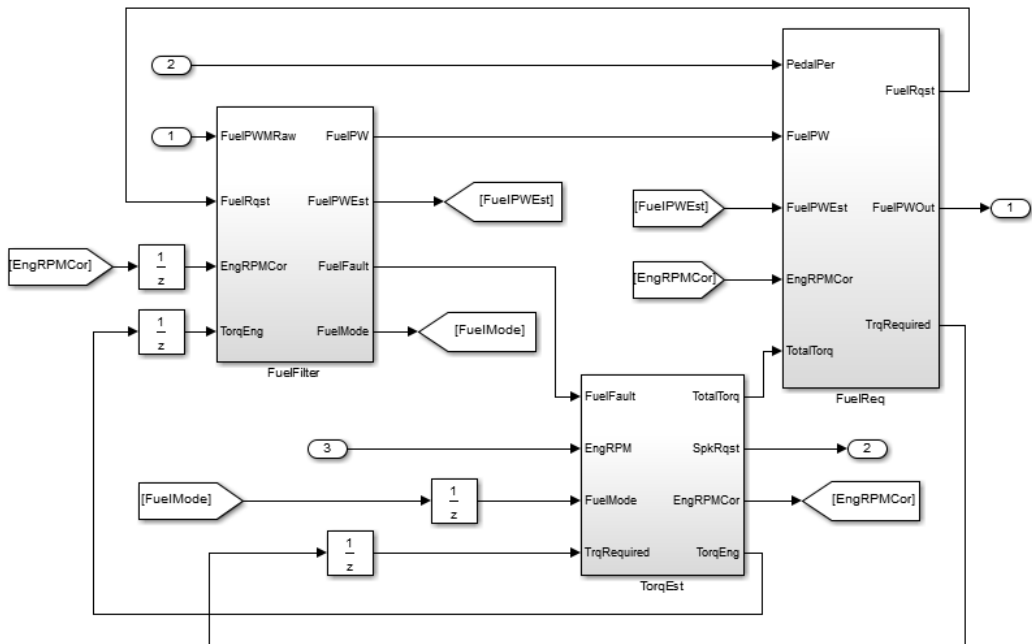
None

## Description

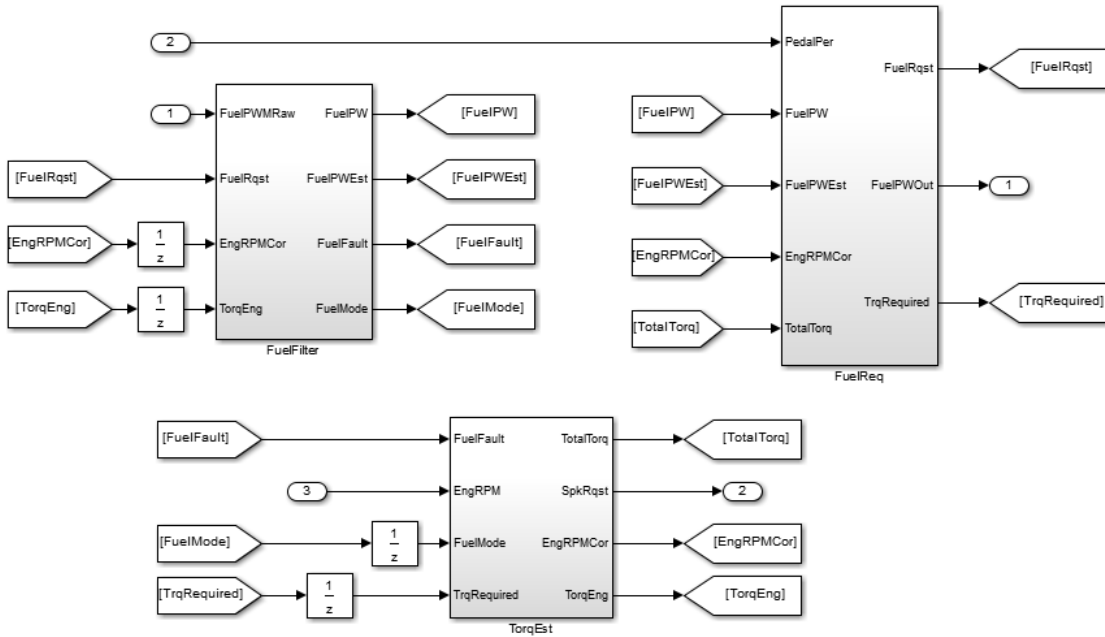
- You must maintain visual depiction of signal flow between subsystems.
- You can use Goto and From blocks if:



- You use at least one signal line between connected subsystems.
- Subsystems connected in a feed-forward and feedback loop have at least one signal line for each direction.
- Using Goto and From blocks to create buses or connect inputs to merge blocks are exceptions to this rule.



Correct



Incorrect

## Rationale

- Readability

## Last Changed

V2.2

## Model Advisor Check

Not applicable

**Introduced in R2010a**

## na\_0032: Use of merge blocks

### ID: Title

na\_0032: Use of merge blocks

### Priority

Strongly recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

When using Merge blocks:

- Signals entering a merge block must not branch off to other blocks
- With buses:
  - Buses must be identical This includes:
    - Number of elements

- Element names
- Element order
- Element data type
- Element size
- Buses must be either all virtual or all nonvirtual
- Bus lines entering a merge block must not branch off to other blocks.

## **Rationale**

- Workflow
- Code Generation

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**

# 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 options:

- **Names of Inport and Outport blocks must match 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 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 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
- Code Generation
- Simulation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for mismatches between names of ports and corresponding signals**

For check details, see “Check for matching port and signal names”.

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Strongly recommended

## Scope

J-MAAB

## MATLAB Versions

All

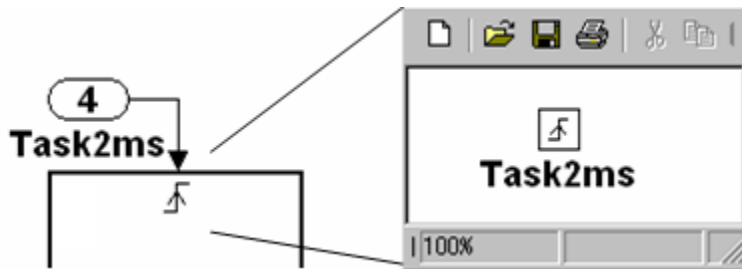
## Prerequisites

None

## Description

For Trigger and Enable port blocks, match the block name of the signal triggering the subsystem.





## Rationale

- Readability
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check Trigger and Enable block port names**

For check details, see “Check Trigger and Enable block names”.

**Introduced in R2010a**

## Signals

- na\_0008: Display of labels on signals
- na\_0009: Entry versus propagation of signal labels
- db\_0097: Position of labels for signals and busses
- db\_0081: Unconnected signals, block inputs and block outputs

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

## ID: Title

na\_0008: Display of labels on signals

## Priority

Recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

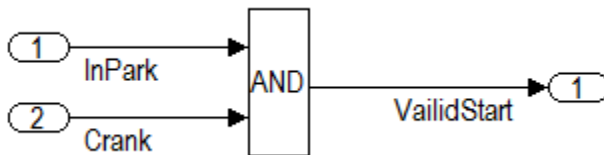
- A label must be displayed on a signal originating from the following blocks:
  - Inport block
  - From block (block icon exception applies – see the following Note)
  - Subsystem block or Stateflow chart block (block icon exception applies)
  - Bus Selector block (the tool forces this to happen)

- Demux block
- Selector block
- Data Store Read block (block icon exception applies)
- Constant block (block icon exception applies)
- 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):
  - Outport block
  - Goto block
  - Data Store Write block
  - Bus Creator block
  - Mux block
  - Subsystem block
  - Chart block

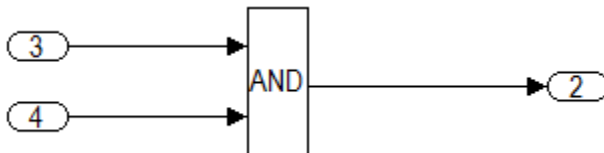
---

**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.

---



**Correct**



**Incorrect**

## Rationale

- Readability
- Verification and Validation
- Workflow
- Verification and Validation
- Code Generation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check signal line labels**

For check details, see “Check signal line labels”.

**Introduced in R2010a**

## na\_0009: Entry versus propagation of signal labels

### ID: Title

na\_0009: Entry versus propagation of signal labels

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

na\_0008: Display of labels on signals

### Description

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 (<) character).

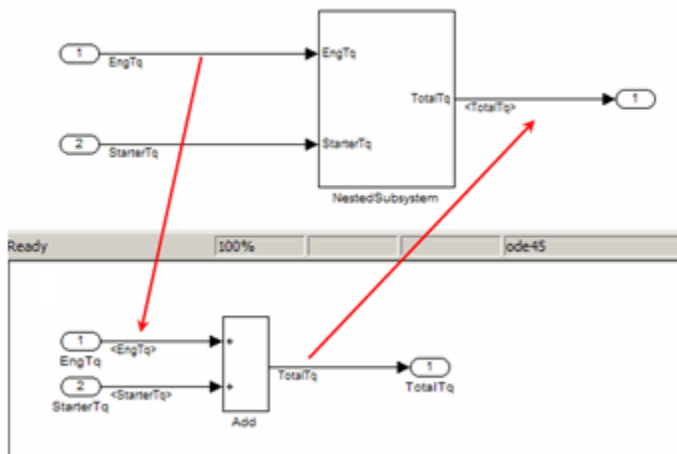
- Any displayed signal label must be *entered* for signals that:
  - Originate from an Inport at the Root (top) Level of a model

- 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.)
- Any displayed signal label must be *propagated* for signals that that:
  - 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.

- Originate from a basic block that performs a nontransformative operation
- Originate from a Subsystem or Stateflow chart block

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
- Verification and Validation

- Workflow
- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for propagated labels on signal lines**

For check details, see “Check for propagated signal labels”.

**Introduced in R2010a**



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

## ID: Title

db\_0097: Position of labels for signals and busses

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

The labels must be visually associated with the corresponding signal and not overlap other labels, signals, or blocks.

Labels should be located consistently below horizontal lines and close to the corresponding source or destination block.

## **Rationale**

- Readability

## **Last Changed**

V2.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Mandatory

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

A system must not have any:

- Unconnected subsystem or basic block inputs
- Unconnected subsystem or basic block outputs
- Unconnected signal lines

In addition:

- An otherwise unconnected input should be connected to a ground block
- An otherwise unconnected output should be connected to a terminator block

## Rationale

- Readability
- Workflow
- Verification and Validation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check whether model has unconnected block input ports, output ports, or signal lines**

For check details, see “Check for unconnected ports and signal lines”.

**Introduced in R2010a**

## Block Usage

- na\_0003: Simple logical expressions in If Condition block
- na\_0002: Appropriate implementation of fundamental logical and numerical operations
- jm\_0001: Prohibited Simulink standard blocks inside controllers
- hd\_0001: Prohibited Simulink sinks
- na\_0011: Scope of Goto and From blocks
- jc\_0141: Use of the Switch block
- jc\_0121: Use of the Sum block
- jc\_0131: Use of Relational Operator block
- jc\_0161: Use of Data Store Read/Write/Memory 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 || u2 || u3 || u4 || u5`
- `u1 && u2 && u3 && u4`

## Examples of Primary Expressions

- `u1`
- `5`
- `K`
- `(u1 > 0)`
- `(u1 <= G)`
- `(u1 > U2)`
- `(~u1)`
- `(EngineState.ENGINE_RUNNING)`

## Examples of Acceptable Logical Expressions

- `u1 || u2`
- `(u1 > 0) && (u1 < 20)`
- `(u1 > 0) && (u2 < u3)`

- `(u1 > 0) && (~u2)`
- `(EngineState.ENGINE_RUNNING > 0) && (PRNDLState.PRNDL_PARK)`

---

**Note:** In this example, `EngineState.ENGINE_RUNNING` and `PRNDLState.PRNDL_PARK` are enumeration literals.

---

## Examples of Unacceptable Logical Expressions

<code>u1 &amp;&amp; u2    u3</code>	(too many primary expressions)
<code>u1 &amp;&amp; (u2    u3)</code>	(unacceptable operator within primary expression)
<code>(u1 &gt; 0) &amp;&amp; (u1 &lt; 20) &amp;&amp; (u2 &gt; 5)</code>	(too many primary expressions that are not inputs)
<code>(u1 &gt; 0) &amp;&amp; ((2*u2) &gt; 6)</code>	(unacceptable operator within primary expression)

## Rationale

- Readability
- Workflow
- Code Generation

## Last Changed

V2.2

## Model Advisor Check

Not applicable

Introduced in R2010a



# 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

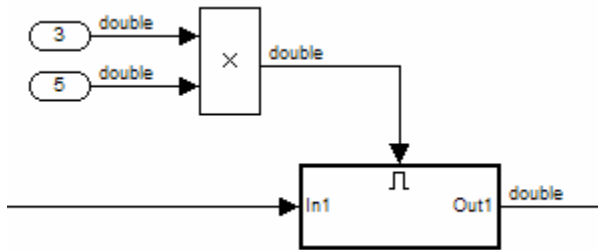
All

## Prerequisites

None

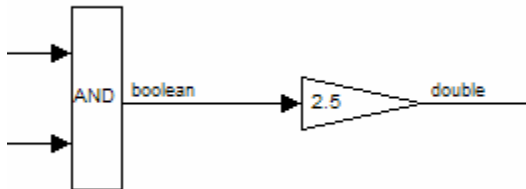
## Description

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



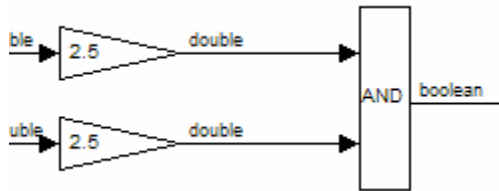
**Incorrect**

- 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.



**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
- Verification and Validation
- Workflow
- Code Generation

## Last Changed

V3.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

# 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.
- MathWorks “Simulink Block Data Type Support” table provides a list of blocks that support production code generation. See “Simulink Block Data Type Support”.

- Use blocks listed as “Code Generation Support.”
- Do not use blocks listed as “Not recommended for production code.” See footnote 4 in the table.
- In addition to the blocks defined by the above rule, do not use the following blocks:

The following sources *are not* allowed:

Band-Limited White Noise



Random Number



Pulse Generator



Uniform Random Number



Sine Wave



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



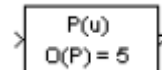
Real-Imag to Complex



Manual Switch



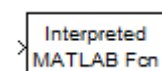
Polynomial



Complex to Magnitude-Angle



Interpreted MATLAB Function



Magnitude-Angle to Complex



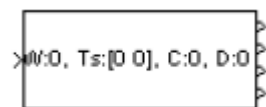
Goto Tag Visibility



Complex to Real-Imag



Probe



## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Simulation

## Last Changed

V2.2

## Model Advisor Checks

- **By Task > Modeling Standards for MAAB > Simulink > Check for blocks not recommended for C/C++ production code deployment**

For check details, see “Check for blocks not recommended for C/C++ production code deployment”.

- **By Task > Modeling Standards for MAAB > Simulink > Check for blocks that are not discrete**

For check details, see “Check for prohibited blocks in discrete controllers”.

**Introduced in R2010a**

# hd\_0001: Prohibited Simulink sinks

## ID: Title

hd\_0001: Prohibited Simulink sinks

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

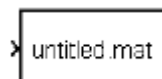
None

## Description

Controller models must be designed from discrete blocks.

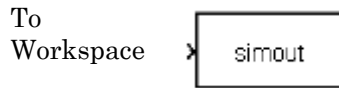
The following sink blocks *are not* allowed:

To File



Stop  
Simulation





---

**Note:** Simulink Scope and Display blocks are allowed in the model diagram. Consider using Simulink Signal logging and Signal and Scope Manager for data logging and viewing requirements.

---

## Rationale

- Verification and Validation
- Code Generation
- Simulation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for prohibited sink blocks**

For check details, see “Check for prohibited sink blocks”.

**Introduced in R2010a**



# na\_0011: Scope of Goto and From blocks

## ID: Title

na\_0011: Scope of Goto and From blocks

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

For signal flows, the following rules apply:

- From and Goto blocks must use local scope.

---

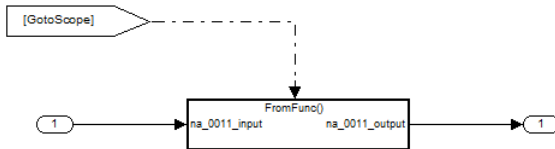
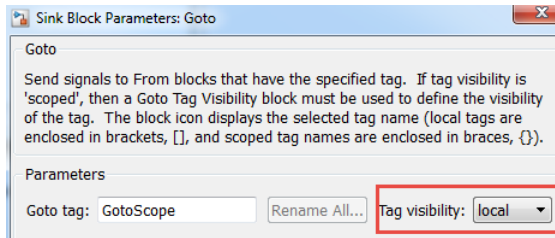
**Note:** Control flow signals may use global scope.

---

Control flow signals are output from:

- Function-call generators
- If and Case blocks
- Function call outputs from MATLAB and Stateflow blocks

Control flow signals are identified as dashed lines in the model after updating a Simulink model.



## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Simulation

## Last Changed

V2.2

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check for proper scope of From and Goto blocks**

For check details, see “Check scope of From and Goto blocks”.

**Introduced in R2010a**

# jc\_0141: Use of the Switch block

## ID: Title

jc\_0141: Use of the Switch block

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

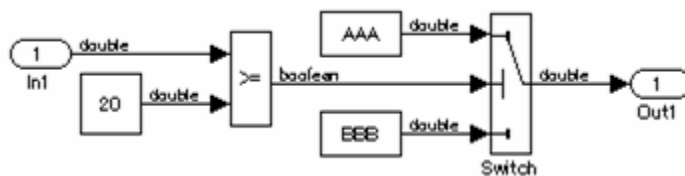
All

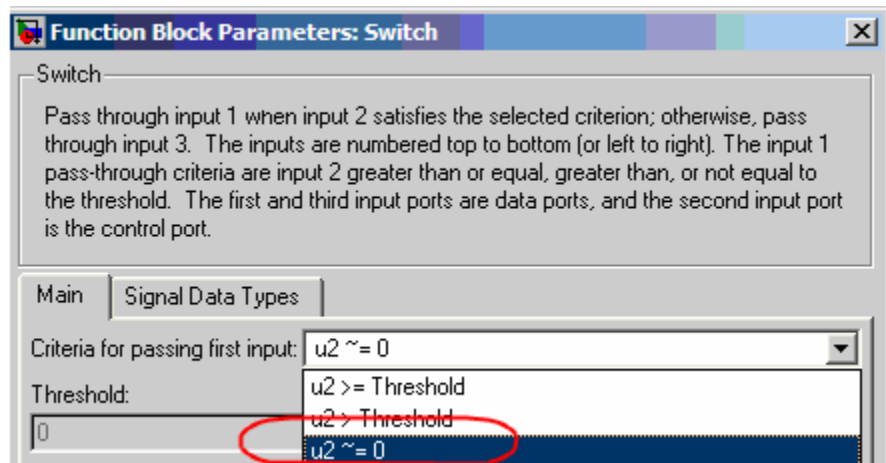
## Prerequisites

None

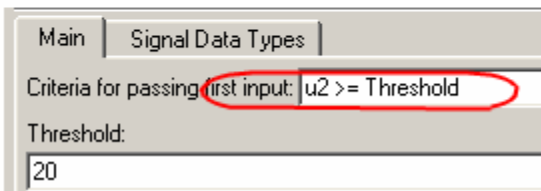
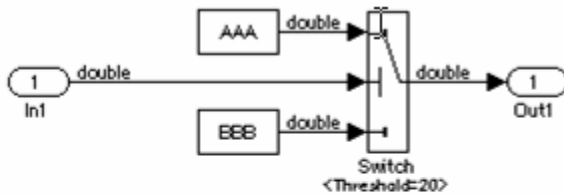
## Description

- The switch condition, input 2, must be a Boolean value.
- The block parameter, **Criteria for passing first input**, should be set to  $u2 \sim 0$ .





**Correct**



**Incorrect**

## Rationale

- Readability
- Verification and Validation
- Workflow

- Code Generation

## **Last Changed**

V2.2

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Simulink > Check for proper use of Switch blocks**

For check details, see “Check usage of Switch blocks”.

**Introduced in R2010a**

# jc\_0121: Use of the Sum block

## ID: Title

jc\_0121: Use of the Sum block

## Priority

Recommended

## Scope

MAAB

## MATLAB Versions

All

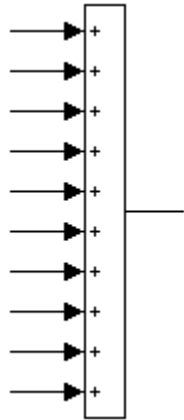
## Prerequisites

None

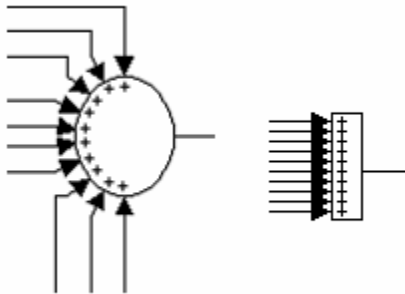
## Description

Sum blocks should:

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



**Correct**

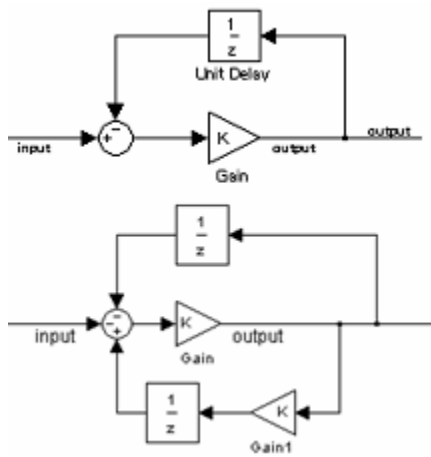


**Incorrect**

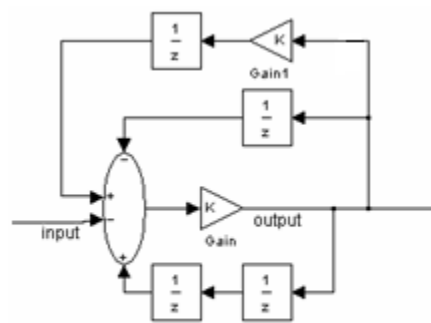
You may use the round shape in feedback loops.

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

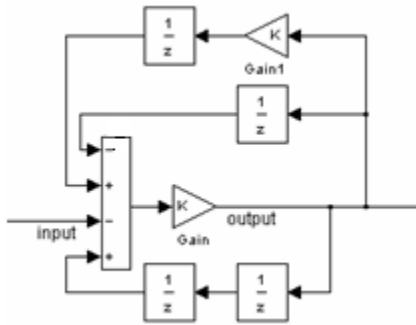




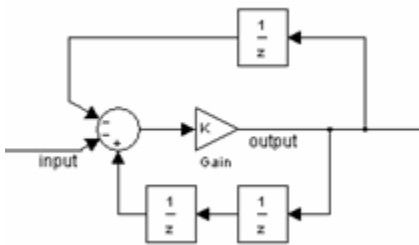
Correct



Incorrect



**Correct**



**Incorrect**

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

## jc\_0131: Use of Relational Operator block

### ID: Title

jc\_0131: Use of Relational Operator block

### Priority

Recommended

### Scope

J-MAAB

### MATLAB Versions

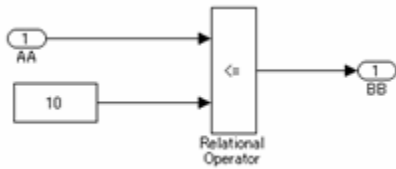
All

### Prerequisites

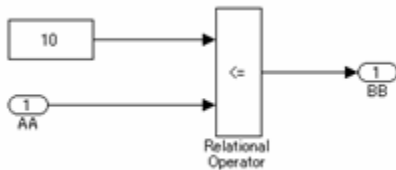
None

### 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

By Task > Modeling Standards for MAAB > Simulink > Check for proper position of constants used in Relational Operator blocks

For check details, see “Check usage of Relational Operator blocks”.

Introduced in R2010a

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

## ID: Title

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

## Priority

Strongly recommended

## Scope

J-MAAB

## MATLAB Versions

All

## Prerequisites

jc\_0341: Data flow layer

## Description

Data Store Memory, Data Store Read, and Data Store Write blocks are

- Prohibited in a data flow layer
- Allowed between subsystems running at different rates

## **Rationale**

- Readability
- Workflow

## **Last Changed**

V2.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## **Block Parameters**

- db\_0112: Indexing
- na\_0010: Grouping data flows into signals
- db\_0110: Tunable parameters in basic 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.



# db\_0112: Indexing

## ID: Title

db\_0112: Indexing

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

Use a consistent vector indexing method for all blocks.

When possible, use zero-based indexing to improve code efficiency. However, since MATLAB blocks do not support zero-based indexing, one-based indexing can be used for models containing MATLAB blocks.

## See Also

- “cgsl\_0101: Zero-based indexing”
- “hisl\_0021: Consistent vector indexing method”

## Rationale

- Readability
- Verification and Validation
- Code Generation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for blocks not using one-based indexing**

For check details, see “Check for indexing in blocks”.

**Introduced in R2010a**

# 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]
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
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check for proper use of signal buses and Mux block usage**

For check details, see “Check usage of buses and Mux blocks”.

**Introduced in R2010a**

## db\_0110: Tunable parameters in basic blocks

### ID: Title

db\_0110: Tunable parameters in basic blocks

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

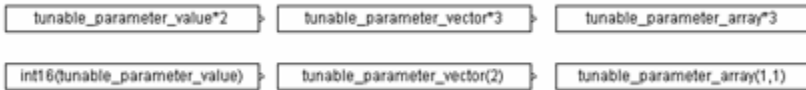
### Prerequisites

None

### Description

To ensure that a parameter is tunable, enter it in a block dialog field:

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

**Correct****Incorrect**

## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Simulation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Simulink > Check whether tunable parameters specify expressions, data type conversions, or indexing operations**

For check details, see “Check usage of tunable parameters in blocks”.

**Introduced in R2010a**

## Simulink Patterns

- na\_0012: Use of Switch vs. If-Then-Else Action Subsystem
- db\_0114: Simulink patterns for If-then-else-if constructs
- db\_0115: Simulink patterns for case constructs
- na\_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches
- db\_0116: Simulink patterns for logical constructs with logical blocks
- db\_0117: Simulink patterns for vector signals
- jc\_0351: Methods of initialization
- jc\_0111: Direction of 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

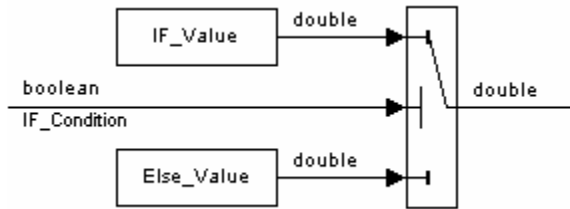
All

## Prerequisites

None

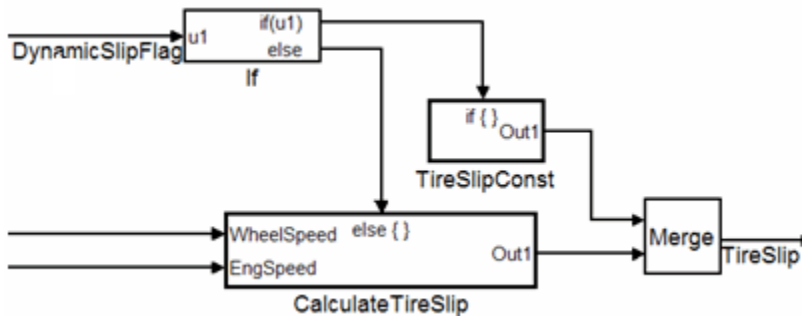
## Description

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.)



- 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
- Verification and Validation
- Workflow

## **Last Changed**

V2.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

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

### **ID: Title**

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

### **Priority**

Strongly recommended

### **Scope**

MAAB

### **MATLAB Versions**

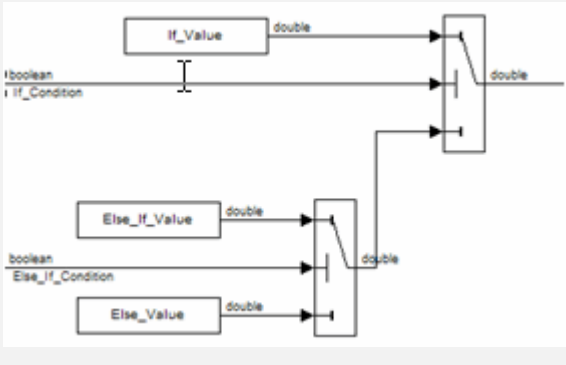
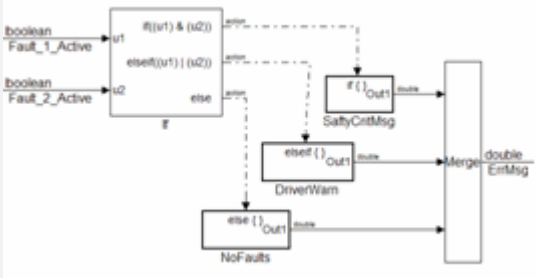
All

### **Prerequisites**

None

### **Description**

Use the following patterns for If-then-else-if constructs within a Simulink model:

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

## Last Changed

V2.0

## Model Advisor Check

Not applicable

**Introduced in R2010a**

# db\_0115: Simulink patterns for case constructs

## **ID: Title**

db\_0115: Simulink patterns for case constructs

## **Priority**

Strongly recommended

## **Scope**

MAAB

## **MATLAB Versions**

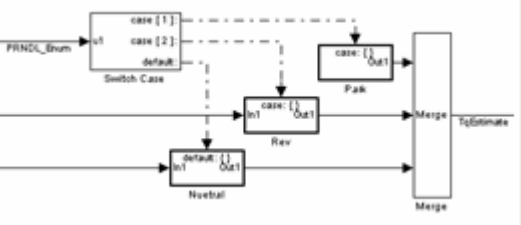
All

## **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 code. It features a 'Switch Case' block with three cases: 'case [1]' (Park), 'case [2]' (Rev), and 'default' (Neutral). Each case block has an input 'in1' and an output 'Out1'. The outputs of these three cases are connected to a 'Merge' block, which produces the final output 'TqEstimate'.</p>

## Rationale

- Readability

## Last Changed

V2.2

## Model Advisor Check

Not applicable

Introduced in R2010a



# na\_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches

## ID: Title

na\_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

All

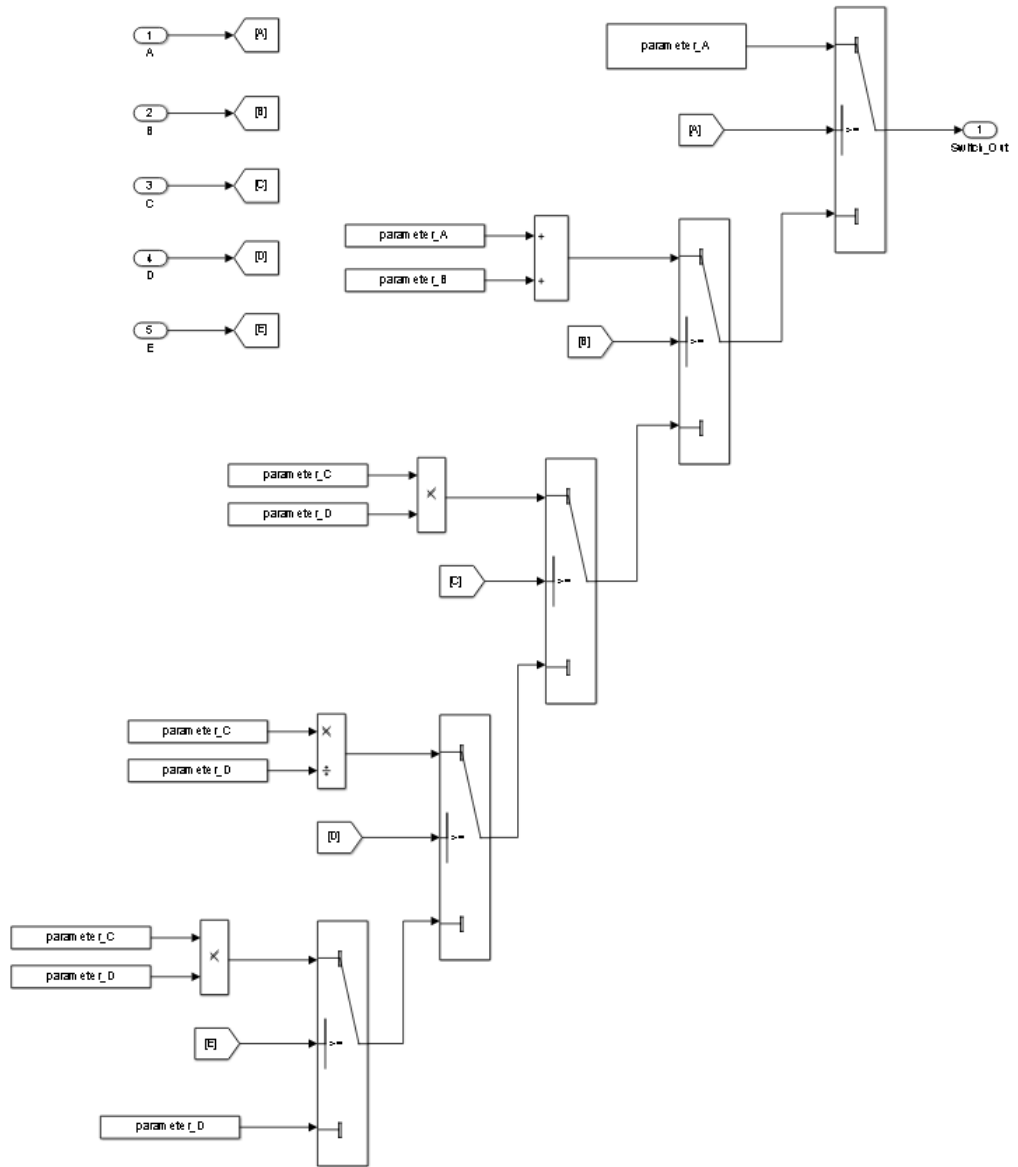
## Prerequisites

- na\_0012: Use of Switch vs. If-Then-Else Action Subsystem
- db\_0144: Use of Subsystems

## Description

The use of switch constructs should be limited, typically to 3 levels. Replace switch constructs that have more than 3 levels with an If-Then-Else action subsystem construct.

**Incorrect**



## **Rationale**

- Readability

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**

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

## ID: Title

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

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

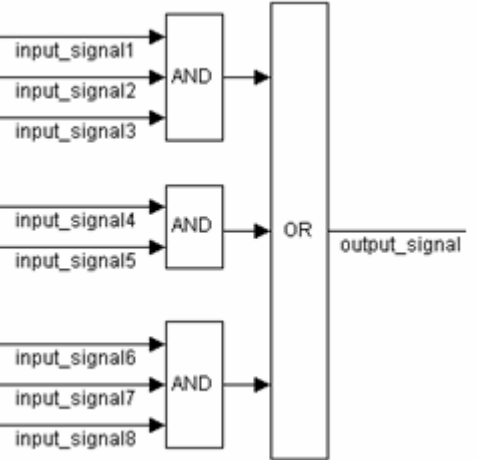
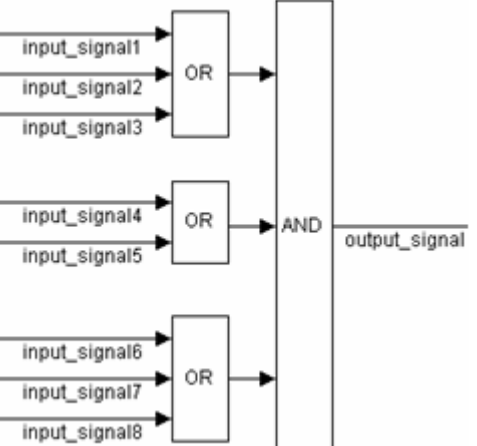
All

## Prerequisites

None

## Description

Use the following patterns for logical combinations within Simulink:

Equivalent Functionality	Simulink Pattern
Combination of logical signals: conjunctive	
Combination of logical signals: disjunctive	

## Rationale

- Readability

## **Last Changed**

V1.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

# db\_0117: Simulink patterns for vector signals

## ID: Title

db\_0117: Simulink patterns for vector signals

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

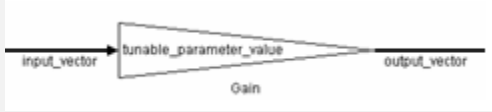
All


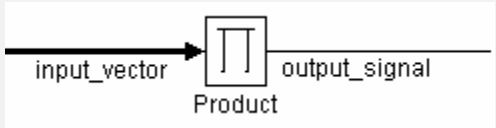
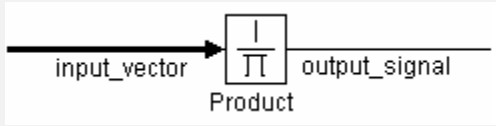
## Prerequisites

None

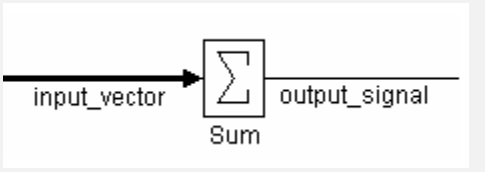
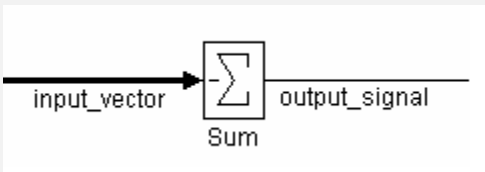
## Description

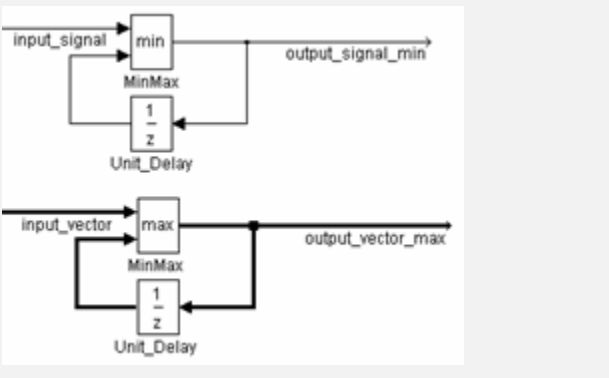
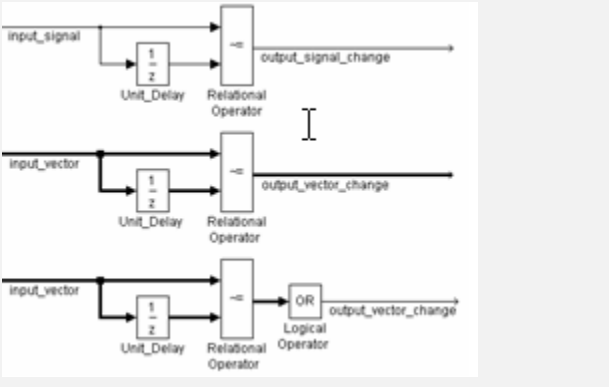
Simulink is a vectorizable modeling language allowing for the direct processing of vector data. Use the following patterns for vector signals within a Simulink model:

Equivalent Functionality	Simulink Pattern
Vector loop  <pre>for (i=0; i&lt;input_vector_size; i++)</pre>	 <p>The diagram shows a Simulink Gain block. An input signal labeled 'input_vector' enters the block from the left. The block is a trapezoid that tapers from left to right, with the text 'tunable_parameter_value' inside. Below the block is the label 'Gain'. An output signal labeled 'output_vector' exits the block to the right.</p>

Equivalent Functionality	Simulink Pattern
<pre> { output_vector(i) = input_vector(i) * tunable_parameter_value; } </pre>	
<p>Vector loop</p> <pre> for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_vector(i); } </pre>	 <p>The diagram shows a Simulink Gain block. An arrow labeled 'input_vector' enters the block from the left. The block is a trapezoid with 'tunable_parameter_vector' written inside and 'Gain' written below it. An arrow labeled 'output_vector' exits the block to the right.</p>
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i&lt;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. The block is a square with a product symbol (Π) inside and 'Product' written below it. An arrow labeled 'output_signal' exits the block to the right.</p>
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i&lt;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. The block is a square with a product symbol (Π) inside and 'Product' written below it. An arrow labeled 'output_signal' exits the block to the right.</p>



Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre> for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_value; } </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_value' and is connected to a 'Constant' block. The output of the Sum block is labeled 'output_vector'.</p>
<p>Vector loop</p> <pre> for (i=0; i&lt;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' and is connected to a 'Constant' block. The output of the Sum block is labeled 'output_vector'.</p>
<p>Vector loop:</p> <pre> output_signal = 0; for (i=0; i&lt;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' and a plus sign inside the block. The output of the Sum block is labeled 'output_signal'.</p>
<p>Vector loop:</p> <pre> output_signal = 0; for (i=0; i&lt;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' and a minus sign inside the block. The output of the Sum block is labeled 'output_signal'.</p>

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

## Rationale

- Readability
- Verification and Validation
- Code Generation

## Last Changed

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## jc\_0351: Methods of initialization

### ID: Title

jc\_0351: Methods of initialization

### Priority

Recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

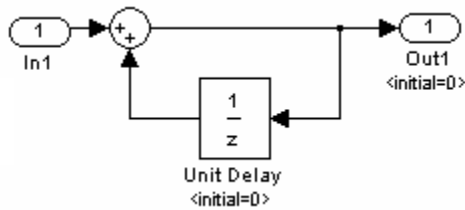
db\_0140: Display of basic block parameters

### Description

#### Simple Initialization

- Blocks such as Unit Delay, which have an initial value field, can be used to set simple initial values.

- To determine if the initial value needs to be displayed, see MAAB Guideline db\_0140: Display of basic block parameters.



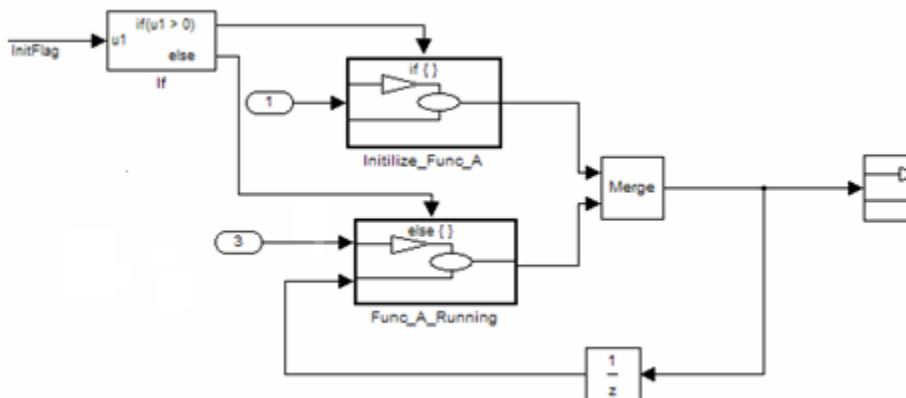
### Example

## Initialization that Requires Computation

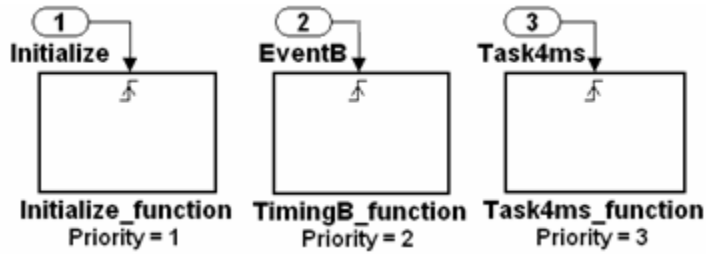
The following rules apply for complex initialization:

- 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.

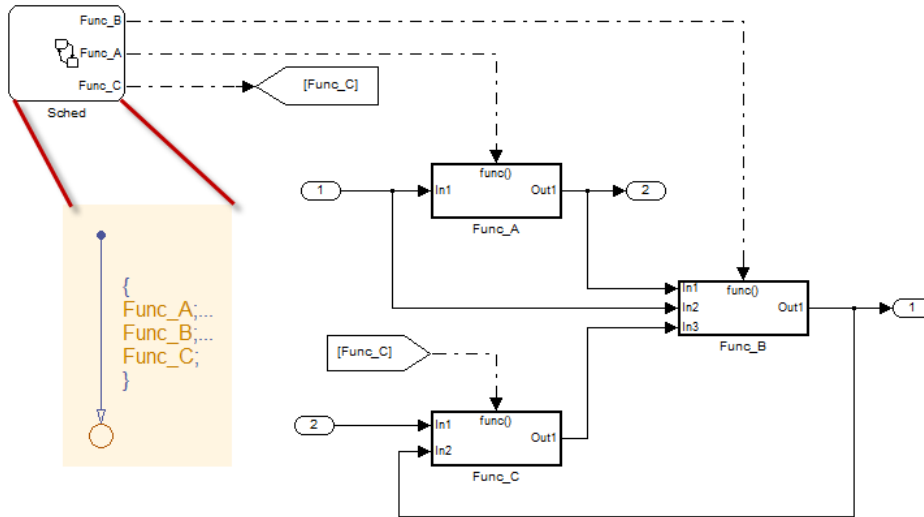


### Example A



### Example B

Or



## Rationale

- Readability
- Code Generation

## Last Changed

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## jc\_0111: Direction of Subsystem

### **ID: Title**

jc\_0111: Direction of Subsystem

### **Priority**

Strongly recommended

### **Scope**

J-MAAB

### **MATLAB Versions**

All

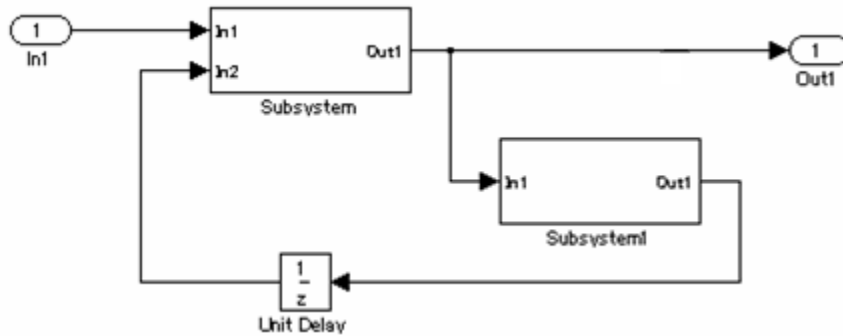
### **Prerequisites**

None

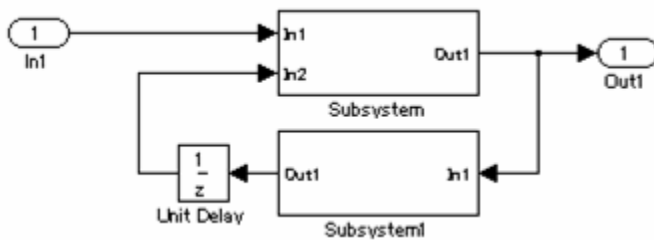
### **Description**

Subsystem must not be reversed.





**Correct**



**Incorrect**

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > Check for direction of subsystem blocks

For check details, see “Check orientation of Subsystem blocks”.

**Introduced in R2010a**

# Stateflow

---

- “Chart Appearance” on page 7-2
- “Stateflow Data and Operations” on page 7-28
- “Events” on page 7-57
- “State Chart Patterns” on page 7-64
- “Flow Chart Patterns” on page 7-72
- “State Chart Architecture” on page 7-91

## Chart Appearance

- db\_0123: Stateflow port names
- db\_0129: Stateflow transition appearance
- db\_0137: States in state machines
- db\_0133: Use of patterns for flow charts
- db\_0132: Transitions in flow charts
- jc\_0501: Format of entries in a State block
- jc\_0511: Setting the return value from a graphical function
- jc\_0531: Placement of the default transition
- jc\_0521: Use of the return value from graphical functions

# db\_0123: Stateflow port names

## ID: Title

db\_0123: Stateflow port names

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

The name of a Stateflow input or output should be the same as the corresponding signal.

**Exception:** Reusable Stateflow blocks may have different port names.

## Rationale

- Readability

- Code Generation

## Last Changed

V1.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for mismatches between Stateflow ports and associated signal names**

For check details, see “Check for mismatches between names of Stateflow ports and associated signals”.

**Introduced in R2010a**

# db\_0129: Stateflow transition appearance

## ID: Title

db\_0129: Stateflow transition appearance

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

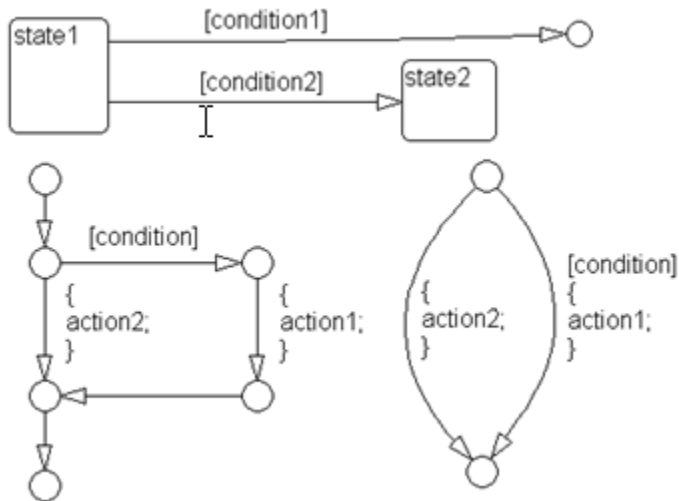
## Description

Transitions in Stateflow:

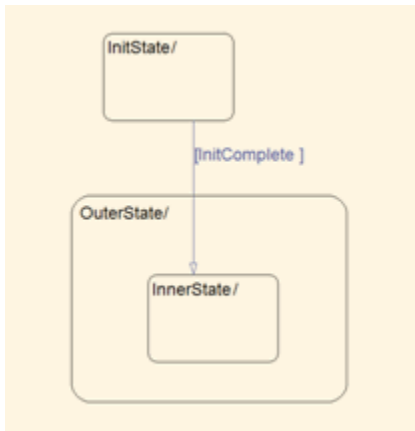
- Do not cross each other, if possible.
- Are not drawn one upon the other.
- Do not cross any states, junctions, or text fields.

- Allowed if transition is to an internal state.

Transition labels may be visually associated to the corresponding transition.

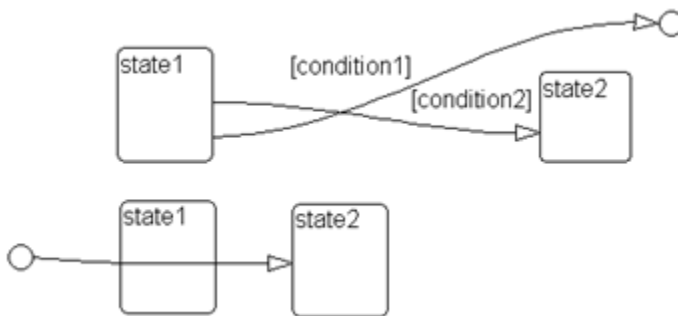


**Correct**



**Correct: Transition crosses state boundary to connect to substate**





**Incorrect:** Transitions cross each other and transition crosses through state

## Rationale

- Readability

## Last Changed

V2.2

## Model Advisor Check

Not applicable

Introduced in R2010a

## db\_0137: States in state machines

### ID: Title

db\_0137: States in state machines

### Priority

Mandatory

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

db\_0149: Flow chart patterns for condition actions

### Description

For all levels in a state machine, including the root level, for states with exclusive decomposition the following rules apply:

- At least two exclusive states must exist.
- A state cannot have only one substate.
- The initial state of every hierarchical level with exclusive states is clearly defined by a default transition. In the case of multiple default transitions, there must always be an unconditional default transition.

## Rationale

- Readability
- Workflow
- Code Generation
- Verification and Validation

## Last Changed

V3.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check usage of exclusive and default states in state machines**

For check details, see “Check usage of exclusive and default states in state machines”.

**Introduced in R2010a**

## db\_0133: Use of patterns for flow charts

### ID: Title

db\_0133: Use of patterns for flow charts

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

A flow chart is built with the help of flow chart patterns (for example, `if-then-else`, `for` loop, and so on):

- The data flow is oriented from the top to the bottom.
- Patterns are connected with empty transitions.

## **Rationale**

- Readability

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## db\_0132: Transitions in flow charts

### ID: Title

db\_0132: Transitions in flow charts

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

### Prerequisites

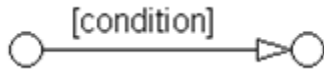
None

### Description

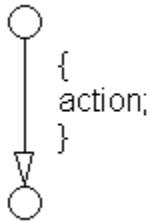
The following rules apply to transitions in flow charts:

- Conditions are drawn on the horizontal.
- Actions are drawn on the vertical.

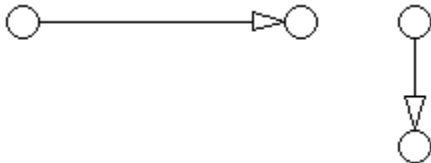
- Loop constructs are intentional exceptions to this rule.
- Transitions have a condition, a condition action, or an empty transition.



### Transition with Condition

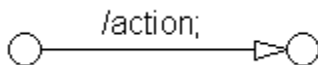


### Transition with Condition Action



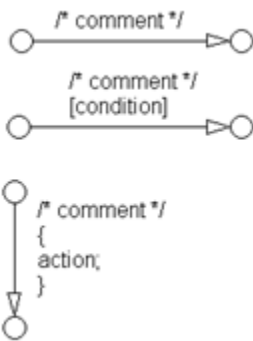
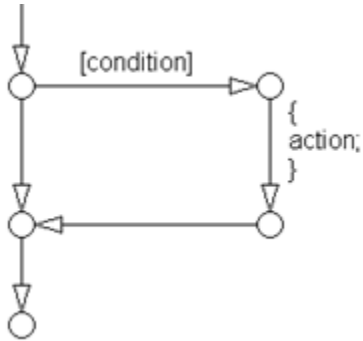
### Empty Transition

Transition actions are not used in flow charts. 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

**Last Changed**

V2.0



## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Stateflow > Check transition orientations in flow charts**

For check details, see “Check Transition orientations in flow charts”.

**Introduced in R2010a**

## 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

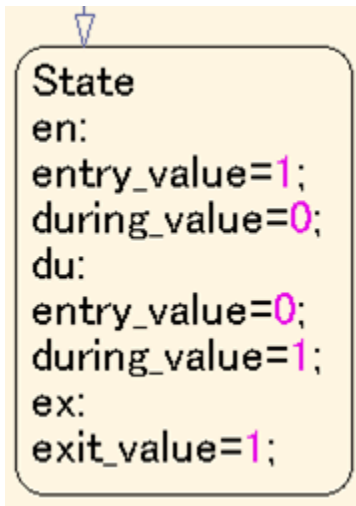
### Prerequisites

None

### Description

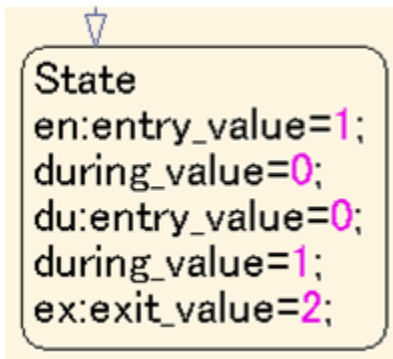
A new line should:

- Start after the entry (**en**), during (**du**), and exit (**ex**) statements.
- Start 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;
```

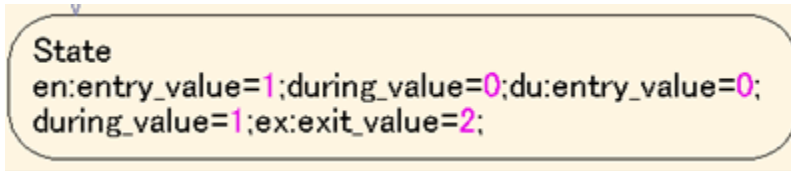
**Correct**



```
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.



## Incorrect

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

## Rationale

Readability

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for entry format in state blocks**

For check details, see “Check entry formatting in State blocks in Stateflow charts”.

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Mandatory

## Scope

J-MAAB

## MATLAB Versions

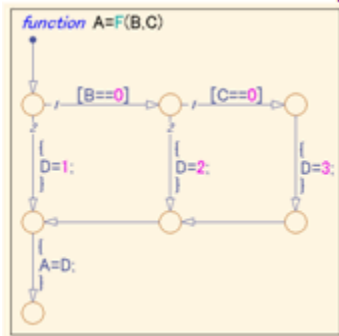
All

## Prerequisites

None

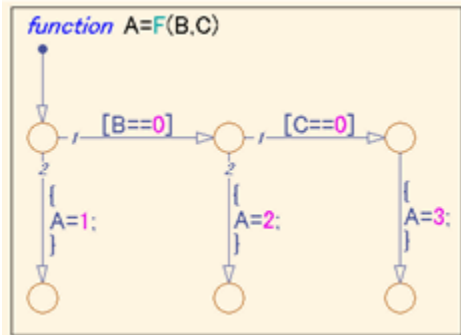
## Description

The return value from a graphical function must be set in only one place.



**Correct**

Return value A is set in one place.



**Incorrect**

Return value A is set in multiple places.

## Rationale

- Readability
- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check setting Stateflow graphical function return value**

For check details, see “Check return value assignments of graphical functions in Stateflow charts”.

**Introduced in R2010a**

## jc\_0531: Placement of the default transition

### ID: Title

jc\_0531: Placement of the default transition

### Priority

Recommended

### Scope

J-MAAB

### MATLAB Versions

All

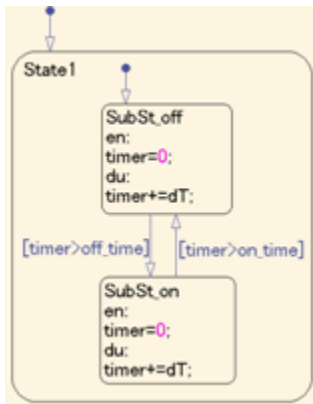
### Prerequisites

None

### Description

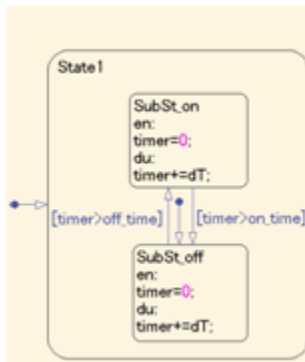
- Default transition is connected at the top of the state.
- The destination state of the default transition is put above the other states in the same hierarchy.





### 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.



### 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**

**By Task > Modeling Standards for MAAB > Stateflow > Check default transition placement in Stateflow charts**

For check details, see “Check default transition placement in Stateflow charts”.

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Recommended

## Scope

J-MAAB

## MATLAB Versions

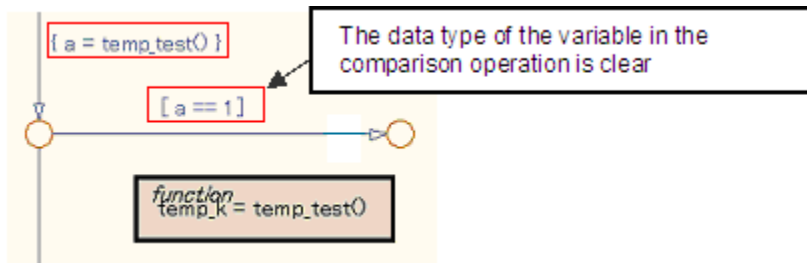
All

## Prerequisites

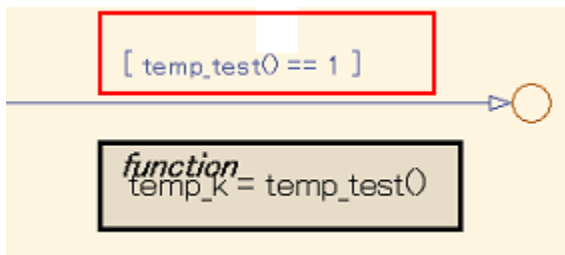
None

## Description

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`.

**Incorrect**

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

**Rationale**

- Readability
- Verification and Validation
- Code Generation

**Last Changed**

V2.0

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Stateflow > Check usage of return values from a graphical function in Stateflow charts**

For check details, see “Check usage of return values from a graphical function in Stateflow charts”.

**Introduced in R2010a**

## Stateflow Data and Operations

- na\_0001: Bitwise Stateflow operators
- jc\_0451: Use of unary minus on unsigned integers in Stateflow
- na\_0013: Comparison operation in Stateflow
- db\_0122: Stateflow and Simulink interface signals and parameters
- db\_0125: Scope of internal signals and local auxiliary variables
- jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow
- jc\_0491: Reuse of variables within a single Stateflow scope
- jc\_0541: Use of tunable parameters in Stateflow
- db\_0127: MATLAB commands in Stateflow
- jm\_0011: Pointers in Stateflow

# na\_0001: Bitwise Stateflow operators

## ID: Title

na\_0001: Bitwise Stateflow operators

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

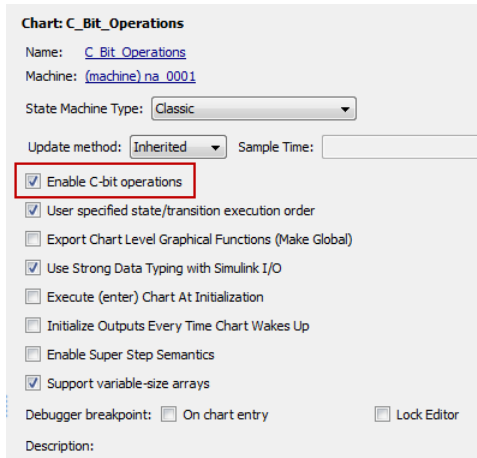
None

## Description

The bitwise Stateflow operators (&, |, and ^) should not be used in Stateflow charts unless you want bitwise operations:

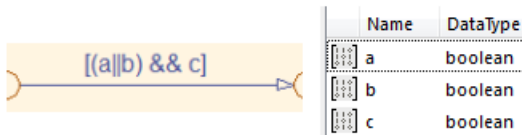
To enable bitwise operations,

- 1 Select **File > Chart Properties**.
- 2 Select **Enable C-bit operations**.

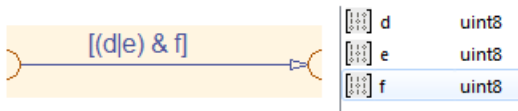


## Correct

Use `&&` and `||` for Boolean operation.



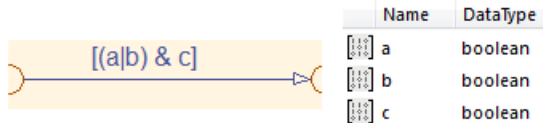
Use `&` and `|` for bit operation.





## Incorrect

Use & and | for Boolean operation.



## Rationale

- Readability
- Verification and Validation
- Code Generation

## Last Changed

V2.2

## Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > Check for bitwise operations in Stateflow charts

For check details, see “Check for bitwise operations in Stateflow charts”.

Introduced in R2010a

# 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

- Verification and Validation
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for unary minus operations on unsigned integers in Stateflow charts**

For check details, see “Check for unary minus operations on unsigned integers in Stateflow charts”.

**Introduced in R2010a**

## na\_0013: Comparison operation in Stateflow

### ID: Title

na\_0013: Comparison operation in Stateflow

### Priority

Recommended

### Scope

MAAB

### MATLAB Versions

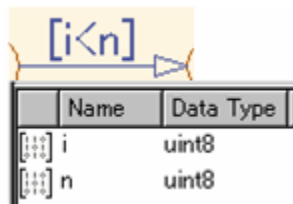
All

### Prerequisites

None

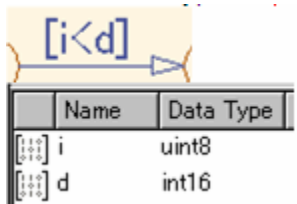
### Description

- 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.



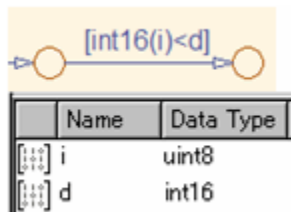
**Correct**

Same data type in “i” and “n”



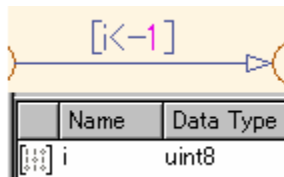
**Incorrect**

Different data type in “i” and “d”



**Correct**

Do not make comparisons between unsigned integers and negative numbers.



**Incorrect**

## **Rationale**

- Verification and Validation
- Code Generation
- Simulation

## **Last Changed**

V2.1

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Stateflow > Check for comparison operations in Stateflow charts**

For check details, see “Check for comparison operations in Stateflow charts”.

**Introduced in R2010a**

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

## ID: Title

db\_0122: Stateflow and Simulink interface signals and parameters

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

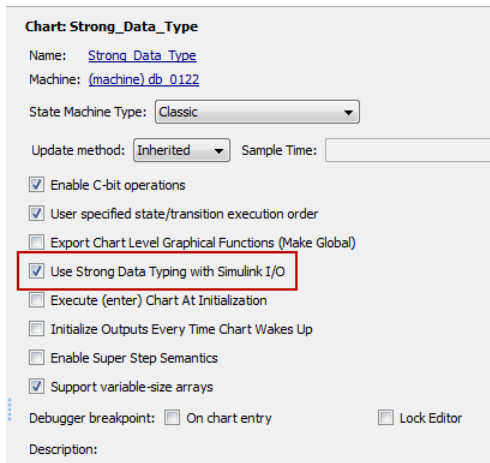
All

## Prerequisites

None

## Description

A Chart uses strong data typing with Simulink and requires that you select the **Use Strong Data Typing with Simulink I/O** parameter.



## Rationale

- Verification and Validation
- Code Generation
- Simulation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check interface signals and parameters**

For check details, see “Check for Strong Data Typing with Simulink I/O”.

**Introduced in R2010a**



# 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

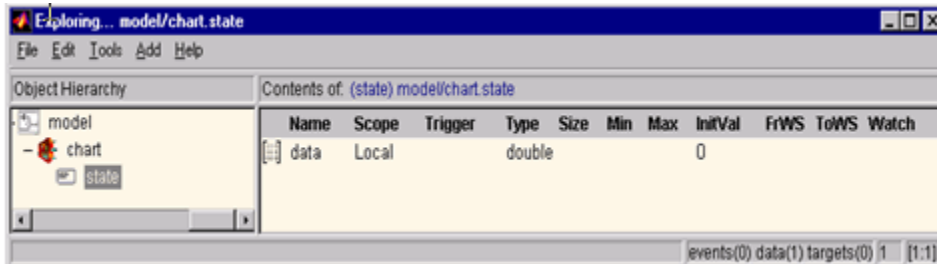
None

## Description

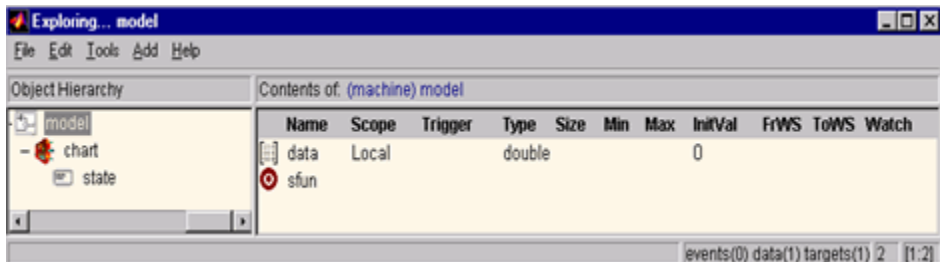
Internal signals and local auxiliary variables are "Local data" in Stateflow:

- 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



### Incorrect

## Rationale

- Readability
- Code Generation

## Last Changed

V2.0

## Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > Check Stateflow data objects with local scope

For check details, see “Check Stateflow data objects with local scope”.

**Introduced in R2010a**

# 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

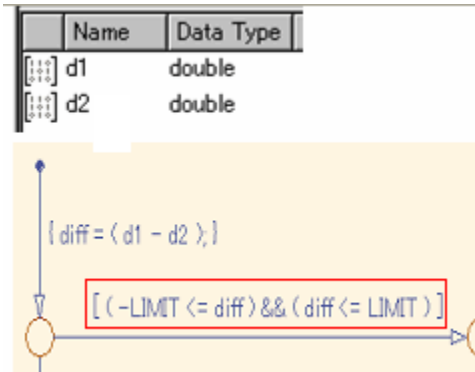
All

## Prerequisites

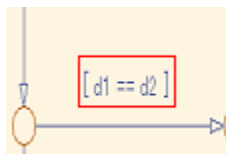
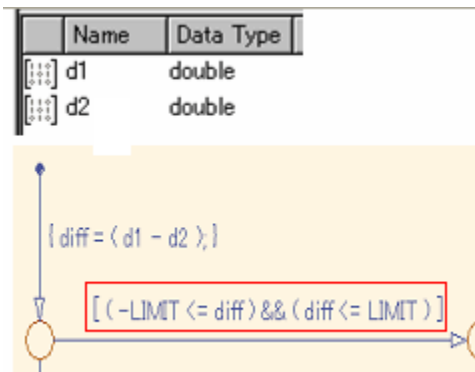
None

## 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**



**Incorrect**

## Rationale

- Verification and Validation

- Code Generation

## Last Changed

V2.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for equality operations between floating-point expressions in Stateflow charts**

For check details, see “Check for equality operations between floating-point expressions in Stateflow charts”.

**Introduced in R2010a**

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

## ID: Title

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

## Priority

Recommended

## Scope

MAAB

## MATLAB Versions

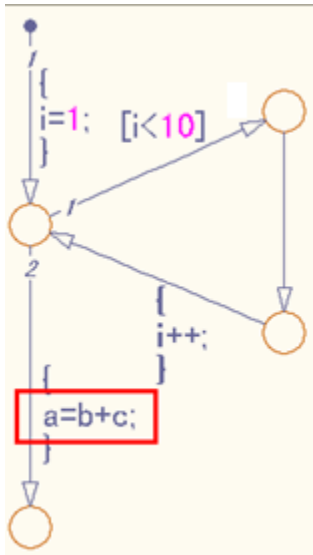
All

## Prerequisites

None

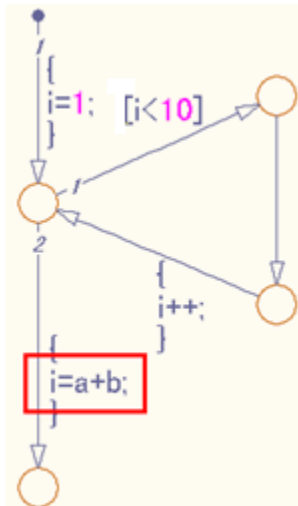
## Description

The same variable should not have multiple meanings (usages) within a single Stateflow state.



**Correct**

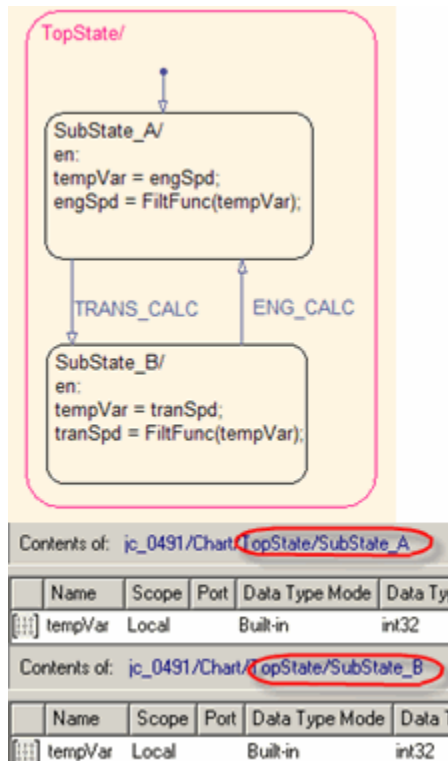
Variable of loop counter must not be used other than loop counter.



**Incorrect**



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



### Correct

`tempVar` is defined as local scope in both `SubState_A` and `SubState_B`.

## Rationale

- Readability
- Verification
- Code Generation

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

# 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

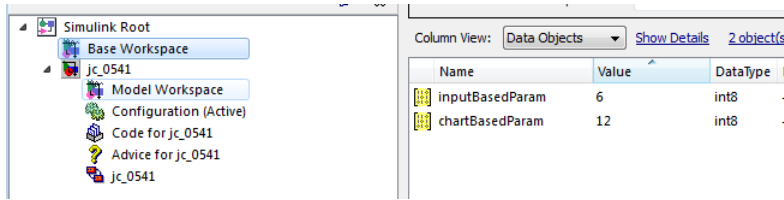
## Prerequisites

None

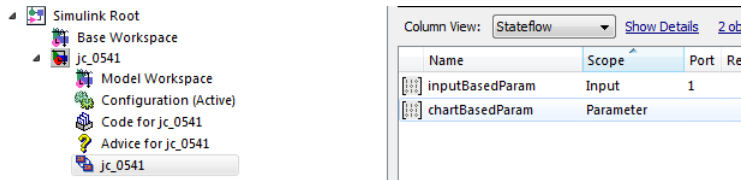
## Description

Create tunable parameters in Stateflow charts in one of the following ways:

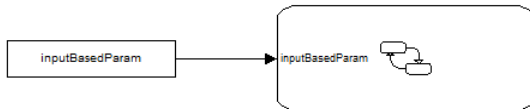
- Define the parameters in the Stateflow chart and corresponding parameters in the base workspace.
- Include the tunable parameters an input into the Stateflow chart. The parameters must be defined in the base workspace.



### Base Workspace Definitions



### Stateflow Chart Definitions



### Stateflow Chart

## Rationale

- Verification
- Code Generation

## Last Changed

V2.2

## Model Advisor Check

Not applicable

**Introduced in R2010a**

## db\_0127: MATLAB commands in Stateflow

### ID: Title

db\_0127: MATLAB commands in Stateflow

### Priority

Mandatory

### Scope

MAAB

### MATLAB Versions

All

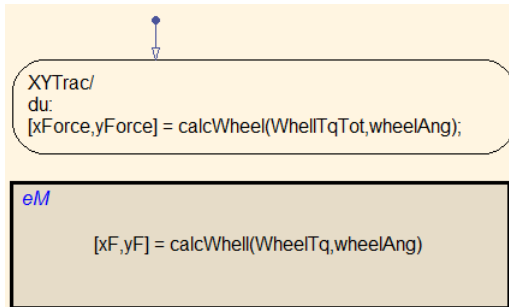
### Prerequisites

None

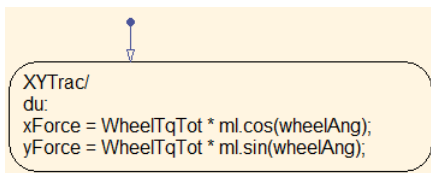
### Description

In Stateflow charts, do not use the .ml syntax.

Individual companies should decide on the use of MATLAB functions. If they are permitted, then MATLAB functions should only be accessed through the MATLAB function block.



**Correct**



**Incorrect**

## Rationale

- Verification and Validation
- Code Generation
- Simulation

---

**Note:** Code generation supports a limited subset of the MATLAB functions. For a complete list of the supported function, see the MathWorks documentation.

---

## Last Changed

V2.2

## **Model Advisor Check**

**By Task > Modeling Standards for MAAB > Stateflow > Check for MATLAB expressions in Stateflow charts**

For check details, see “Check for MATLAB expressions in Stateflow charts”.

**Introduced in R2010a**



# jm\_0011: Pointers in Stateflow

## ID: Title

jm\_0011: Pointers in Stateflow

## Priority

Strongly recommended

## Scope

MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

In a Stateflow diagram, pointers to custom code variables are not allowed.

## Rationale

- Readability

- Verification and Validation
- Code Generation

## Last Changed

V1.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for pointers in Stateflow charts**

For check details, see “Check for pointers in Stateflow charts”.

**Introduced in R2010a**

## Events

- db\_0126: Scope of events
- jm\_0012: Event broadcasts

## db\_0126: Scope of events

### ID: Title

db\_0126: Scope of events

### Priority

Mandatory

### Scope

MAAB

### MATLAB Versions

Pre R2009b

### Prerequisites

None

### Description

The following rules apply to events in Stateflow:

- All events of a Chart must be defined on the chart level or lower.
- There is no event on the machine level (i.e. there is no interaction with local events between different charts).

### Specifics

## **Rationale**

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Verification and Validation

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## jm\_0012: Event broadcasts

### ID: Title

jm\_0012: Event broadcasts

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

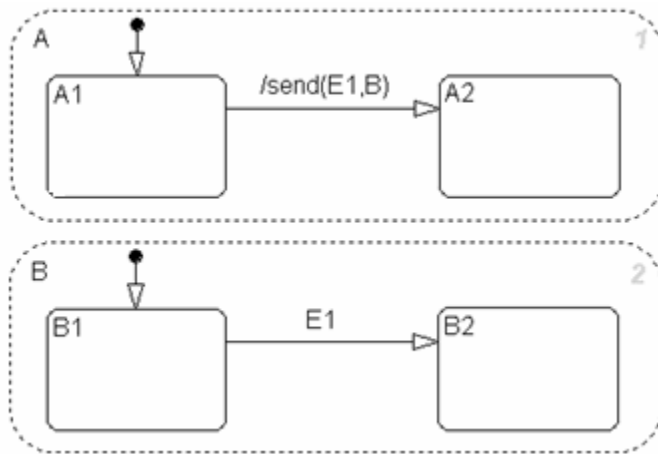
### Prerequisites

db\_0126: Scope of events

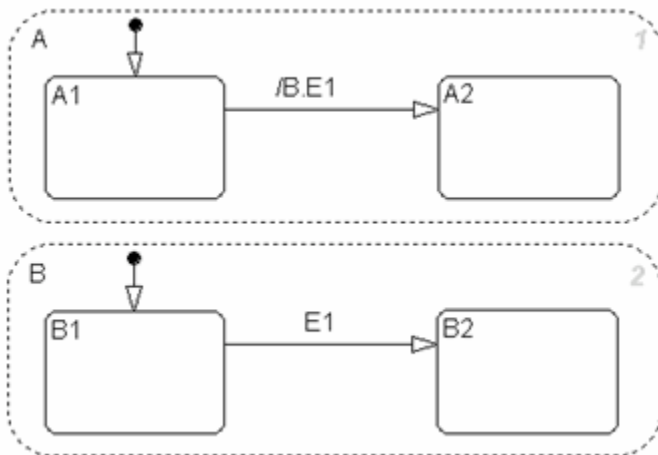
### Description

The following rules apply to event broadcasts in Stateflow:

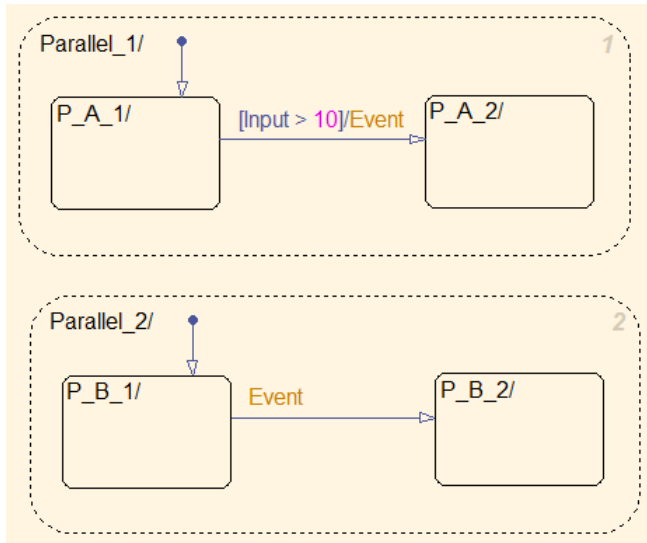
- Directed event broadcasts are the only type of event broadcasts allowed.
- The send syntax or qualified event names are used to direct the event to a particular state.
- Multiple send statements should be used to direct an event to more than one state.



**Correct: Example Using Send Syntax**



**Correct: Example Using Qualified Event Names**



**Incorrect: Use of a non-directed event**

## Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation
- Simulation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check for event broadcasts in Stateflow charts**



For check details, see “Check for event broadcasts in Stateflow charts”.

**Introduced in R2010a**

## State Chart 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

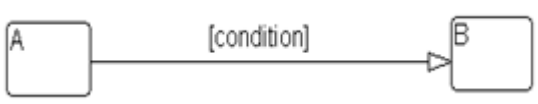
All

## Prerequisites

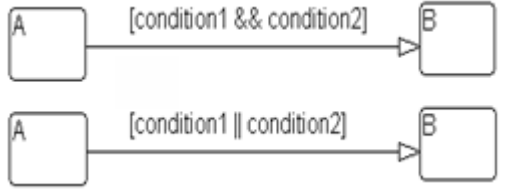
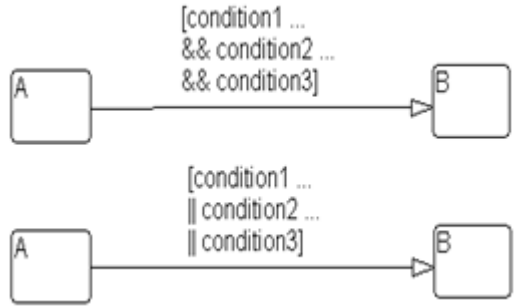
None

## Description

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

Equivalent Functionality	State Machine Pattern
One condition: (condition)	



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

## Last Changed

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

# 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

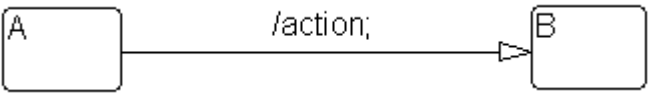
All

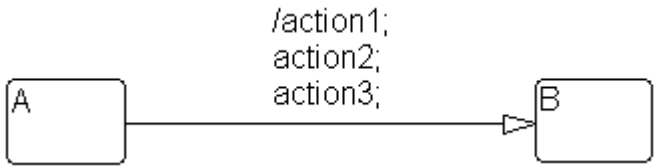
## Prerequisites

None

## Description

The following patterns are used for transition actions within Stateflow state machines:

Equivalent Functionality	State Machine Pattern
One transition action: <pre>action;</pre>	 <p>The diagram shows two state boxes, A and B. A horizontal arrow points from box A to box B. Above the arrow is the text '/action;'. The arrow ends in a triangular arrowhead pointing into box B.</p>

Equivalent Functionality	State Machine Pattern
<p>Two or more transition actions, multiline form:</p> <p>(Two or more transition actions in one line are not allowed.)</p> <pre>action1; action2; action3;</pre>	 <p>The diagram shows a state machine pattern with two states, A and B, represented by rounded rectangles. A transition arrow points from state A to state B. The transition is labeled with the following actions: /action1; action2; action3;.</p>



## Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation
- Simulation

## Last Changed

V2.2

## Model Advisor Check

**By Task > Modeling Standards for MAAB > Stateflow > Check transition actions in Stateflow charts**

For check details, see “Check transition actions in Stateflow charts”.

**Introduced in R2010a**

## Flow Chart Patterns

- db\_0148: Flow chart patterns for conditions
- db\_0149: Flow chart patterns for condition actions
- db\_0134: Flow chart patterns for If constructs
- db\_0159: Flow chart patterns for case constructs
- db\_0135: Flow chart patterns for loop 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: Flow chart patterns for conditions

## **ID: Title**

db\_0148: Flow chart patterns for conditions

## **Priority**

Strongly recommended

## **Scope**

MAAB

## **MATLAB Versions**

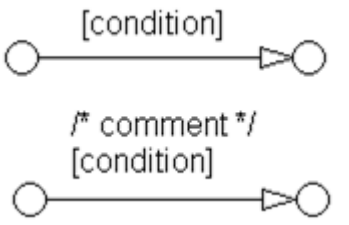
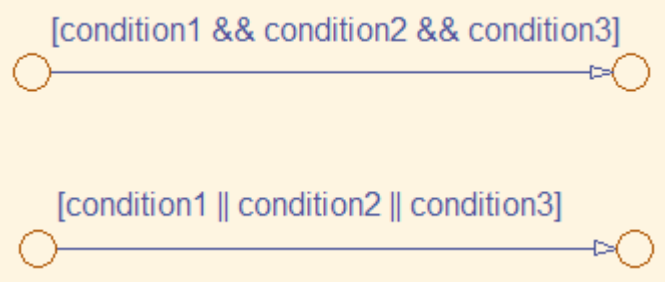
All

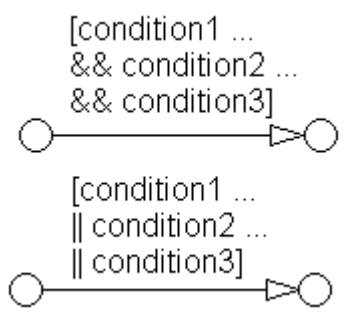
## **Prerequisites**

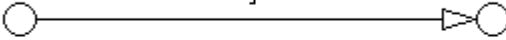

None

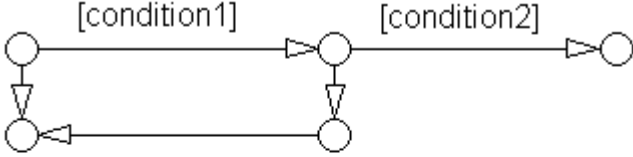
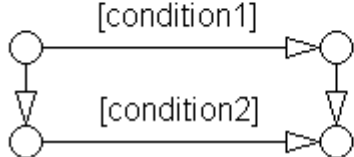
## **Description**

Use the following patterns for conditions within Stateflow flow charts:

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

Equivalent Functionality	Flow Chart Pattern
<p>Two or more conditions, multiline form: (The use of different logical operators in this form is not allowed. Use subconditions instead.)</p> <p>[condition1 ... &amp;&amp; condition2 ... &amp;&amp; condition3] [condition1 ...    condition2 ...    condition3]</p>	

Equivalent Functionality	Flow Chart Pattern
<p>Conditions with subconditions:</p> <p>(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>	<pre> [(condition1a    condition1b) ...  &amp;&amp; (condition2a    condition2b) ...  &amp;&amp; condition3] </pre>  <pre> [(condition1a &amp;&amp; condition1b) ...     (condition2a &amp;&amp; condition2b) ...     condition3] </pre> 

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

## Rationale

- Readability

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

# db\_0149: Flow chart patterns for condition actions

## **ID: Title**

db\_0149: Flow chart patterns for condition actions

## **Priority**

Strongly recommended

## **Scope**

MAAB

## **MATLAB Versions**

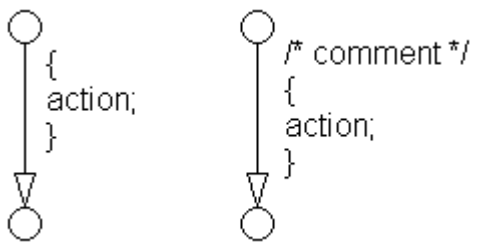
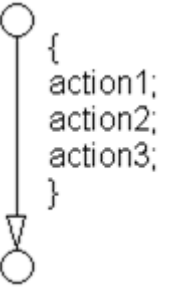
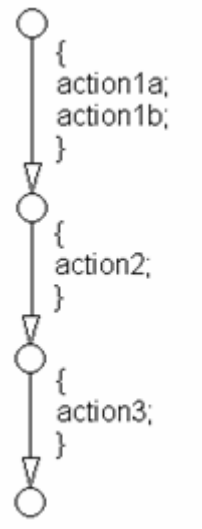
All

## **Prerequisites**

None

## **Description**

You should use the following patterns for condition actions within Stateflow flow charts:

Equivalent Functionality	Flow Chart Pattern
<p>One condition action:</p> <pre>action;</pre>	
<p>Two or more condition actions, multiline form: (Two or more condition actions in one line are not allowed.)</p> <pre>action1; ... action2; ... action3; ...</pre>	
<p>Condition actions, that are visually separated: (This form may be combined with the preceding patterns.)</p> <pre>action1a; action1b; action2; action3;</pre>	



## **Rationale**

- Readability

## **Last Changed**

V2.2

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## db\_0134: Flow chart patterns for If constructs

### ID: Title

db\_0134: Flow chart patterns for If constructs

### Priority

Strongly recommended

### Scope

MAAB

### MATLAB Versions

All

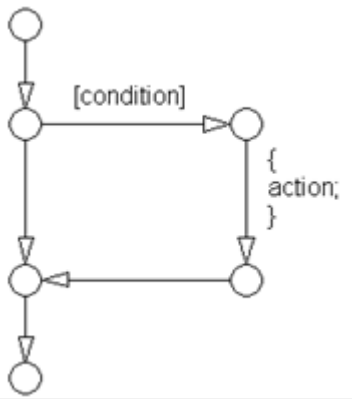
### Prerequisites

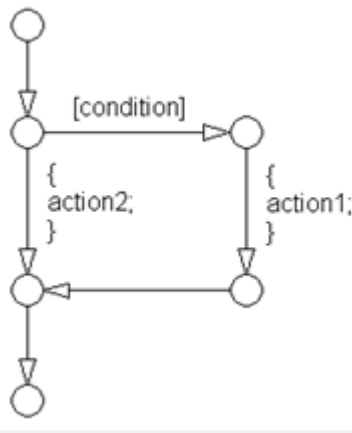
db\_0148: Flow chart patterns for conditions

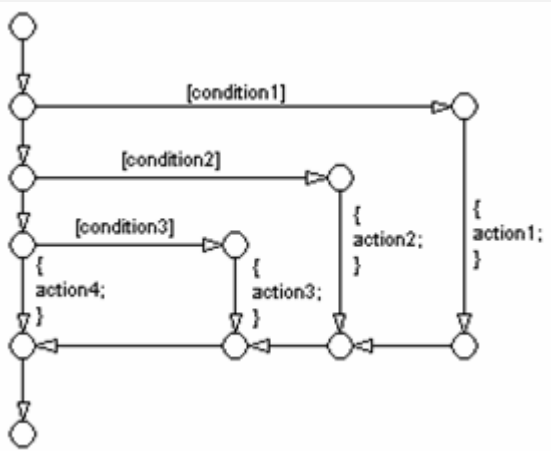
db\_0149: Flow chart patterns for condition actions

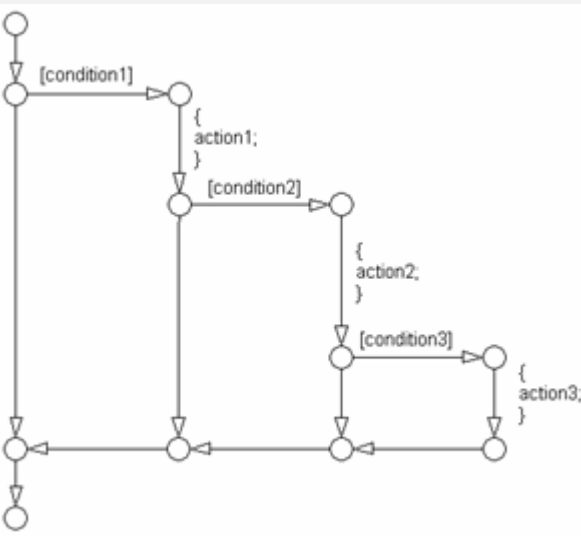
### Description

Use the following patterns for If constructs within Stateflow flow charts:

Equivalent Functionality	Flow Chart Pattern
<pre> if then if (condition){ action; } </pre>	 <p>The flowchart illustrates the execution of an 'if then' statement. It begins with a start node (circle) leading to a decision diamond (inverted triangle). From the diamond, a horizontal arrow labeled '[condition]' points to a right-hand node. From this node, a vertical arrow labeled '{ action; }' points down to another node. From this second node, a horizontal arrow points left to a node below the decision diamond. Finally, a vertical arrow points down from this node to the end node (circle).</p>

Equivalent Functionality	Flow Chart Pattern
<pre> if then else if (condition){ action1; } else { action2; } </pre>	 <p>The flowchart illustrates the execution of an 'if then else' statement. It begins with a start node (circle) leading to a decision diamond (inverted triangle). From the diamond, a horizontal arrow labeled '[condition]' points to a right-hand node. From this node, a vertical arrow labeled '{ action1; }' points down to another node. From this second node, a horizontal arrow points left to a node below the decision diamond. From this node, a vertical arrow labeled '{ action2; }' points down to another node. Finally, a vertical arrow points down from this node to the end node (circle).</p>

Equivalent Functionality	Flow Chart Pattern
<pre> if then else if if (condition1){  action1; } else if (condition2) {  action2; } else if (condition3){   _action3; } else {   action4; }                     </pre>	 <p>The flow chart pattern for 'if then else if' consists of a vertical sequence of five states. The top state is the start state. Transitions from the top state to the second, third, and fourth states are labeled with conditions [condition1], [condition2], and [condition3] respectively. From the second state, a transition labeled { action1; } leads to the top state. From the third state, a transition labeled { action2; } leads to the second state. From the fourth state, a transition labeled { action3; } leads to the third state. From the fifth state, a transition labeled { action4; } leads to the fourth state. A final transition leads from the fifth state to an end state.</p>

Equivalent Functionality	Flow Chart Pattern
<pre> Cascade of if then if (condition1){  action1;   if (condition2){  action2;     if (condition3){  action3;     }   } }                     </pre>	 <p>The flow chart pattern for 'Cascade of if then' consists of a vertical sequence of five states. The top state is the start state. A transition labeled [condition1] leads from the top state to the second state, with action { action1; } occurring on the transition. From the second state, a transition labeled [condition2] leads to the third state, with action { action2; } occurring on the transition. From the third state, a transition labeled [condition3] leads to the fourth state, with action { action3; } occurring on the transition. From the fourth state, a transition leads to the fifth state. From the second, third, and fourth states, transitions lead back to the first state. A final transition leads from the fifth state to an end state.</p>

## **Rationale**

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Simulation

## **Last Changed**

V1.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## **db\_0159: Flow chart patterns for case constructs**

### **ID: Title**

db\_0159: Flow chart patterns for case constructs

### **Priority**

Strongly recommended

### **Scope**

MAAB

### **MATLAB Versions**

All

### **Prerequisites**

db\_0148: Flow chart patterns for conditions

db\_0149: Flow chart patterns for condition actions

## Description

Use the following patterns must be used for case constructs within Stateflow flow charts:

Equivalent Functionality	Flow Chart Pattern
<pre> case with exclusive selection  selection = ...; switch (selection) { case 1:   action1; break; case 2:   action2; break; case 3:   action3; break; default:   action4; } </pre>	

Equivalent Functionality	Flow Chart Pattern
<pre> case with exclusive conditions  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; S1(( ))     S1 -- "[c1 &amp;&amp; !c2 &amp;&amp; !c3]" --&gt; A1["{action1;}"]     A1 --&gt; S2(( ))     S2 -- "[!c1 &amp;&amp; c2 &amp;&amp; !c3]" --&gt; A2["{action2;}"]     A2 --&gt; S3(( ))     S3 -- "[!c1 &amp;&amp; !c2 &amp;&amp; c3]" --&gt; A3["{action3;}"]     A3 --&gt; S4(( ))     S4 -- "{action4;}" --&gt; S5(( ))     S5 --&gt; End(( )) </pre>

## Rationale

- Readability

## Last Changed

V1.0

## Model Advisor Check

Not applicable



**Introduced in R2010a**

## db\_0135: Flow chart patterns for loop constructs

### **ID: Title**

db\_0135: Flow chart patterns for loop constructs

### **Priority**

Recommended

### **Scope**

MAAB

### **MATLAB Versions**

All

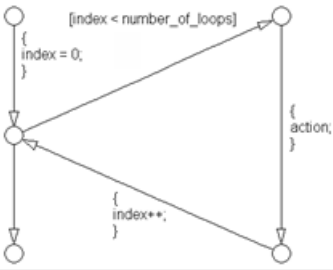
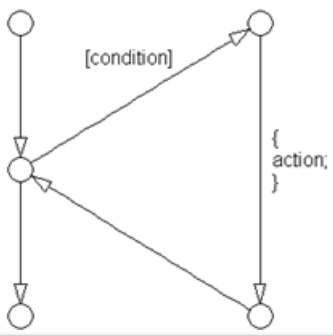
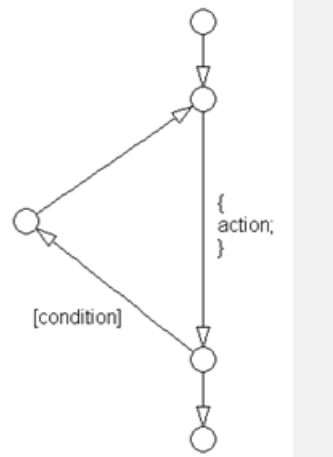
### **Prerequisites**

db\_0148: Flow chart patterns for conditions

db\_0149: Flow chart patterns for condition actions

### **Description**

Use the following patterns to create Loops within Stateflow flow charts:

Equivalent Functionality	Flow Chart Pattern
<pre> for loop for (index=0; index&lt;number_of_loops; index++) { action; } </pre>	
<pre> while loop while (condition) { action; } </pre>	
<pre> do while loop do { action; } while (condition); </pre>	

## **Rationale**

- Readability

## **Last Changed**

V1.0

## **Model Advisor Check**

Not applicable

**Introduced in R2010a**

## State Chart Architecture

- na\_0038: Levels in Stateflow charts
- na\_0039: Use of Simulink in Stateflow charts
- na\_0040: Number of states per container
- na\_0041: Selection of function type
- na\_0042: Location of Simulink functions

## na\_0038: Levels in Stateflow charts

### ID: Title

na\_0038: Levels in Stateflow charts

### Priority

Recommended

### Scope

NA-MAAB

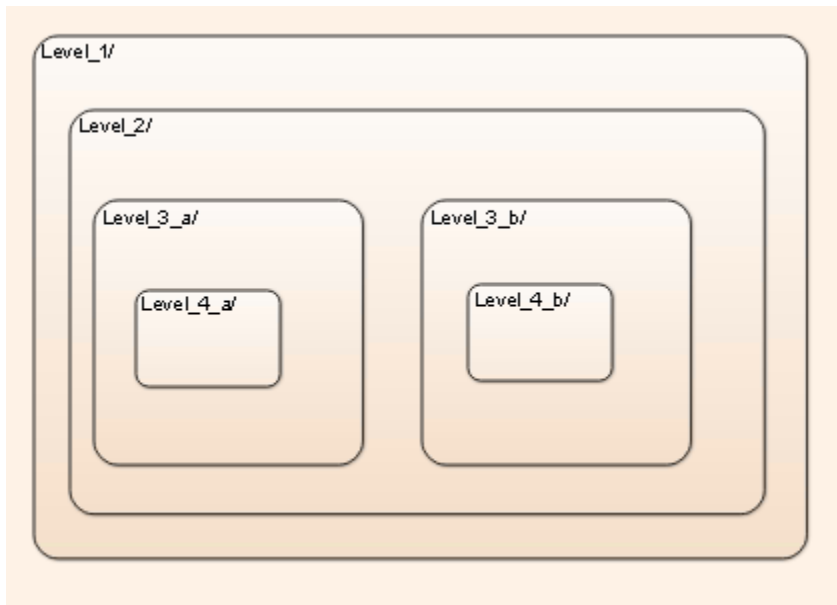
### MATLAB Versions

All

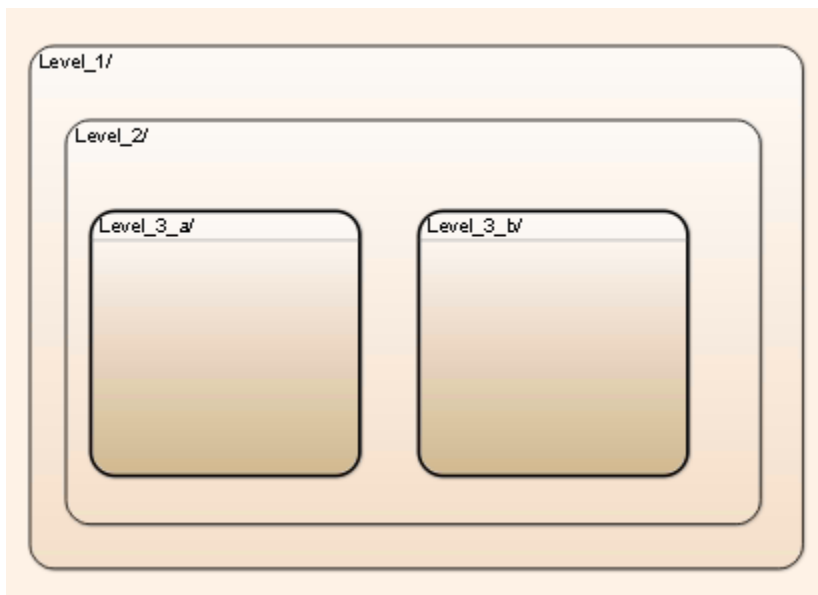
### Description

The number of nested States should be limited, typically 3 per level. If additional levels are required, use sub-charts.

**Incorrect:** Level\_4\_a and Level\_4\_b are nested more than 3 deep



**Correct:** The 4 levels are encapsulated inside a subchart



## **Rationale**

- Readability

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**



# na\_0039: Use of Simulink in Stateflow charts

## ID: Title

na\_0039: Use of Simulink in Stateflow charts

## Priority

Recommended

## Scope

NA-MAAB

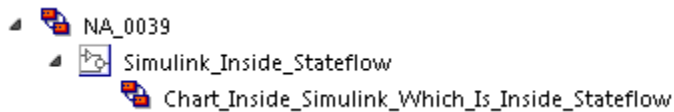
## MATLAB Versions

R2010b and later

## Description

Do not nest Stateflow charts inside Simulink functions that are included in Stateflow charts.

### Incorrect



## Rationale

- Readability

- Verification and Validation
- Code Generation

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**

## **na\_0040: Number of states per container**

### **ID: Title**

na\_0040: Number of states per container

### **Priority**

Recommended

### **Scope**

NA-MAAB

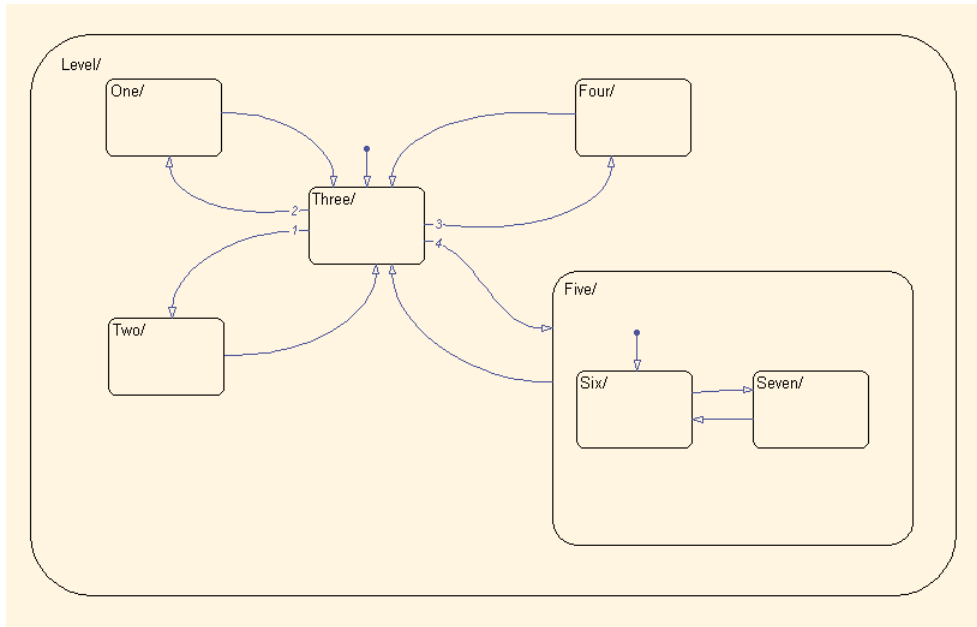
### **MATLAB Versions**

All

### **Description**

The number of viewable States per container should be limited, typically to 6 to 10 states per container. The number is based on the visible states in the diagram.

**Correct**



## Note

A container is either a State, Box or root level chart.

## Rationale

- Readability
- Verification and Validation
- Code Generation

## Last Changed

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**

## na\_0041: Selection of function type

### ID: Title

na\_0041: Selection of function type

### Priority

Recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Description

Stateflow supports three types of functions: Graphical, MATLAB and Simulink. The appropriate function depends on the type of operations required:

- Simulink
  - Transfer functions
  - Integrators
  - Table look-ups
- MATLAB
  - Complex equations
  - If / then / else logic

- Graphical functions
  - If / then / else logic

## **Rationale**

- Workflow
- Code Generation

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**

## na\_0042: Location of Simulink functions

### ID: Title

na\_0042: Location of Simulink functions

### Priority

Recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

na\_0039: Use of Simulink in Stateflow charts

### Description

When deciding whether to embed Simulink functions inside a Stateflow chart, the following conditions make embedding the preferred option. If the Simulink functions

- Use only local Chart data.

OR

- Use a mixture of local Chart data and inputs from Simulink. OR



OR

- Are called from multiple locations within the chart.

OR

- Are not called every time step.

## **Rationale**

- Readability
- Workflow

## **Last Changed**

V3.0

## **Model Advisor Check**

Not applicable

**Introduced in R2013a**



# Enumerated Data

---

## **General Guidelines**

- na\_0033: Enumerated Types Usage
- na\_0031: Definition of default enumerated value

# na\_0033: Enumerated Types Usage

## ID: Title

na\_0033: Enumerated Types Usage

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

R2010b and later

## Prerequisites

None

## Description

An enumerated data type should be used when a signal or parameter can take on a finite set of integer values, and those values are associated with a set of named items. The names, called literals, have meaning in the context of the algorithm or the domain in which it operates. Typically, these literals represent an operating mode, signal status, build variation, or some other discrete property that the quantity represented by the variable can take on. A typical automotive example of this is the modes of a transmission: Park, Reverse Neutral, Drive, Low.

## **Rationale**

- Readability
- Verification and Validation
- Workflow
- Code Generation
- Simulation

## **See Also**

- NASA Orion style guideline dm\_0002: Enumerated type usage

## **Last Changed**

V3.0

**Introduced in R2013a**

# na\_0031: Definition of default enumerated value

## ID: Title

na\_0031: Definition of default enumerated value

## Priority

Recommended

## Scope

NA-MAAB

## MATLAB Versions

R2010b and later

## Prerequisites

None

## Description

The default value of the enumeration should always be explicitly defined for the enumerated type.

## Rationale

- Readability

- Verification and Validation
- Code Generation

## **Last Changed**

V3.0

**Introduced in R2013a**



# MATLAB Functions

---

- “MATLAB Function Appearance” on page 9-2
- “MATLAB Function Data and Operations” on page 9-9
- “MATLAB Function Patterns” on page 9-15
- “MATLAB Function Usage” on page 9-19

## MATLAB Function Appearance

- na\_0018: Number of nested if/else and case statement
- na\_0019: Restricted Variable Names
- na\_0025: MATLAB Function Header

# na\_0018: Number of nested if/else and case statement

## ID: Title

na\_0018: Number of nested if/else and case statement

## Priority

Strongly recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

The number of levels of nested if / else and case statements should be limited, typically to 3 levels.

## Rationale

- Readability

- Code Generation

## See Also

- NASA Orion style guideline jr\_0002: Number of nested if/else and case statement blocks

## Last Changed

V3.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > MATLAB Functions > Check MATLAB Function block metrics**

For check details, see “Check MATLAB Function metrics”.

**Introduced in R2013a**

# na\_0019: Restricted Variable Names

## ID: Title

na\_0019: Restricted Variable Names

## Priority

Mandatory

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

To improve the readability of the MATLAB code, avoid using reserved C variable names. For example, avoid using `const`, `TRUE`, `FALSE`, `infinity`, `nil`, `double`, `single`, or `enum`.

Avoid using variable names that conflict with MATLAB Functions, for example `conv`.

## Note

Reserved keywords are defined in the Simulink Coder™ documentation.

## Rationale

- Readability
- Verification and Validation

## See Also

- Derived from NASA Orion style guideline jh\_0042: Required software

## Last Changed

V3.0

**Introduced in R2013a**

# na\_0025: MATLAB Function Header

## ID: Title

na\_0025: MATLAB Function Header

## Priority

Strongly recommended

## Scope

NA-MAAB

## MATLAB Versions

All

## Prerequisites

None

## Description

MATLAB Functions must have a descriptive header. Header content may include, but is not limited to, the following types of information:

- Function name
- Description of function
- Assumptions and limitations

- Description of changes from previous versions
- Lists of inputs and outputs

**Example:**

```
%% Function Name: NA_0025_Example_Header
%
% Assumptions: None
%
% Inputs:
%   List of input arguments
%
% Outputs:
%   List of output arguments

%
% $Date: August 27, 2012
%
```

---

## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

## See Also

- NASA Orion style guideline jh\_0073: eML Header

## Last Changed

V3.0

**Introduced in R2013a**



## **MATLAB Function Data and Operations**

- na\_0034: MATLAB Function block input/output settings
- na\_0024: Global Variables

## na\_0034: MATLAB Function block input/output settings

### ID: Title

na\_0034:MATLAB Function block input/output settings

### Priority

Strongly recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

All inputs and outputs to MATLAB function blocks should have the data type explicitly defined, either in the Model Explorer or at the start of the function. This provides a more rigorous data type check for MATLAB Function blocks and prevents the need for using `assert` statements.

## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

## Last Changed

V3.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > MATLAB Functions > Check for fully defined interface at MATLAB Function block boundary**

For check details, see “Check input and output settings of MATLAB Functions”.

**Introduced in R2013a**

## na\_0024: Global Variables

### ID: Title

na\_0024: Global Variables

### Priority

Strongly recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

The preferred method for accessing common data is by signal lines. However, if required, Data Store Memory can be used to emulate global memory.

#### Example:

In this example, the same Data Store Memory (`ErrorFlag_DataStore`) is written to two separate MATLAB Functions.

```
function EngineFaultEvaluation(EngineData)
    %# codegen
    global ErrorFlag_DataStore
    if (EngineData.RPM_HIGH)
        ErrorFlag_DataStore = bitor(ErrorFlag_DataStore, HIGHRPMFAULT);
    end

    if (EngineData.RPM_LOW)
        ErrorFlag_DataStore = bitor(ErrorFlag_DataStore, LOWRPMFAULT);
    end
end

function WheelFaultEvaluation(WheelData)
    %# codegen
    global ErrorFlag_DataStore
    if (WheelData.SlipHigh)
        ErrorFlag_DataStore = bitor(ErrorFlag_DataStore, WHEELSLIP);
    end

    if (WheelData.SlipHigh)
        ErrorFlag_DataStore = bitor(ErrorFlag_DataStore, LOWRPMFAULT);
    end
end
```

## Rationale

- Readability
- Verification and Validation
- Code Generation
- Simulation

## See Also

- NASA Orion style guideline ek\_0003: Global Variables

## Last Changed

V3.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > MATLAB Functions > Check MATLAB code for global variables**

For check details, see “Check MATLAB code for global variables”.

**Introduced in R2013a**

## **MATLAB Function Patterns**

- na\_0022: Recommended patterns for Switch/Case statements

## na\_0022: Recommended patterns for Switch/Case statements

### ID: Title

na\_0022: Recommended patterns for Switch/Case statements

### Priority

Mandatory

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

Switch / Case statements must use constant values for the **Case** arguments. Input variables cannot be used in the **Case** arguments.

#### Correct:

```
function outVar = NA_0022_Pass(SwitchVar)
    %# codegen
    switch SwitchVar
```



```
case Case_1_Parameter % Parameter
    outVar = 0;
case NA_0022.Case % Enumerated Data type
    outVar = 1;
case 3 % Hard Code Value
    outVar = 2;
otherwise
    outVar = 10;
end
end
```

**Incorrect:**

```
function outVar = NA_0022_Fail(Case_1, Case_2, Case_3, SwitchVar)
%# codegen
switch SwitchVar
case Case_1
    outVar = 1;
case Case_2
    outVar = 2;
case Case_3
    outVar = 3;
otherwise
    outVar = 10;
end
end
```

## Rationale

- Verification and Validation
- Code Generation
- Simulation

## See Also

- NASA Orion style guideline jh\_0026: Switch / Case statement

## Last Changed

V3.0

**Introduced in R2013a**

## **MATLAB Function Usage**

- na\_0016: Source lines of MATLAB Functions
- na\_0017: Number of called function levels
- na\_0021: Strings

## na\_0016: Source lines of MATLAB Functions

### ID: Title

na\_0016: Source lines of MATLAB Functions

### Priority

Mandatory

### Scope

NA-MAAB

### MATLAB Versions

See description

### Prerequisites

None

### Description

The length of MATLAB functions should be limited, with a recommended limit of 60 lines of code. This restriction applies to MATLAB Functions that reside in the Simulink block diagram and external MATLAB files with a `.m` extension.

If sub-functions are used, they may use additional lines of code. Also limit the length of sub-functions to 60 lines of code.

## Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

## See Also

- NASA Orion style guideline IM\_0008: Source lines of eML

## Last Changed

V3.0

## Model Advisor Check

**By Task > Modeling Standards for MAAB > MATLAB Functions > Check MATLAB Function block metrics**

For check details, see “Check MATLAB Function metrics”.

**Introduced in R2013a**

## na\_0017: Number of called function levels

### ID: Title

na\_0017: Number of called function levels

### Priority

Mandatory

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

The number of levels of sub-functions should be limited, typically to 3 levels. MATLAB Function blocks that reside at the Simulink block diagram level count as the first level, unless it is simply a wrapper for an external MATLAB file with a `.m` extension.

This includes functions that are defined within the MATLAB block and those in the separate `.m` files.

## Note

Standard utility functions, such as built-in functions like `sqrt` or `log`, are not include in the number of levels. Likewise, commonly used custom utility functions can be excluded from the number of levels.

## Rationale

- Readability
- Verification and Validation

## Last Changed

V3.0

**Introduced in R2013a**

## na\_0021: Strings

### ID: Title

na\_0021: Strings

### Priority

Strongly recommended

### Scope

NA-MAAB

### MATLAB Versions

All

### Prerequisites

None

### Description

The use of strings is not recommended. MATLAB Functions store strings as character arrays. The arrays cannot be re-sized to accommodate a string value of different length, due to lack of dynamic memory allocation. Strings are not a supported data type in Simulink, so MATLAB Function blocks cannot pass the string data outside the block.

For example, the following code will produce an error:

```
name='rate_error'; %this creates a 1 x 10 character array
```



```
name = 'x_rate_error'; %this causes an error because the array
size is now 1 x 12, not 1 x 10.
```

## Note

If the string is being used for switch / case behavior, consider using enumerated data types

## Rationale

- Verification and Validation
- Workflow
- Code Generation

## See Also

- NASA Orion style guideline jh\_0024: Strings

## Last Changed

V3.0

**Introduced in R2013a**



# 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, it should be

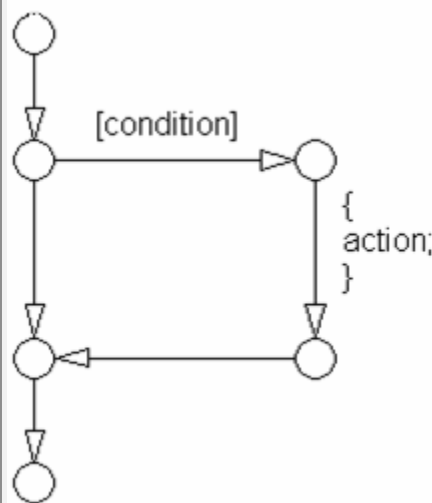
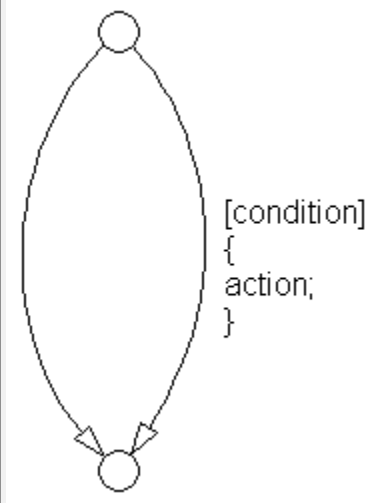
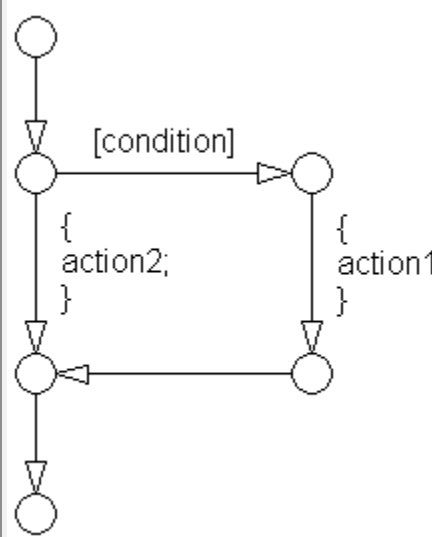
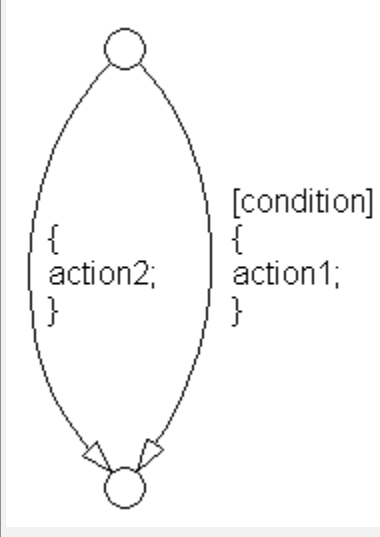
Understandable and unambiguous	<p>A guideline's description should be precise, clearly worded, concise, and should define a characteristic of a model (or part of a model) that a checking tool can evaluate. Use the words "must," "shall," "should," and "may" carefully; they have distinct meanings that 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.</p> <p>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.)</p>
Easy to find	
Minimal	<p>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.</p>

	Small, focused guidelines are less likely to change due to compromise and easier adoption.
--	--

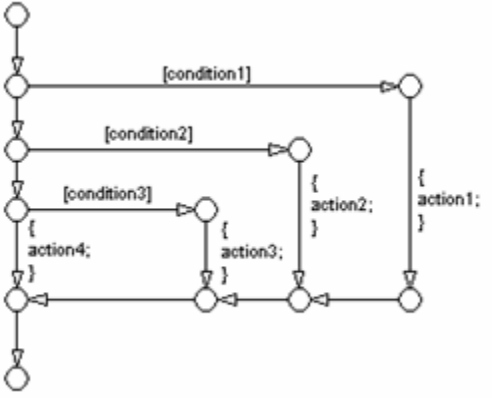
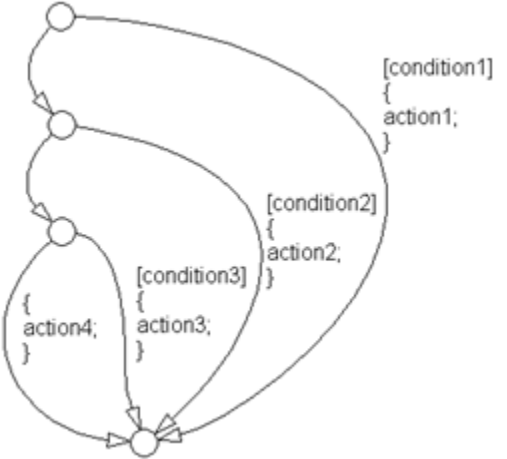
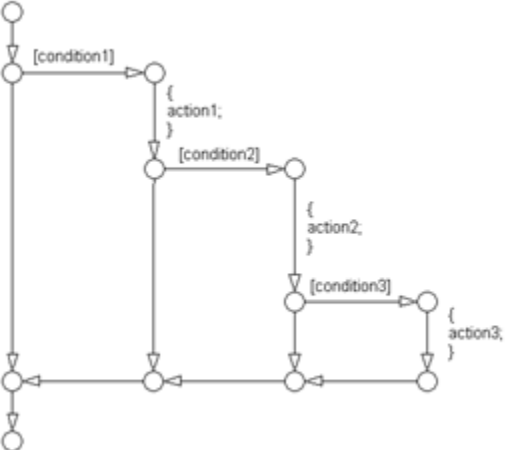
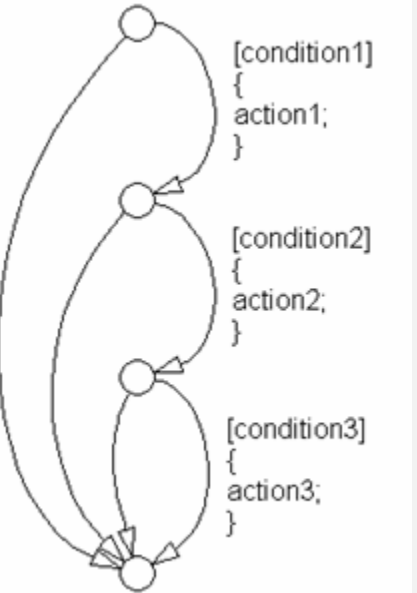
# Flow Chart Reference

---

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

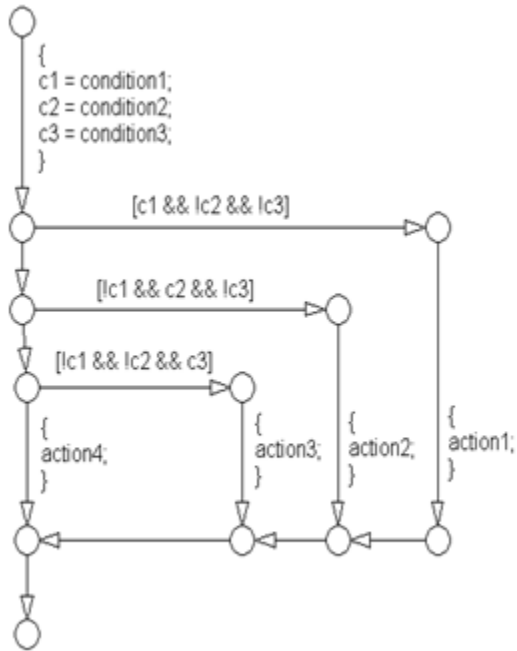
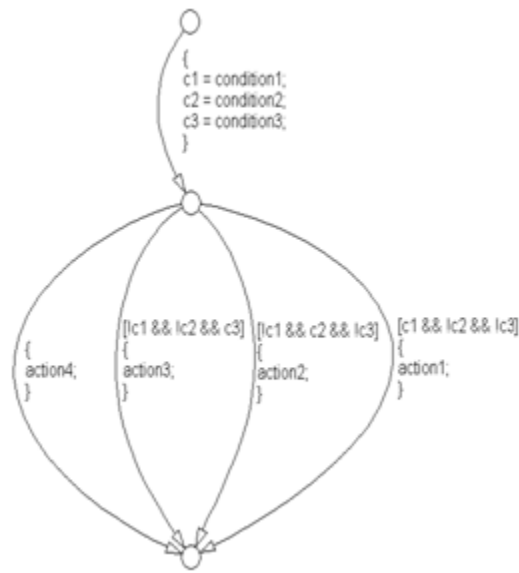
Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="111 296 230 329">if then</p>  <pre> graph TD     N1(( )) --&gt; N2(( ))     N2 -- "[condition]" --&gt; N3(( ))     N3 --&gt; N4(( ))     N4 --&gt; N5(( ))     N4 --&gt; N3     </pre>	 <pre> graph TD     N1(( )) --&gt; N2(( ))     N2 -- "[condition]" --&gt; N3(( ))     N3 --&gt; N4(( ))     N4 --&gt; N2     </pre>
<p data-bbox="111 887 304 920">if then else</p>  <pre> graph TD     N1(( )) --&gt; N2(( ))     N2 -- "[condition]" --&gt; N3(( ))     N3 --&gt; N4(( ))     N4 --&gt; N5(( ))     N4 --&gt; N3     N2 --&gt; N6(( ))     N6 --&gt; N4     </pre>	 <pre> graph TD     N1(( )) --&gt; N2(( ))     N2 -- "[condition]" --&gt; N3(( ))     N3 --&gt; N4(( ))     N4 --&gt; N2     N2 --&gt; N5(( ))     N5 --&gt; N4     </pre>



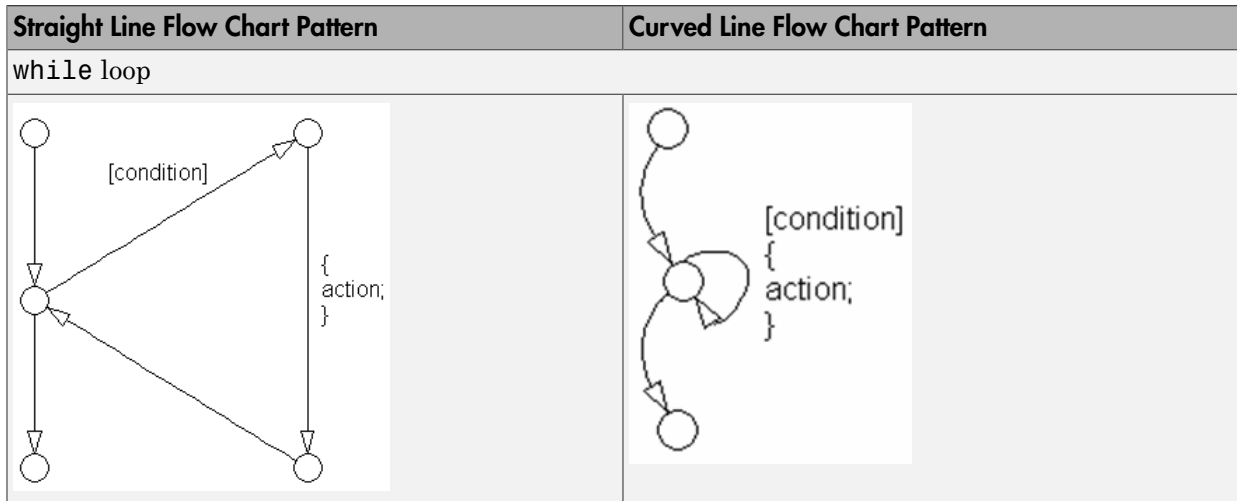
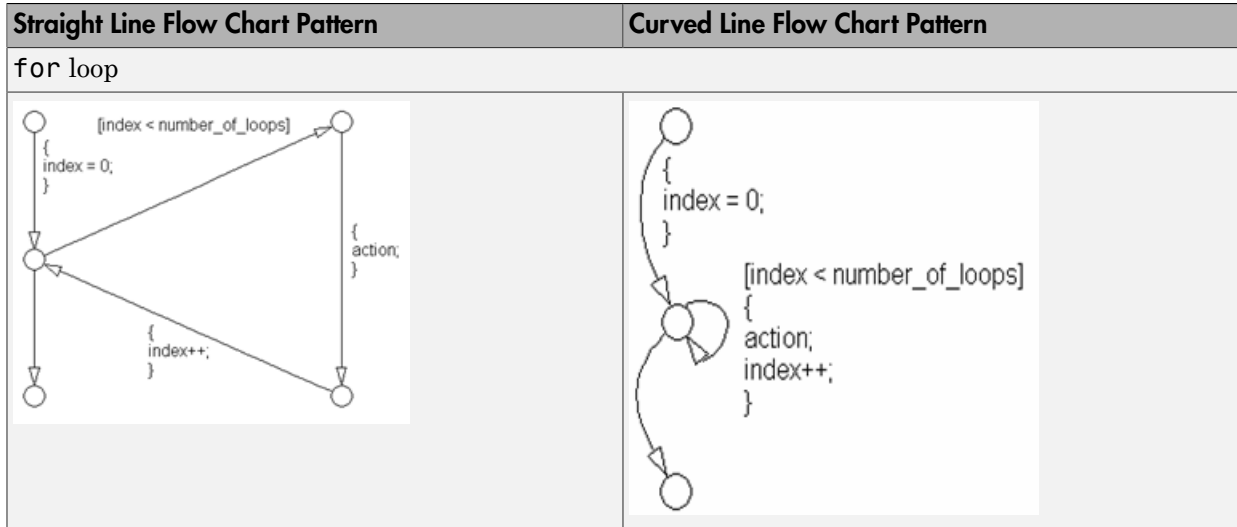
Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="111 300 357 331">if then else if</p> 	
<p data-bbox="111 843 368 874">Cascade of if then</p> 	

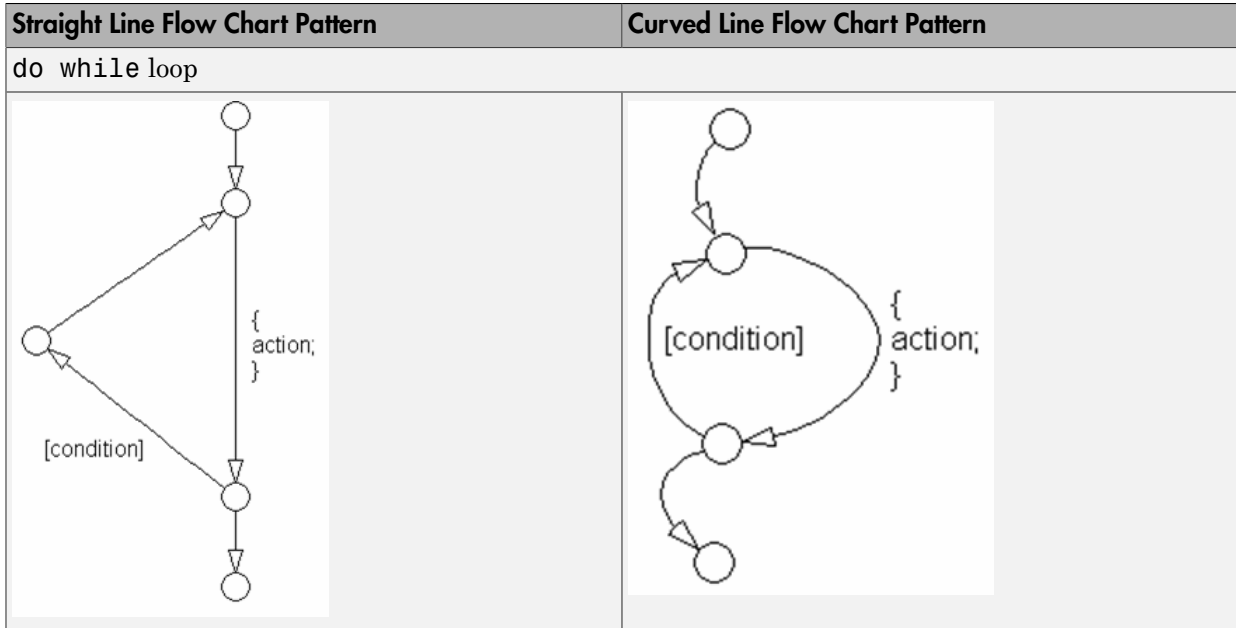
The following patterns are used for case constructs within Stateflow flow charts:

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
case with exclusive selection	
case with exclusive conditions	

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

The following patterns are used for for loops within Stateflow flow charts:

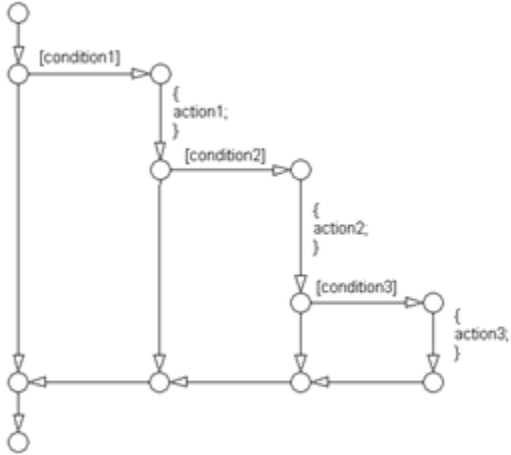
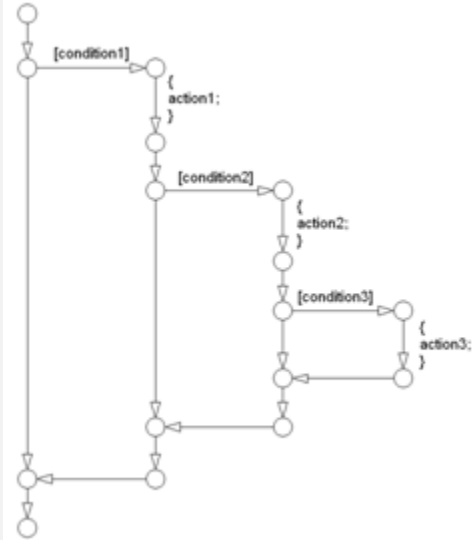




The following patterns are alternately used for If-then-else-if constructs within Stateflow flow charts:

Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
if then else if	

Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
Cascade of if then	

**Straight Line Flow Chart Pattern****Alternate Straight Line Flow Chart Pattern**






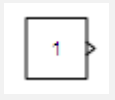
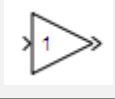

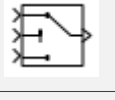
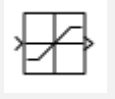
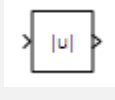
# Background Information on Basic Blocks and Signals

---

## Basic Blocks

This document uses the term *basic blocks* to refer to blocks built into the “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 names 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.

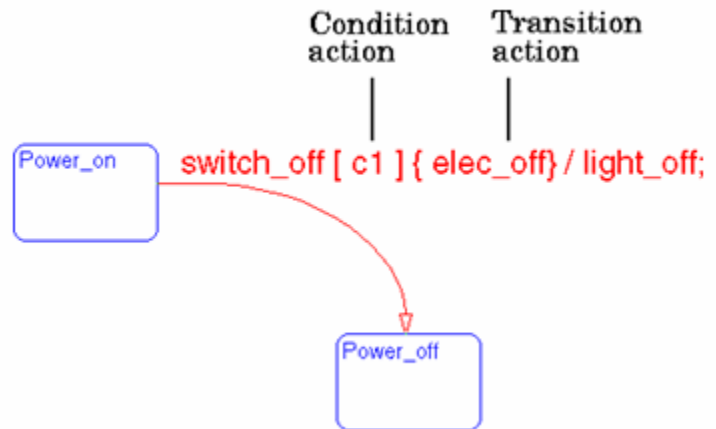
Unless explicitly stated otherwise, the guidelines in “Signals” on page 6-42 apply to all types of signals.

For more information about the representation of signals in Simulink models, see “Signal Basics” in the Simulink documentation.

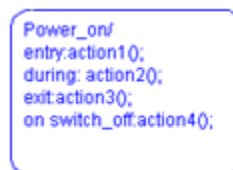


## Actions

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.

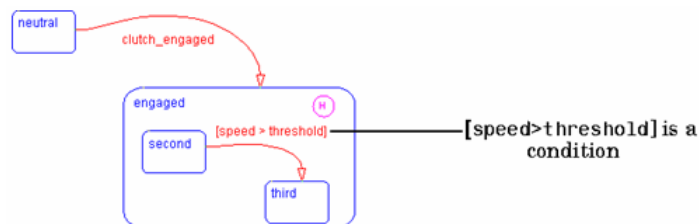
The action language defines the categories of actions you can specify and their associated notations. Violations of the action language notation are flagged as errors by the parser. This section describes the action language notation rules.

### 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,

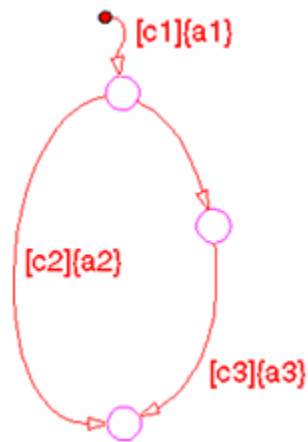


The action language defines the notation to define conditions associated with transitions.

### Connective Junction

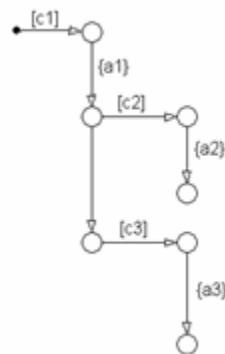
Connective junctions are decision points in the system. A connective junction is a graphical object that simplifies

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 flow charts that start from a transition segment that, in turn, starts from a state or a default transition segment.

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

A flow chart 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 flow charts 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 Simulink Coder 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

<b>Parallelism</b>	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.
<b>Real-Time System</b>	A system that uses actual hardware to implement algorithms, for example, digital signal processing or control applications.
<b>Simulink Coder</b>	Simulink Coder 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.
<b>Simulink Coder Target</b>	An executable built from code generated by the Simulink Coder product.
<b>S-function</b>	A customized Simulink block written in C or MATLAB-code. S-functions written in C can be inlined in the Simulink Coder 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.
<b>Signal propagation</b>	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.
<b>Signal source</b>	The signal source is the block of origin for a signal. The signal source may or may not be the true source.
<b>Simulink</b>	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**

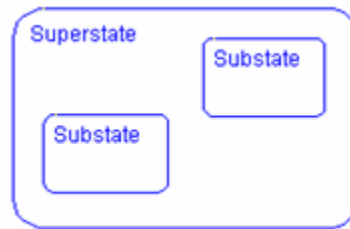
Use the Stateflow Explorer to add, remove, and modify data, event, and target objects.

**Stateflow Finder**

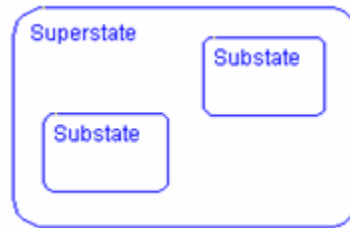
Use the Finder to display a list of objects based on search criteria that you specify. You can directly access the properties dialog box of any object in the search output display by clicking on that object.

**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 Simulink Coder 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 flow chart 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 Outputport 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),



Block Name	Condition Under Which Block Is Virtual
	and does not reside in the root (top-level) Simulink window.
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.

