

MathWorks® Automotive Advisory Board

Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow®

R2013a

MATLAB® & SIMULINK®

How to Contact MathWorks



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

Web
Newsgroup
Technical Support



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

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

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

© COPYRIGHT 2007–2013 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 for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

March 2009 Online only
September 2009 Online only
March 2010 Online only
September 2010 Online only
April 2011 Online only
September 2011 Online only
March 2012 Online only
September 2012 Online only
March 2013 Online only

New for Version 2.0 (Release 2009a)
Revised for Version 2.1 (Release 2009b)
Rereleased for Version 2.1 (Release 2010a)
Rereleased for Version 2.1 (Release 2010b)
Rereleased for Version 2.1 (Release 2011a)
Rereleased for Version 2.1 (Release 2011b)
Rereleased for Version 2.2 (Release 2012a)
Rereleased for Version 2.2 (Release 2012b)
Revised for Version 3.0 (Release 2013a)

Introduction

1

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

Software Environment

2

General Guidelines	2-2
-------------------------------------	-----

Naming Conventions

3

General Guidelines	3-2
Model Content	3-9

Model Architecture

4

Simulink and Stateflow Partitioning	4-2
Subsystem Hierarchies	4-14
J-MAAB Model Architecture Decomposition	4-26

Model Configuration Options

5

Model Configuration Options	5-2
-----------------------------------	-----

Simulink

6

Diagram Appearance	6-2
Signals	6-33
Block Usage	6-42

Block Parameters	6-66
Simulink Patterns	6-72

Stateflow

7

Chart Appearance	7-2
Stateflow Data and Operations	7-20
Events	7-42
Statechart Patterns	7-47
Flowchart Patterns	7-53
State Chart Architecture	7-69

Enumerated Data

8

General Guidelines	8-2
--------------------------	-----

MATLAB Functions

9

MATLAB Function Appearance	9-2
MATLAB Function Data and Operations	9-7

MATLAB Function Patterns	9-11
MATLAB Function Usage	9-14

Recommendations for Automation Tools

A

Guideline Writing

B

Flowchart Reference

C

**Background Information on Basic Blocks and
Signals**

D

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

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 1-12

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 1-7
“Scope” on page 1-8
“MATLAB Versions” on page 1-9
“Prerequisites” on page 1-9
“Description” on page 1-9
“Rationale” on page 1-10
“Last Change” on page 1-11
“Model Advisor Check” on page 1-11

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

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

Description	Description of the guideline (text, images)
Rationale	Motivation for the guideline
Last Change	Version number of last change
Model Advisor Check	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
Definition		
<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>
Consequences: If the guideline is violated,		

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

Scope

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

Scope	Description
MAAB (MathWorks Automotive Advisory Board)	A group of automotive manufacturers and suppliers that work closely together with 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"> • Readable models • Uniform appearance of models, code, and documentation • Clean interfaces • Professional documentation
Workflow	Effective development process and workflow <ul style="list-style-type: none"> • Ease of maintenance • Rapid model changes • Reusable components • Problem-free exchange of models • Model portability
Simulation	Efficient simulation and analysis <ul style="list-style-type: none"> • Simulation speed • Simulation memory • Model instrumentation
Verification and validation	Ability to verify and validate a model and generated code with: <ul style="list-style-type: none"> • Requirements traceability • Testing • Problem-free system integration • Clean interfaces
Code generation	Generation of code that is efficient and effective for embedded systems <ul style="list-style-type: none"> • Fast software changes • Robustness of generated code

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 “MathWorks Automotive Advisory Board Checks” in the Simulink Verification and Validation documentation. For information on using the Model Advisor, see “Consult the Model Advisor” in the Simulink documentation.

Document Usage

- *Name Conventions* and *Model Architecture* provide basic guidelines that apply to all types of models.
- *Simulink* and *Stateflow*[®] 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, “Recommendations for Automation Tools”.

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	<p>During software development, it is recommended that a consistent software environment is used across the project. Software includes, but is not limited, to:</p> <ul style="list-style-type: none">• MATLAB• Simulink• C Compiler (for simulation)• C Compiler (for target hardware) <p>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.</p>
Rationale	<ul style="list-style-type: none">• Readability• Code Generation
See Also	<ul style="list-style-type: none">• jh_0042: Required software
Last Changed	V3.0

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	<p>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.</p> <p>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.</p>
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Code Generation• Simulation
See Also	<ul style="list-style-type: none">• hyl_0201: Use of standard library blocks only
Last Changed	V3.0

Naming Conventions

- “General Guidelines” on page 3-2
- “Model Content” on page 3-9

General Guidelines

- ar_0001: Filenames
- ar_0002: Directory names
- na_0035: Adoption of naming conventions

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:

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789_

extension:

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789

Underscores

name:

ar_0001: Filenames

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

extension:

Should not use underscores

Rationale

- Readability
- Workflow
- Code Generation
- Simulation

Last Changed

V3.0

Model Advisor Check

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

Priority Mandatory

Scope MAAB

MATLAB Versions All

Prerequisites None

Description A directory name conforms to the following constraints:

Form

directory name = name

name: no leading digits, no blanks

Uniqueness

All directory names within the parent project directory

Allowed characters

name:

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

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

0 1 2 3 4 5 6 7 8 9 _

Underscores

name:

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

Rationale

- Readability

- Workflow

ar_0002: Directory names

- Code Generation
- Simulation

Last Changed

V1.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming
Conventions > “Check folder names”

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	<p>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:</p> <ul style="list-style-type: none">• Compliance with the programming language and downstream tools<ul style="list-style-type: none">▪ Length▪ Use of symbols• Readability<ul style="list-style-type: none">▪ Use of underscores▪ Use of capitalization• Encoding information<ul style="list-style-type: none">▪ Use of “meaningful” names▪ Standard abbreviations and acronyms▪ Data type▪ Engineering units▪ Data ownership▪ Memory type
Rationale	<ul style="list-style-type: none">• Readability

na_0035: Adoption of naming conventions

- Workflow
- Code Generation
- Simulation

**Last
Changed**

V3.0

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	<p>The names of all Subsystem blocks should conform to the following constraints:</p> <p>Form</p> <p><i>name:</i></p> <ul style="list-style-type: none">• Should not start with a number• Should not include blank spaces• Should not include carriage returns <p>Allowed Characters</p> <p><i>name:</i></p> <p>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 _</p> <p>Underscores</p> <p><i>name:</i></p> <ul style="list-style-type: none">• Can use underscores to separate parts• Cannot have more than one consecutive underscore• Cannot start with an underscore• Cannot end with an underscore

jc_0201: Usable characters for Subsystem names

Rationale

- Readability

**Last
Changed**

V2.2

**Model
Advisor
Check**

By Task > Modeling Standards for MAAB > Naming
Conventions > “Check subsystem names”

jc_0211: Usable characters for Inport blocks and Output blocks

ID: Title jc_0211: Usable characters for Inport blocks and Output blocks

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

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

Form

name:

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

jc_0211: Usable characters for Inport blocks and Output blocks

Rationale

- Readability

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > “Check port block names”

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

jc_0221: Usable characters for signal line names

- 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”

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

na_0030: Usable characters for Simulink Bus names

Rationale

- Readability

See Also

- jh_0040: Usable characters for Simulink Bus Names

Last Changed

V3.0

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

jc_0231: Usable characters for block names

Last Changed

V2.0

Model Advisor Check

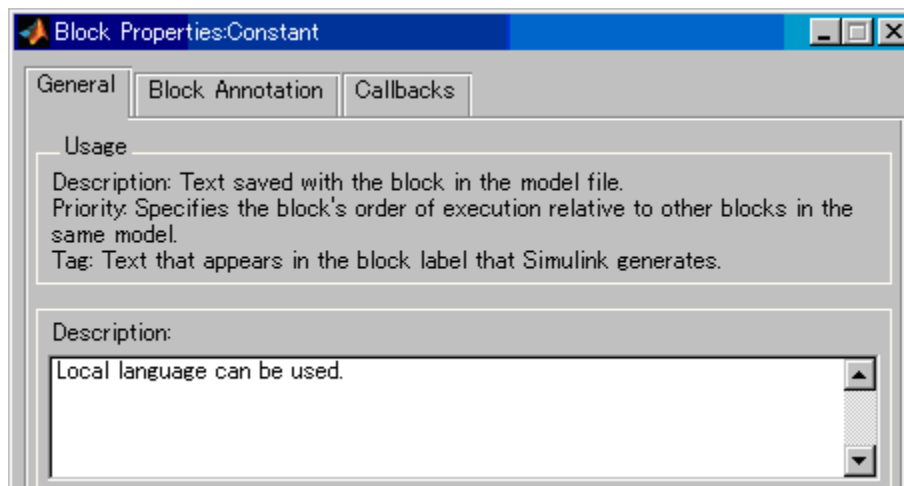
By Task > Modeling Standards for MAAB > Naming
Conventions > “Check character usage in block names”

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.

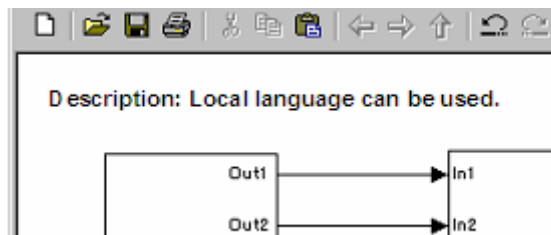
Simulink Examples

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



- Text annotation entered directly in the model

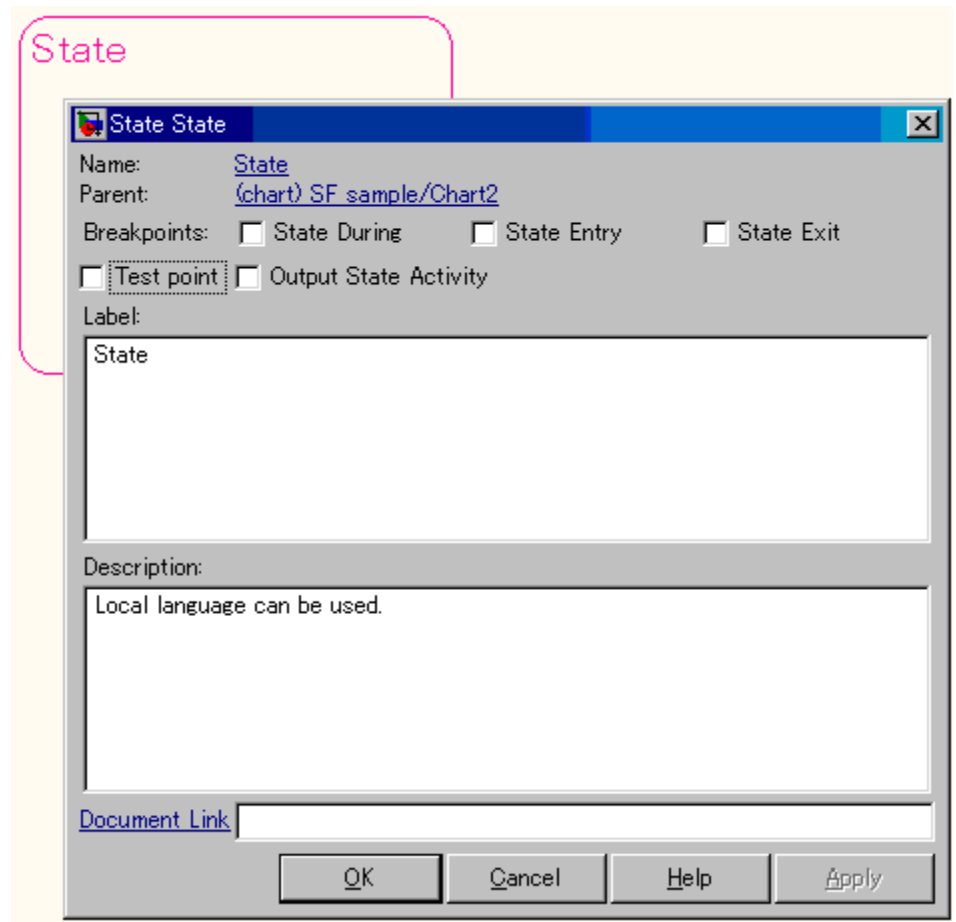
na_0014: Use of local language in Simulink and Stateflow



Stateflow Examples

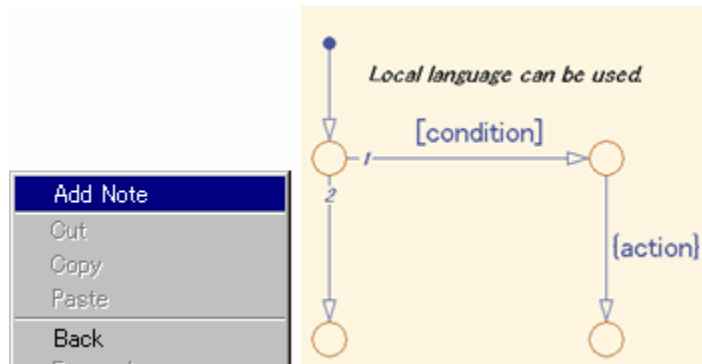
- The **Description** field of chart and state Properties

na_0014: Use of local language in Simulink and Stateflow



- Annotation description added using **Add Note**

na_0014: Use of local language in Simulink and Stateflow



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

Rationale

- Readability

Last Changed

V2.0

Model Advisor Check

Not applicable

na_0014: Use of local language in Simulink and Stateflow

Model Architecture

- “Simulink and Stateflow Partitioning” on page 4-2
- “Subsystem Hierarchies” on page 4-14
- “J-MAAB Model Architecture Decomposition” on page 4-26

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, “Background Information on Basic Blocks and Signals” lists some examples of basic blocks.

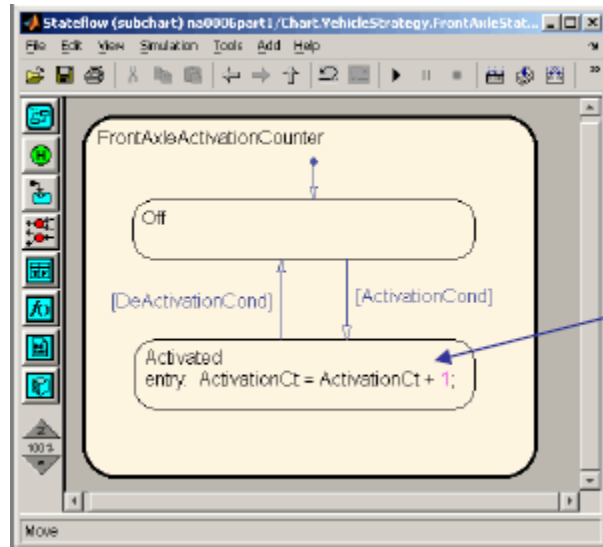
Simulink and Stateflow Partitioning

- na_0006: Guidelines for mixed use of Simulink and Stateflow
- na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

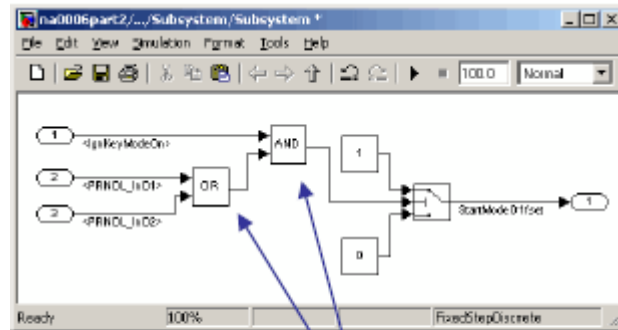
na_0006: Guidelines for mixed use of Simulink and Stateflow

ID: Title	na_0006: Guidelines for mixed use of Simulink and Stateflow
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.</p> <ul style="list-style-type: none">• 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 <i>past and present logical conditions</i>.• If the function primarily involves numerical operations, use Simulink features. <p>Specifics</p> <ul style="list-style-type: none">• 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.

na_0006: Guidelines for mixed use of Simulink and Stateflow

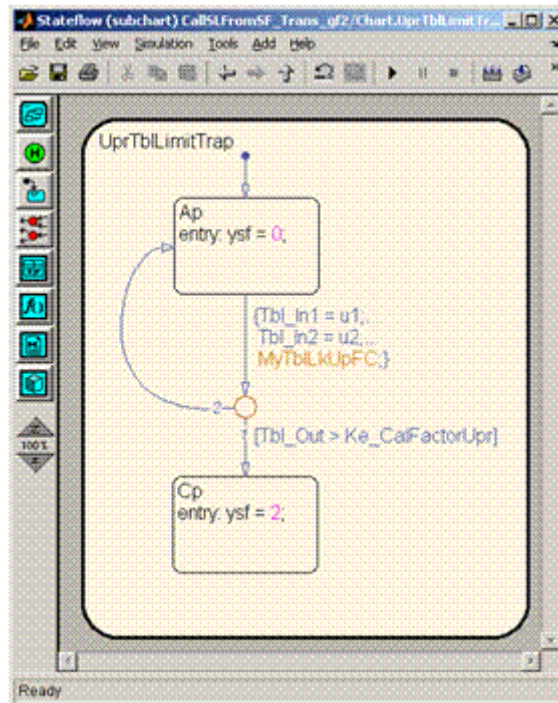


- If the primary nature of the function is numeric, but some simple logical operations are done to support the arithmetic, implement the simple logical functions with Simulink blocks.

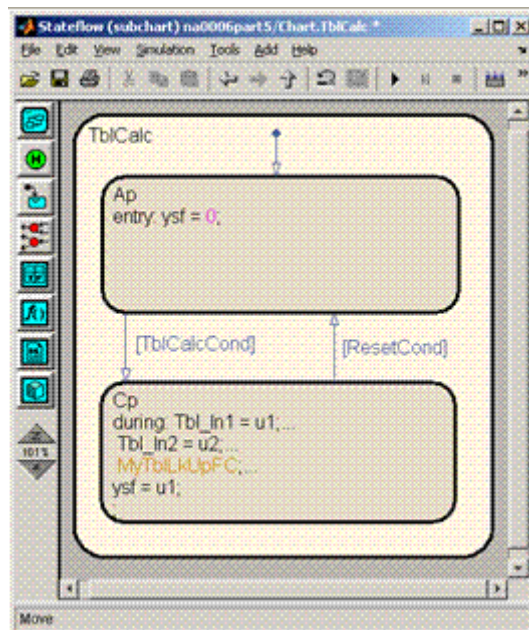
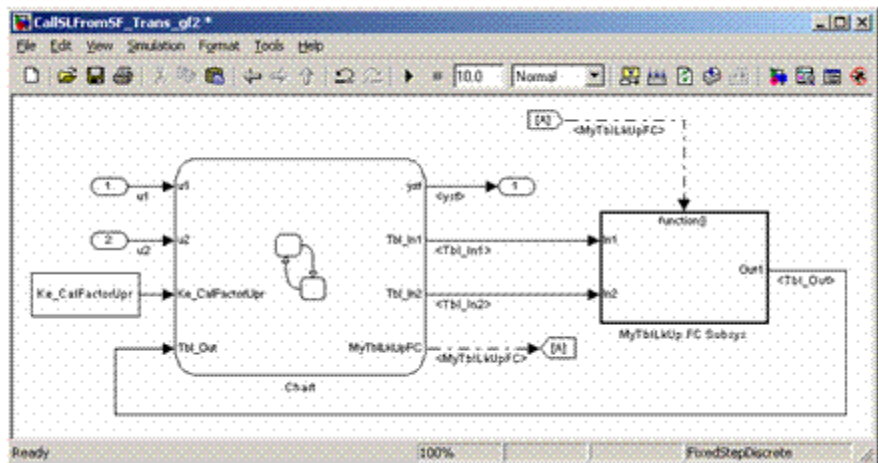


na_0006: Guidelines for mixed use of Simulink and Stateflow

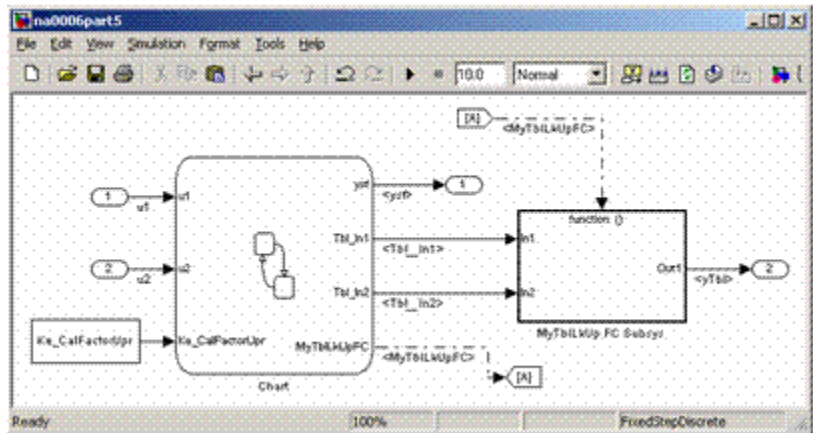
- If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, use a Simulink subsystem to implement the numerical calculations. The Stateflow software should invoke the execution of the subsystem, using a function call.



na_0006: Guidelines for mixed use of Simulink and Stateflow

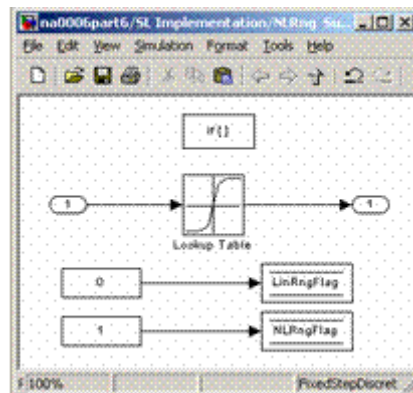
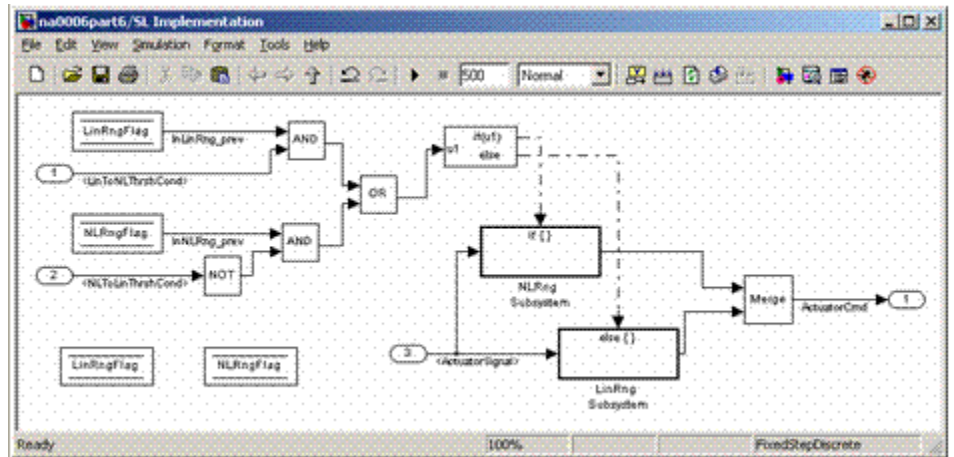


na_0006: Guidelines for mixed use of Simulink and Stateflow



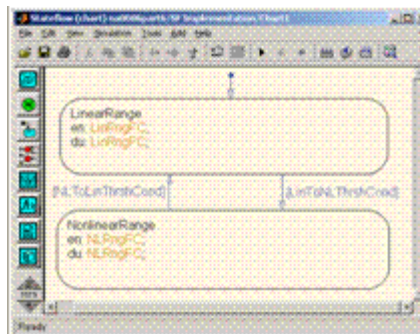
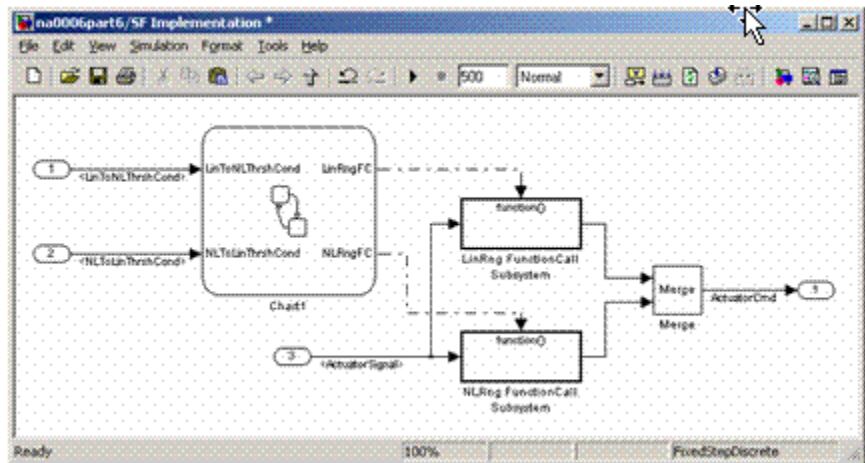
- Use the Stateflow product to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*. (If there is a need to store the result of a logical condition test in a Simulink model, for example, by storing a flag,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

na_0006: Guidelines for mixed use of Simulink and Stateflow



Incorrect

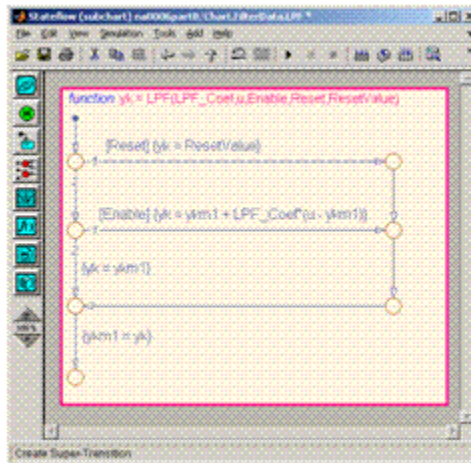
na_0006: Guidelines for mixed use of Simulink and Stateflow



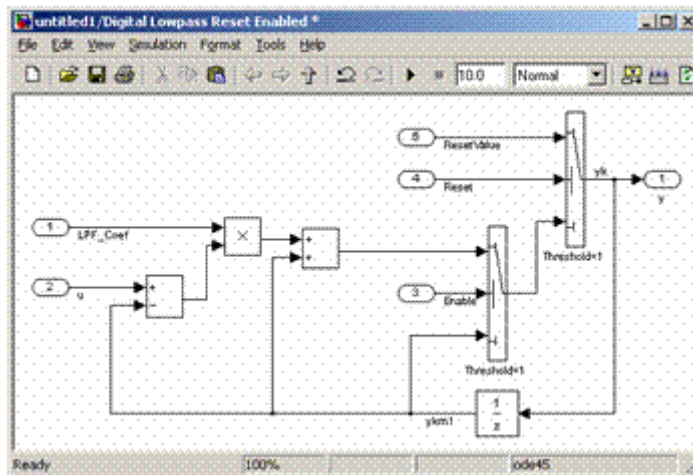
Correct

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

na_0006: Guidelines for mixed use of Simulink and Stateflow



Incorrect



Correct

Rationale

- Readability
- Workflow

na_0006: Guidelines for mixed use of Simulink and Stateflow

- Simulation
- Verification and Validation
- Code Generation

**Last
Changed**

V2.0

**Model
Advisor
Check**

Not applicable

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

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 flowcharts 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 “Flowchart Patterns” on page 7-53.)

- Rationale**
- Readability
 - Workflow

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

- Simulation
- Verification and Validation
- Code Generation

**Last
Changed**

V2.0

**Model
Advisor
Check**

Not applicable

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

Subsystem Hierarchies

- db_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

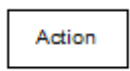


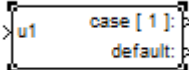
Scope NA-MAAB

MATLAB Versions All

Prerequisites None





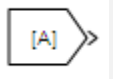


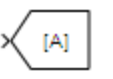
Description To allow partitioning of the model into discreet units, every level of a model must be designed with building blocks of the same type (i.e. only Subsystems or only “Basic Blocks”). The blocks listed in this guideline are used for signal routing. You can place them at any level of the model.

Blocks that You Can Place at any Model Level

Block	Example
Action port ¹	 A rectangular block with the word "Action" centered inside.
Bus Creator	 A vertical black bar with a white arrow pointing to the right, indicating signal flow.
Bus Selector	 A vertical black bar with a white arrow pointing to the left, indicating signal flow.
Case	 A rectangular block with an input port on the left labeled "u1", an output port on the right, and internal text "case [1]:" and "default:".


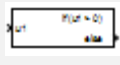

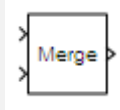

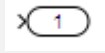
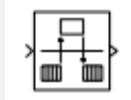
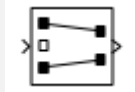

db_0143: Similar block types on the model levels

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

Block	Example
Data Store Memory	
Data Type Conversion	
Demux	
Enable ²	
From	
Function-Call Generator	
Function-Call Split	
Goto	


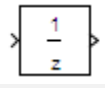
db_0143: Similar block types on the model levels

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

Block	Example
Ground	
If	
Inport	
Merge	
Mux	
Outport	
Rate Transition	
Selector	
Terminator	

db_0143: Similar block types on the model levels

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

Block	Example
Trigger ³	
Unit Delay	

¹Action ports are not allowed at the root level of a model.

²Starting in R2011b, the Enable block is allowed at the root level of the model.

³Starting in R2009a, the Trigger block is allowed at the root level of the model.

Note 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.

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check for mixing basic blocks and subsystems”

ID: Title	db_0144: Use of Subsystems
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>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.</p> <p>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.</p> <p>If creation of a subsystem is required for readability issues, then a virtual subsystem should be used.</p>
Rationale	<ul style="list-style-type: none">• Readability• Workflow• Verification and Validation• Code Generation
Last Changed	V2.2
Model Advisor Check	Not applicable

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

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

na_0037: Use of single variable variant conditionals

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

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	<ul style="list-style-type: none">• Readability• Code Generation• Simulation
Last Changed	V3.0




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

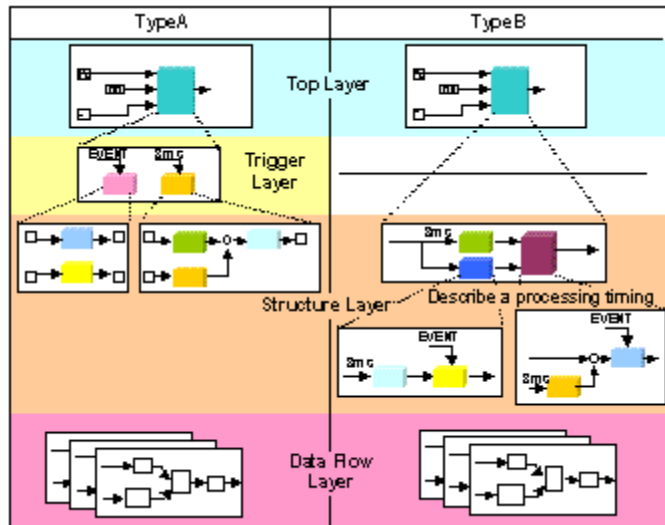
V3.0

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

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

jc_0301: Controller model



Controller Model

Rationale

Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

ID: Title jc_0311: Top layer/root level

Priority Mandatory

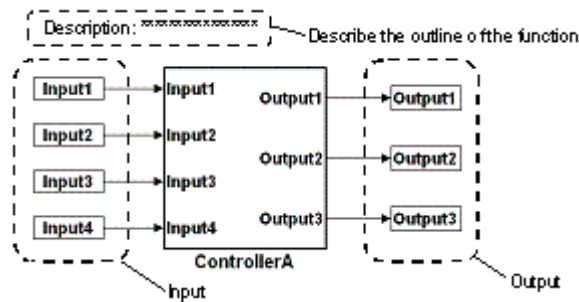
Scope J-MAAB

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

jc_0311: Top layer/root level

**Model
Advisor
Check**

Not applicable

ID: Title jc_0321: Trigger layer

Priority Mandatory

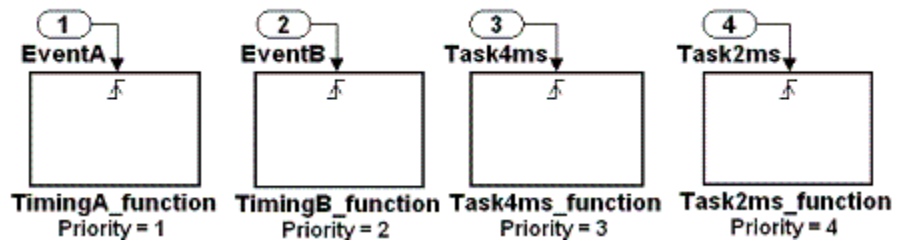
Scope J-MAAB

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

jc_0321: Trigger layer

**Last
Changed**

V2.0

**Model
Advisor
Check**

Not applicable

ID: Title jc_0331: Structure layer

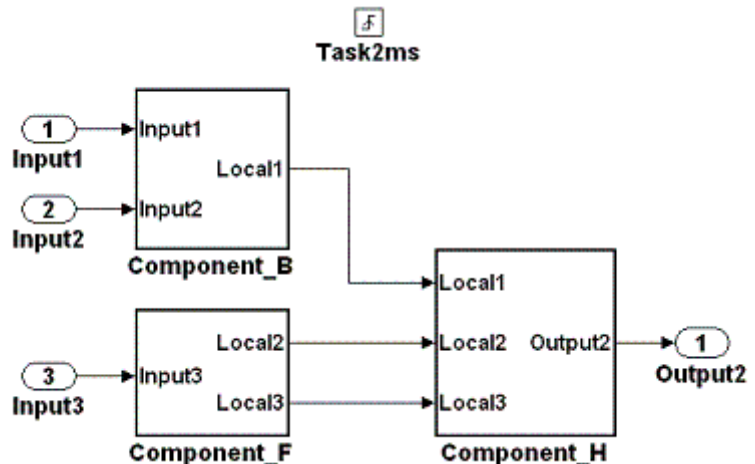
Priority Mandatory

Scope J-MAAB

MATLAB Versions All

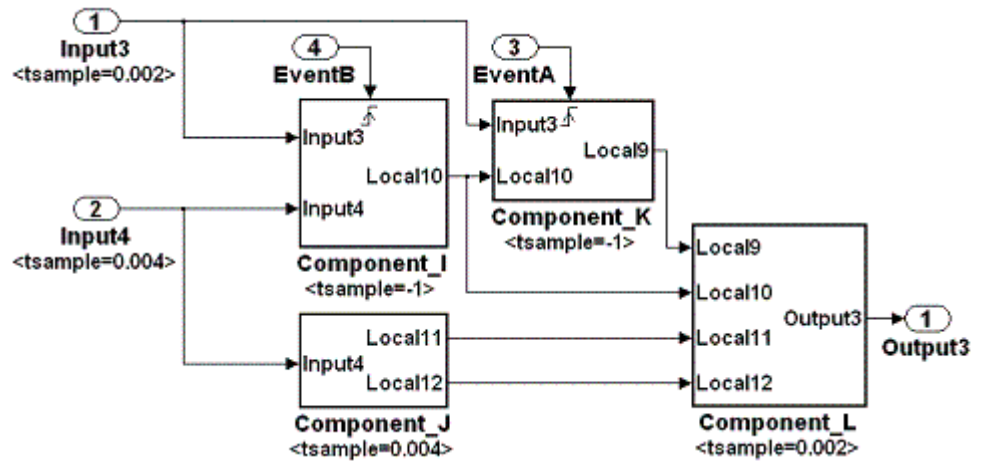
Prerequisites None

- 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)

jc_0331: Structure layer



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

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

Not applicable

ID: Title jc_0341: Data flow layer

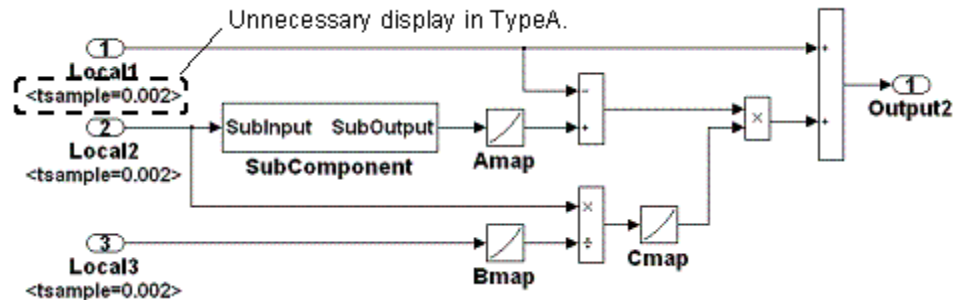
Priority Mandatory

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

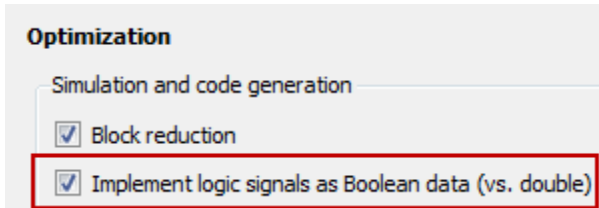
jc_0341: Data flow layer

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	<p>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).</p> 
Rationale	<ul style="list-style-type: none">• Workflow• Code Generation
Last Changed	V2.2
Model Advisor Check	By Task > Modeling Standards for MAAB > Simulink > “Check Implement logic signals as Boolean data (vs. double)”

jc_0021: Model diagnostic settings

ID: Title jc_0021: Model diagnostic settings

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

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

Solver Diagnostics

- Algebraic loop
- Minimize algebraic loop

Sample Time Diagnostics

- Multitask rate transition

Data Validity Diagnostics

- Inf or NaN block output
- Duplicate data store names

Connectivity

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

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

Rationale

- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Model Configuration
Options > “Check model diagnostic parameters”

jc_0021: Model diagnostic settings

Simulink

- “Diagram Appearance” on page 6-2
- “Signals” on page 6-33
- “Block Usage” on page 6-42
- “Block Parameters” on page 6-66
- “Simulink Patterns” on page 6-72

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
Toolbar	Checked
Zoom factor	Normal (100%)

Block Display Options	Setting
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

na_0004: Simulink model appearance

Block Display Options	Setting
Sample Time Colors	Unchecked
Sorted Order	Unchecked

Signal Display Options	Setting
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 using nonstandard display attributes”

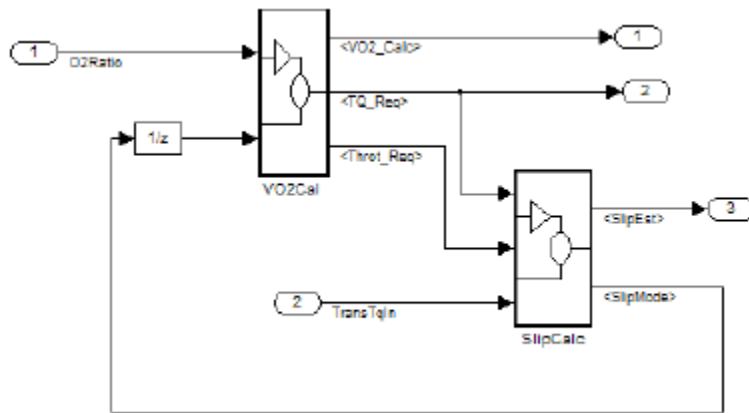
db_0043: Simulink font and font size

ID: Title	db_0043: Simulink font and font size
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>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.</p> <hr/> <p>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).</p> <hr/>
Rationale	<ul style="list-style-type: none">• Readability
Last Changed	V2.0
Model Advisor Check	By Task > Modeling Standards for MAAB > Simulink > “Check font formatting”

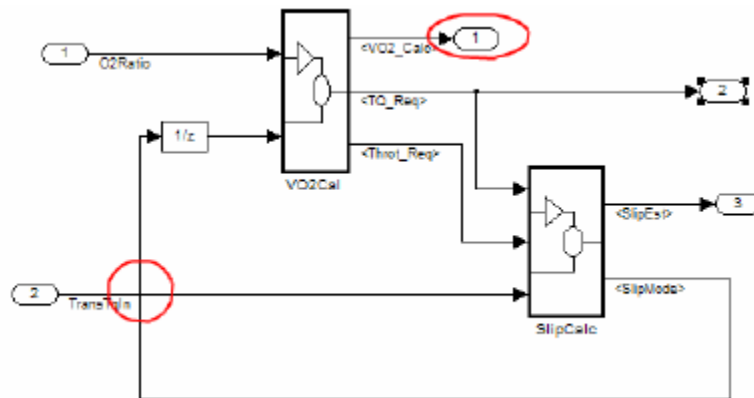
db_0042: Port block in Simulink models

ID: Title	db_0042: Port block in Simulink models
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>In a Simulink model, ports must comply with the following rules:</p> <ul style="list-style-type: none">• 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.<ul style="list-style-type: none">▪ Do not use duplicate Inport blocks at the root level.

db_0042: Port block in Simulink models



Correct



Incorrect

Notes on the incorrect model

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

db_0042: Port block in Simulink models

Rationale

Readability

**Last
Changed**

V2.0

**Model
Advisor
Check**

By Task > Modeling Standards for MAAB > Simulink > “Check positioning and configuration of ports”

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

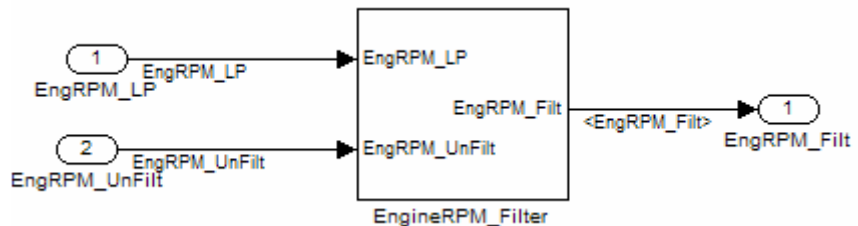
MATLAB Versions All

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:

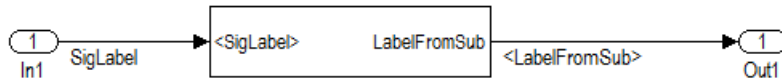
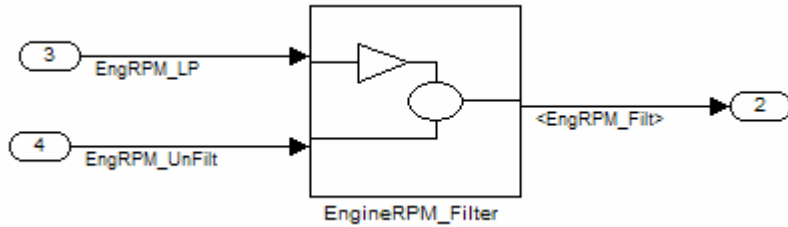
- The name of an Inport or Outport block is not hidden. (**Format > Hide Name** is not allowed.)



- 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.

na_0005: Port block name visibility in Simulink models



Correct: Use of signal label

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

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

jc_0081: Icon display for Port block

ID: Title jc_0081: Icon display for Port block

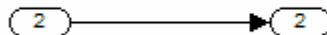
Priority Recommended

Scope MAAB

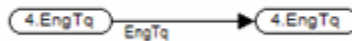
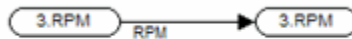
MATLAB Versions R14 and later

Prerequisites None

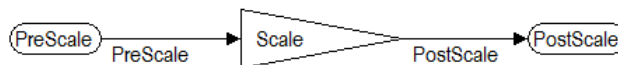
Description The Icon display setting should be set to Port number for Inport and Outport blocks.



Correct



Incorrect



Incorrect

Rationale Readability

jc_0081: Icon display for Port block

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check for unconnected ports and signal lines”

ID: Title jm_0002: Block resizing

Priority Mandatory

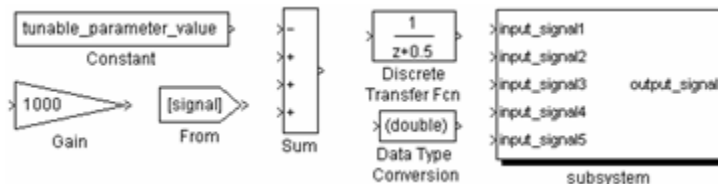
Scope MAAB

MATLAB Versions All

Prerequisites None

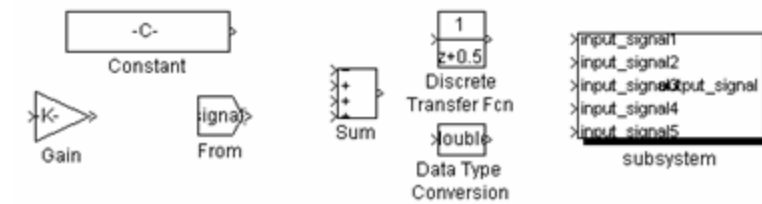
Description All blocks in a model must be sized such that the icon is completely visible and recognizable. In particular, any displayed text (for example, tunable parameters, file names, or equations) in the icon must be readable.

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



Correct

jm_0002: Block resizing



Incorrect

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0142: Position of block names

ID: Title db_0142: Position of block names

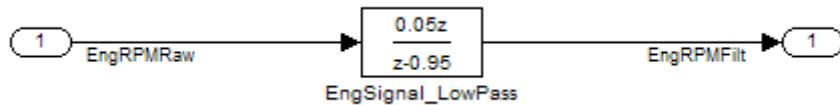
Priority Strongly recommended

Scope MAAB

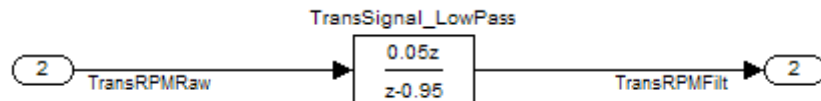
MATLAB Versions All

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 appear below blocks”

jc_0061: Display of block names

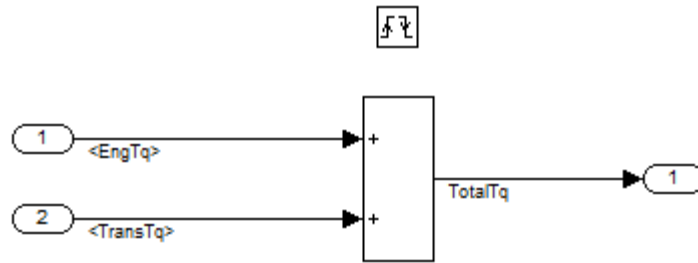
ID: Title	jc_0061: Display of block names
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<ul style="list-style-type: none">• Display a block name when it provides descriptive information.• Do not display a block name if the block function is known and understood from 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”

db_0146: Triggered, enabled, conditional Subsystems

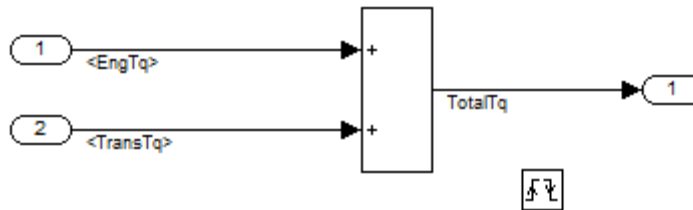
ID: Title	db_0146: Triggered, enabled, conditional Subsystems
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>The blocks that define subsystems as either conditional or iterative should be located at a consistent location at the top of the subsystem diagram. These blocks are:</p> <ul style="list-style-type: none">• 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.

db_0146: Triggered, enabled, conditional Subsystems



Correct



Incorrect

Rationale

- Readability

Last Changed

V2.2

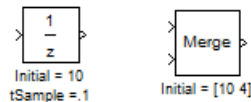
Model Advisor Check

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

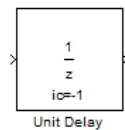
db_0140: Display of basic block parameters

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

db_0140: Display of basic block parameters

Rationale

- Readability

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check for nondefault block attributes”

db_0032: Simulink signal appearance

ID: Title db_0032: Simulink signal appearance

Priority Strongly recommended

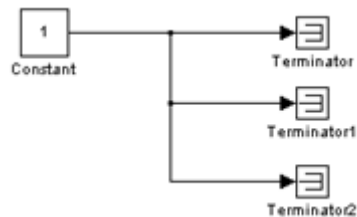
Scope MAAB

MATLAB Versions All

Prerequisites None

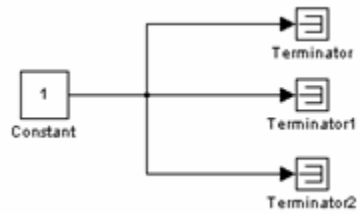
Description Signal lines

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



Correct

db_0032: Simulink signal appearance



Incorrect

Rationale

- Readability

**Last
Changed**

V2.0

**Model
Advisor
Check**

Not applicable

db_0141: Signal flow in Simulink models

ID: Title db_0141: Signal flow in Simulink models

Priority Strongly recommended

Scope MAAB

Versions All

Prerequisites None

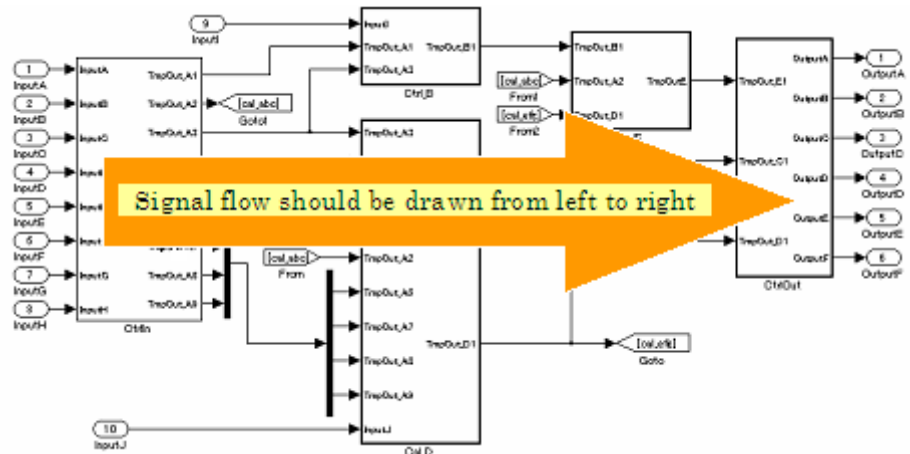
Description The signal flow in a model is from left to right.

Exception: Feedback loops

Sequential blocks or subsystems are arranged from left to right.

Exception: Feedback loops

Parallel blocks or subsystems are arranged from top to bottom.



Rationale • Readability

db_0141: Signal flow in Simulink models

**Last
Changed**

V2.0

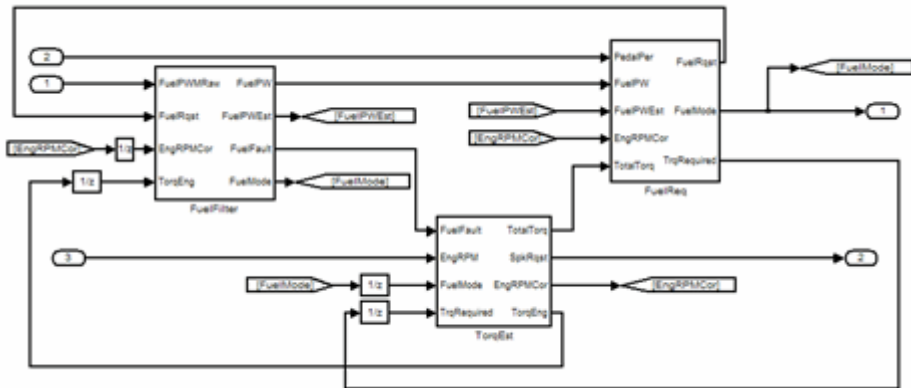
**Model
Advisor
Check**

Not applicable

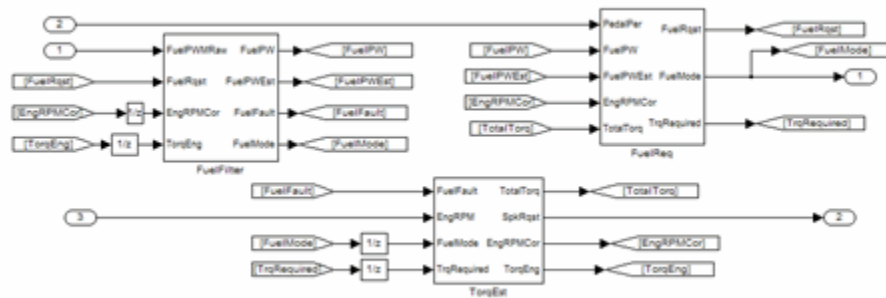
jc_0171: Maintaining signal flow when using Goto and From blocks

ID: Title	jc_0171: Maintaining signal flow when using Goto and From blocks
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<ul style="list-style-type: none">• You must maintain visual depiction of signal flow between subsystems.• You can use Goto and From blocks if:<ul style="list-style-type: none">▪ 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.

jc_0171: Maintaining signal flow when using Goto and From blocks



Correct



Incorrect

Rationale

- Readability

Last Changed

V2.2

jc_0171: Maintaining signal flow when using Goto and From blocks

**Model
Advisor
Check**

Not applicable

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	<p>When using Merge blocks:</p> <ul style="list-style-type: none">• Signals entering a merge block must not branch off to other blocks• With buses:<ul style="list-style-type: none">▪ Buses must be identical This includes:<ul style="list-style-type: none">• Number of elements• Element names• Element order• Element data type• Element size▪ Buses must be either all virtual or all non-virtual• Bus lines entering a merge block must not branch off to other blocks.
Rationale	<ul style="list-style-type: none">• Workflow• Code Generation
See Also	jh_0109: Merge blocks

**Last
Changed**

V3.0

**Model
Advisor
Check**

Not applicable

jm_0010: Port block names in Simulink models

ID: Title jm_0010: Port block names in Simulink models

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites

- db_0042: Port block in Simulink models
- na_0005: Port block name visibility in Simulink models

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

jm_0010: Port block names in Simulink models

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

Rationale

- Readability
- Workflow
- Code Generation
- Simulation

Last Changed

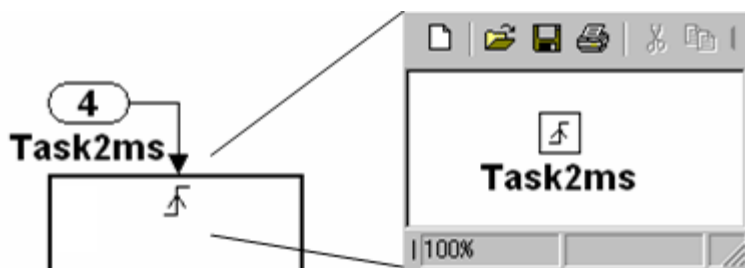
V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check for matching port and signal names”

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 names”

jc_0281: Naming of Trigger Port block and Enable Port block

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.

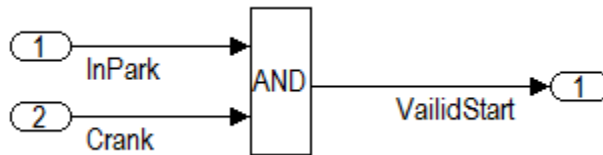
na_0008: Display of labels on signals

ID: Title	na_0008: Display of labels on signals
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<ul style="list-style-type: none">• A label must be displayed on a signal originating from the following blocks:<ul style="list-style-type: none">▪ 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):<ul style="list-style-type: none">▪ Outport block▪ Goto block▪ Data Store Write block▪ Bus Creator block

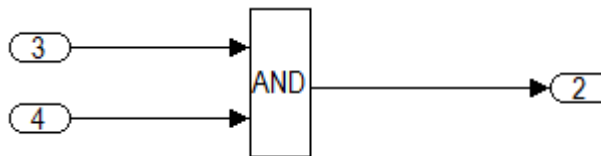
na_0008: Display of labels on signals

- Mux block
- Subsystem block
- Chart block

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



Correct



Incorrect

Rationale

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

na_0008: Display of labels on signals

**Last
Changed**

V2.2

**Model
Advisor
Check**

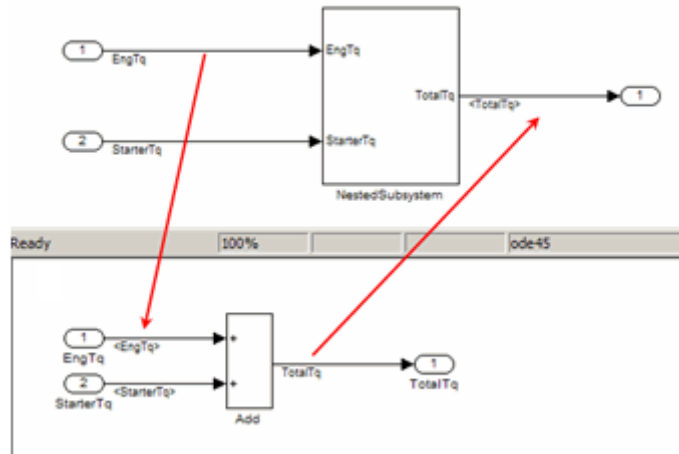
By Task > Modeling Standards for MAAB > Simulink > “Check
signal line labels”

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	<p>If a label is present on a signal, the following rules define whether that label is created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the less than (<) character).</p> <ul style="list-style-type: none">• Any displayed signal label must be <i>entered</i> for signals that:<ul style="list-style-type: none">▪ 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 <i>propagated</i> for signals that:<ul style="list-style-type: none">▪ 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

na_0009: Entry versus propagation of signal labels

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



Rationale

- Readability
- 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 signal labels”

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	<p>The labels must be visually associated with the corresponding signal and not overlap other labels, signals, or blocks.</p> <p>Labels should be located consistently below horizontal lines and close to the corresponding source or destination block.</p>
Rationale	<ul style="list-style-type: none">• Readability
Last Changed	V2.0
Model Advisor Check	Not applicable

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	<p>A system must not have any:</p> <ul style="list-style-type: none">• Unconnected subsystem or basic block inputs• Unconnected subsystem or basic block outputs• Unconnected signal lines <p>In addition:</p> <ul style="list-style-type: none">• An otherwise unconnected input should be connected to a ground block• An otherwise unconnected output should be connected to a terminator block
Rationale	<ul style="list-style-type: none">• Readability• Workflow• Verification and Validation
Last Changed	V2.0

db_0081: Unconnected signals, block inputs and block outputs

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check for unconnected ports and signal lines”

db_0081: Unconnected signals, block inputs and block outputs

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`

na_0003: Simple logical expressions in If Condition block

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 && u2 u3</code>	(too many primary expressions)
<code>u1 && (u2 u3)</code>	(unacceptable operator within primary expression)

na_0003: Simple logical expressions in If Condition block

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

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.2

Model Advisor Check

Not applicable

na_0002: Appropriate implementation of fundamental logical and numerical operations

ID: Title na_0002: Appropriate implementation of fundamental logical and numerical operations

Priority Mandatory

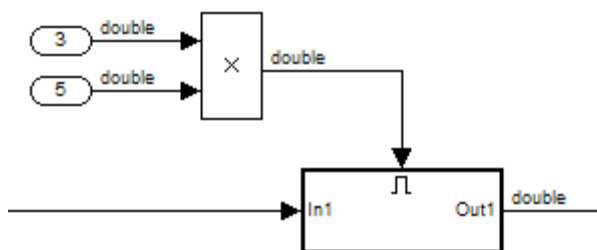
Scope MAAB

MATLAB Versions All

Prerequisites None

Description

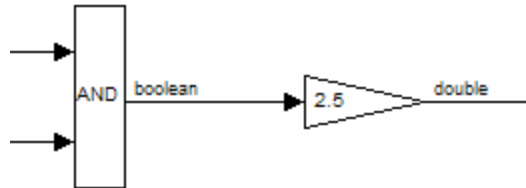
- Blocks that are intended to perform numerical operations must not be used to perform logical operations.



Incorrect

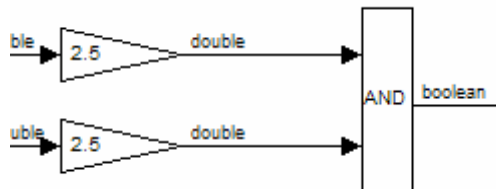
- A logical output should never be connected directly to the input of blocks that operate on numerical inputs.
- The result of a logical expression fragment should never be operated on by a numerical operator.

na_0002: Appropriate implementation of fundamental logical and numerical operations



Incorrect

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



Incorrect

Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

Last Changed

V3.0

na_0002: Appropriate implementation of fundamental logical and numerical operations

**Model
Advisor
Check**

Not applicable

jm_0001: Prohibited Simulink standard blocks inside controllers

ID: Title jm_0001: Prohibited Simulink standard blocks inside controllers

Priority Mandatory

Scope MAAB

MATLAB Versions All

Prerequisites None

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




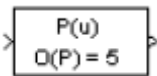

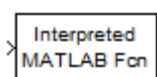



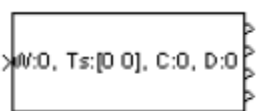


Sine Wave



jm_0001: Prohibited Simulink standard blocks inside controllers

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

jm_0001: Prohibited Simulink standard blocks inside controllers

Model Advisor Checks

- By Task > Modeling Standards for MAAB > Simulink > “Check for blocks not recommended for C/C++ production code deployment”
- By Task > Modeling Standards for MAAB > Simulink > “Check for prohibited blocks in discrete controllers”

hd_0001: Prohibited Simulink sinks

ID: Title hd_0001: Prohibited Simulink sinks

Priority Strongly recommended

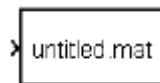
Scope MAAB

MATLAB Versions All

Prerequisites None

Description Controller models must be designed from discrete blocks.
The following sink blocks *are not* allowed:

To File



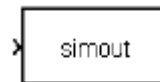
Stop

Simulation



To

Workspace



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”

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.

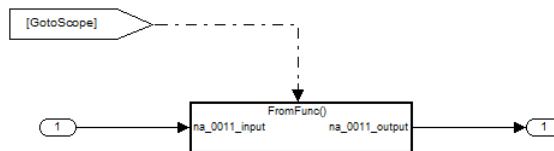
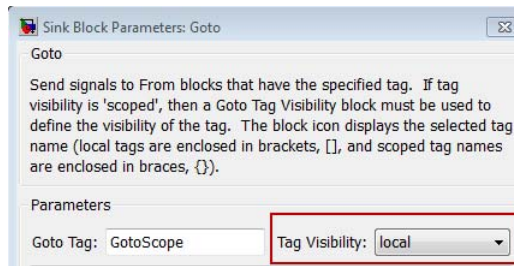
Note Control flow signals may use global scope.

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.

na_0011: Scope of Goto and From blocks



Rationale

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

Last Changed

V2.2

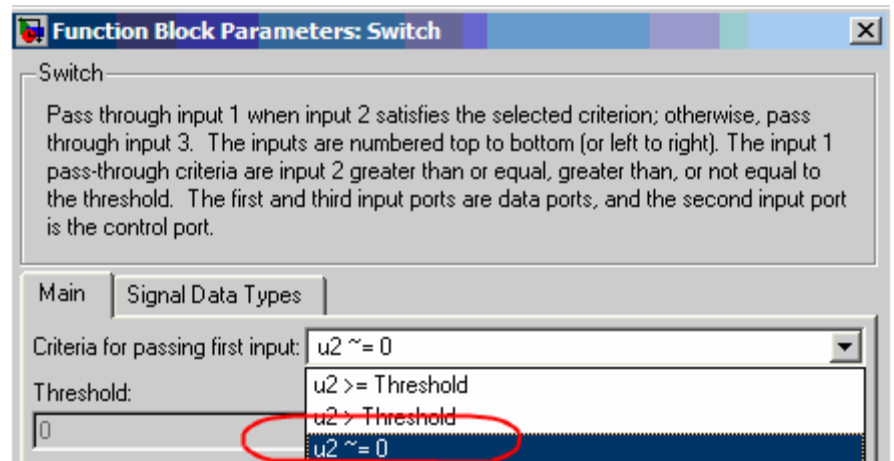
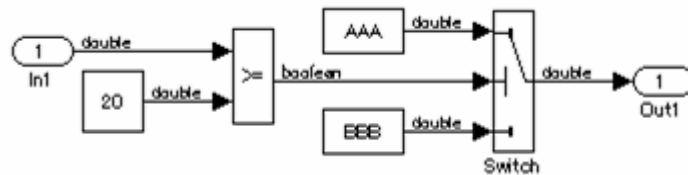
Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check scope of From and Goto blocks”

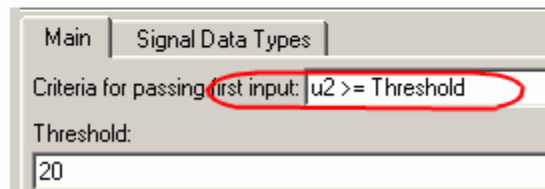
jc_0141: Use of the Switch block

ID: Title	jc_0141: Use of the Switch block
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<ul style="list-style-type: none">• The switch condition, input 2, must be a Boolean value.• The block parameter, Criteria for passing first input, should be set to $u2 \neq 0$.

jc_0141: Use of the Switch block



Correct



Incorrect

jc_0141: Use of the Switch block

Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check use of Switch blocks”

ID: Title jc_0121: Use of the Sum block

Priority Recommended

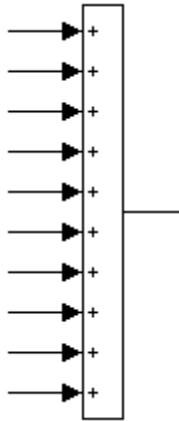
Scope MAAB

**MATLAB
Versions** All

Prerequisites None

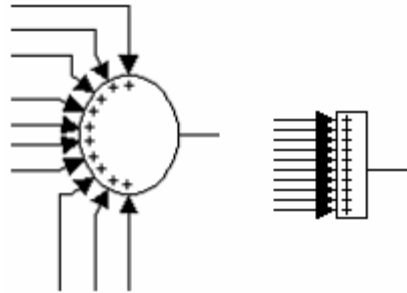
Description Sum blocks should:

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



Correct

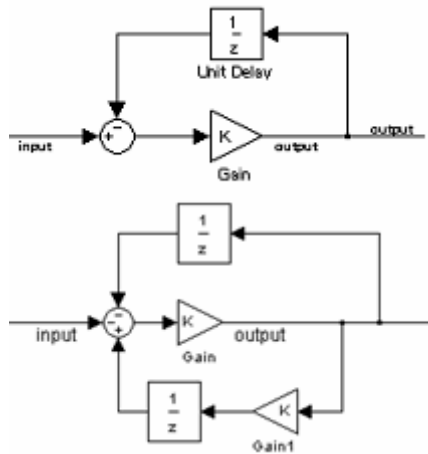
jc_0121: Use of the Sum block



Incorrect

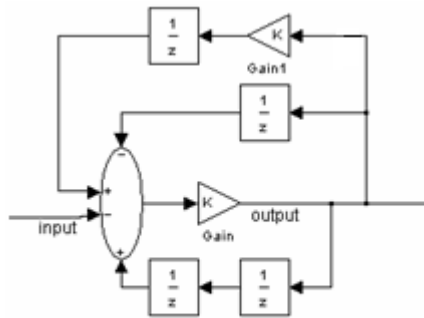
You may use the round shape in feedback loops.

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

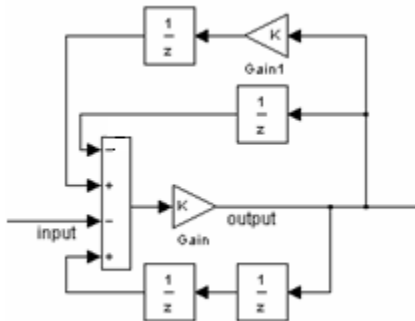


Correct

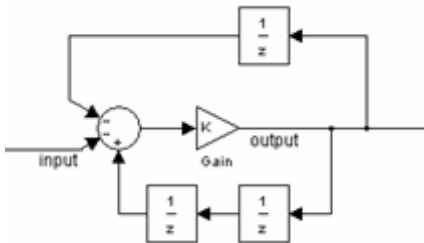
jc_0121: Use of the Sum block



Incorrect



Correct



Incorrect

jc_0121: Use of the Sum block

Rationale	Readability
Last Changed	V2.0
Model Advisor Check	Not applicable

jc_0131: Use of Relational Operator block

ID: Title jc_0131: Use of Relational Operator block

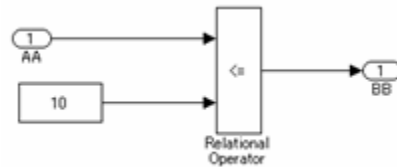
Priority Recommended

Scope J-MAAB

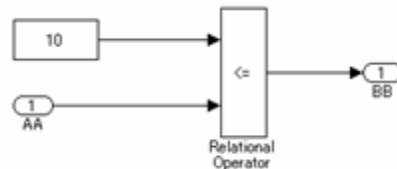
MATLAB Versions All

Prerequisites None

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



Correct



Incorrect

Rationale

- Readability

Last Changed V2.0

jc_0131: Use of Relational Operator block

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check configuration of Relational Operator blocks”

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 <ul style="list-style-type: none">• Prohibited in a data flow layer• Allowed between subsystems running at different rates
Rationale	<ul style="list-style-type: none">• Readability• Workflow
Last Changed	V2.0
Model Advisor Check	Not applicable

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.

ID: Title	db_0112: Indexing
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>Use a consistent vector indexing method for all blocks.</p> <p>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.</p>
See Also	<ul style="list-style-type: none">• “cgsl_0101: Zero-based indexing”• “hisl_0021: Consistent vector indexing method”
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Code Generation
Last Changed	V2.2
Model Advisor Check	By Task > Modeling Standards for MAAB > Simulink > “Check for indexing in blocks”

na_0010: Grouping data flows into signals

ID: Title na_0010: Grouping data flows into signals

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description **Vectors**

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

Busses

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

Examples

Some examples of vector signals include:

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

na_0010: Grouping data flows into signals

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

Some examples of bus signals include:

Bus type	Elements
Sensor Bus	Force Vector [Fx, Fy, Fz]
	Position
	Wheel Speed Vector [θ_{lf} , θ_{rf} , θ_{lr} , θ_{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 signal bus and Mux block usage”

db_0110: Tunable parameters in basic blocks

ID: Title	db_0110: Tunable parameters in basic blocks
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	To ensure that a parameter is tunable, enter it in a block dialog field:

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

tunable_parameter_value } tunable_parameter_vector } tunable_parameter_array }

Correct

tunable_parameter_value*2 } tunable_parameter_vector*3 } tunable_parameter_array*3 }
int16(tunable_parameter_value) } tunable_parameter_vector(2) } tunable_parameter_array(1,1) }

Incorrect

- | | |
|------------------|---|
| Rationale | <ul style="list-style-type: none">• Readability• Verification and Validation• Workflow• Code Generation• Simulation |
|------------------|---|

db_0110: Tunable parameters in basic blocks

Last Changed

V2.2

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > “Check use of tunable parameters in blocks”

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

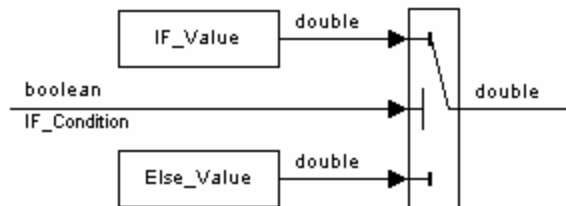
Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

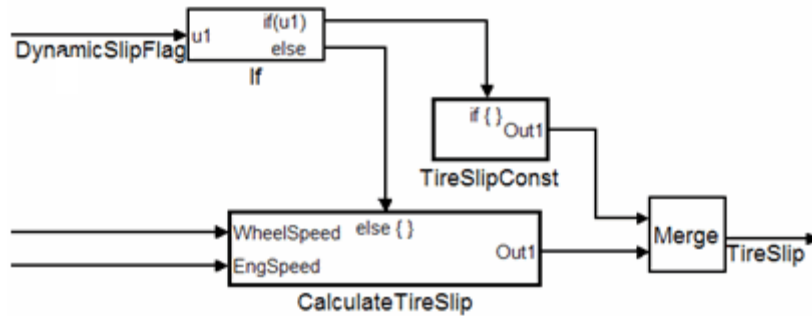
Description The **Switch** block should be used for modeling simple *if-then-else* structures, if the associated *then* and *else* actions involve only the assignment of constant values.



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.)

na_0012: Use of Switch vs. If-Then-Else Action Subsystem



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

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

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

Rationale

- Readability
- Verification and Validation
- Workflow

Last Changed

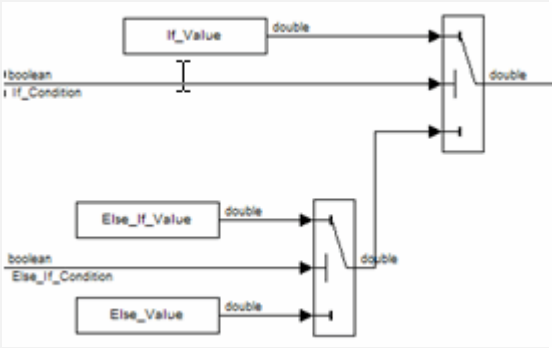
V2.0

Model Advisor Check

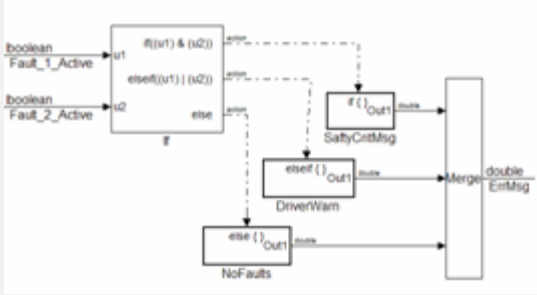
Not applicable

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
<pre>if then else if with blocks if (If_Condition) { output_signal = If_Value; } else if (Else_If_Condition) { output_signal = Else_If_Value; } else { output_signal = Else_Value; } </pre>	

db_0114: Simulink patterns for If-then-else-if constructs

Equivalent Functionality	Simulink Pattern
<p>if then else if with if/then/else subsystems</p> <pre> if(Fault_1_Active & 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

db_0115: Simulink patterns for case constructs

ID: Title db_0115: Simulink patterns for case constructs

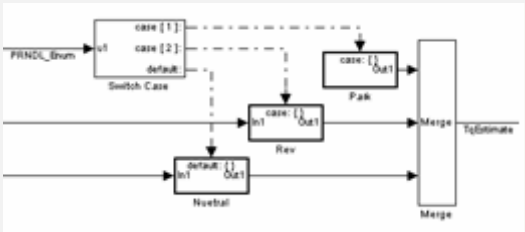
Priority Strongly recommended

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 a Simulink implementation of a switch statement. An input signal 'PRNDL_Enum' enters a 'Switch Case' block. This block has three cases: 'case [1]' labeled 'Park', 'case [2]' labeled 'Rev', and a 'default' case labeled 'Neutral'. Each case outputs to a corresponding 'case: [] Out1' block. The outputs from these three blocks are then fed into a 'Merge' block, which produces the final 'TqEstimate' output signal.</p>

Rationale • Readability

Last Changed V2.2

db_0115: Simulink patterns for case constructs

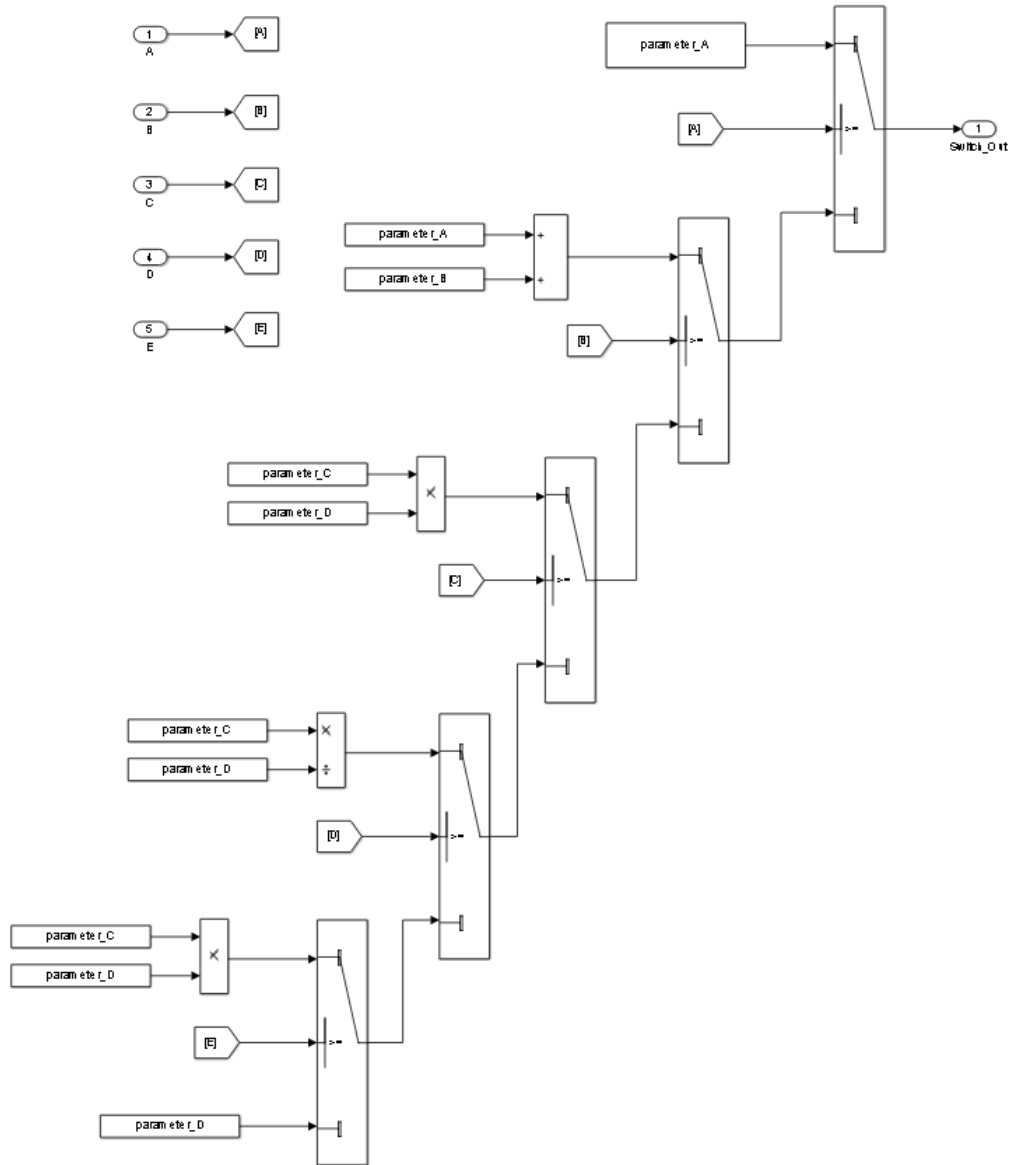
**Model
Advisor
Check**

Not applicable

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	<ul style="list-style-type: none">• na_0012: Use of Switch vs. If-Then-Else Action Subsystem• db_0144: Use of Subsystems
Description	<p>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.</p> <p>Incorrect</p>

na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches



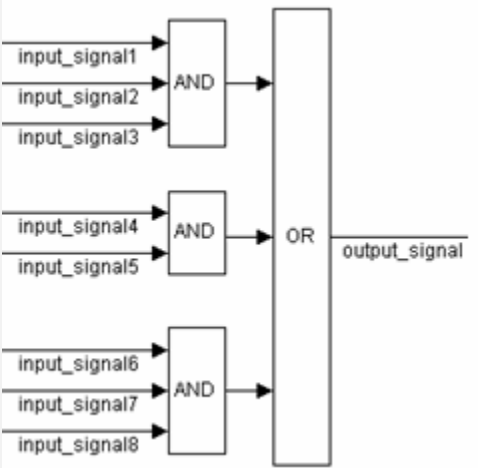
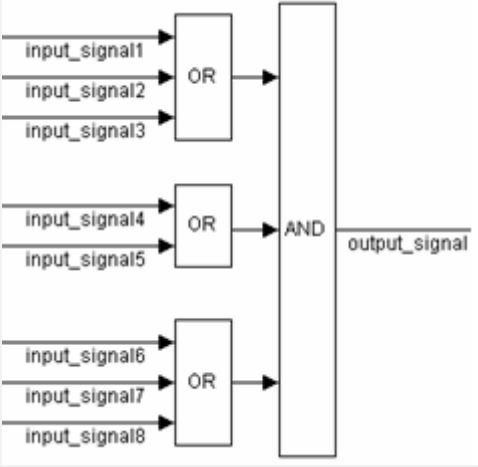
na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches

Rationale	<ul style="list-style-type: none">• Readability
See Also	bn_0003: Use of If-Then-Else Action Subsystem to Replace Multiple Switches
Last Changed	V3.0
Model Advisor Check	Not applicable

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:

db_0116: Simulink patterns for logical constructs with logical blocks

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

Rationale

- Readability

db_0116: Simulink patterns for logical constructs with logical blocks

**Last
Changed**

V1.0

**Model
Advisor
Check**

Not applicable

db_0117: Simulink patterns for vector signals

ID: Title db_0117: Simulink patterns for vector signals

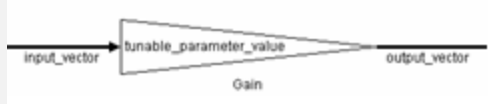

Priority Strongly recommended

Scope MAAB

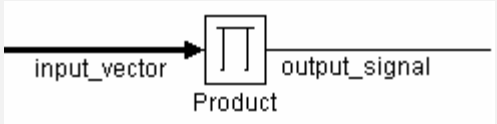
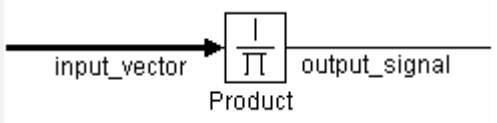
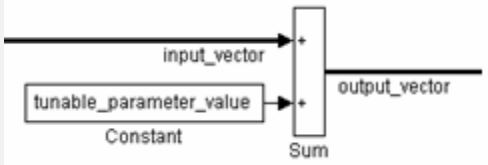
MATLAB Versions All

Prerequisites None

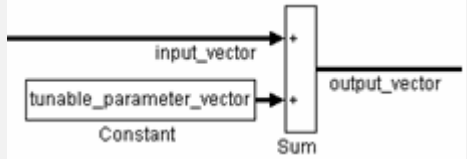
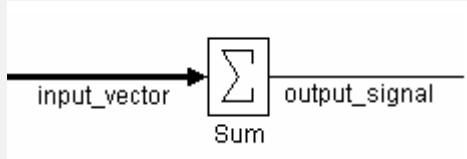
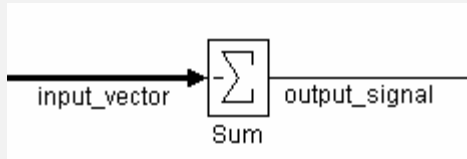
Description Simulink is a vectorizable modeling language allowing for the direct processing of vector data. Use the following patterns for vector signals within a Simulink model:

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre>for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_value; }</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_value' written inside and 'Gain' written below it. An arrow labeled 'output_vector' exits the block to the right.</p>
<p>Vector loop</p> <pre>for (i=0; i<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>

db_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i<input_vector_size; i++) { output_signal = output_signal * input_vector(i); } </pre>	 <p>The diagram shows a Simulink 'Product' block. An arrow labeled 'input_vector' enters the block from the left. An arrow labeled 'output_signal' exits the block to the right. The block is a square with a double Pi symbol (∏) inside.</p>
<p>Vector loop</p> <pre> output_signal = 1; for (i=0; i<input_vector_size; i++) { output_signal = output_signal / input_vector(i); } </pre>	 <p>The diagram shows a Simulink 'Product' block with a division symbol (∏ with a vertical line through it). An arrow labeled 'input_vector' enters the block from the left. An arrow labeled 'output_signal' exits the block to the right. The block is a square with a vertical line through a double Pi symbol.</p>
<p>Vector loop</p> <pre> for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_value; } </pre>	 <p>The diagram shows a Simulink 'Sum' block. Two arrows enter the block from the left: one labeled 'input_vector' and one labeled 'tunable_parameter_value' (with a box around it). An arrow labeled 'output_vector' exits the block to the right. The block is a vertical rectangle with two plus signs (+) on the left side and the word 'Sum' at the bottom.</p>

db_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
<p>Vector loop</p> <pre> for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_vector(i); } </pre>	 <p>The diagram shows a vertical summing junction block labeled 'Sum'. Two inputs enter from the left: 'input_vector' and 'tunable_parameter_vector' (labeled as 'Constant'). One output exits to the right, labeled 'output_vector'.</p>
<p>Vector loop:</p> <pre> output_signal = 0; for (i=0; i<input_vector_size; i++) { output_signal = output_signal + input_vector(i); } </pre>	 <p>The diagram shows a square summing junction block labeled 'Sum' with a summation symbol (Σ). One input enters from the left, labeled 'input_vector'. One output exits to the right, labeled 'output_signal'.</p>
<p>Vector loop:</p> <pre> output_signal = 0; for (i=0; i<input_vector_size; i++) { output_signal = output_signal - input_vector(i); } </pre>	 <p>The diagram shows a square summing junction block labeled 'Sum' with a summation symbol (Σ) and a minus sign (-). One input enters from the left, labeled 'input_vector'. One output exits to the right, labeled 'output_signal'.</p>

db_0117: Simulink patterns for vector signals

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

Rationale

- Readability
- Verification and Validation
- Code Generation

Last Changed

V2.2

db_0117: Simulink patterns for vector signals

**Model
Advisor
Check**

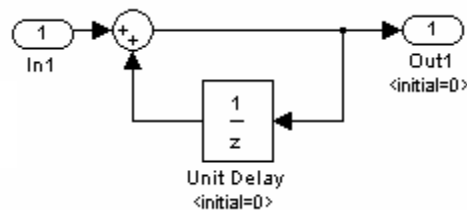
Not applicable

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

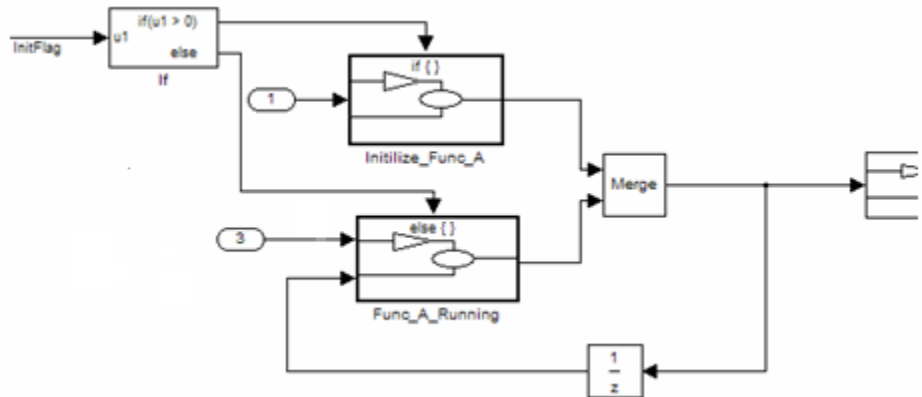
Initialization that Requires Computation

The following rules apply for complex initialization:

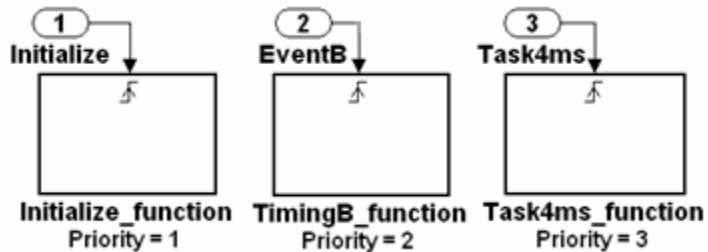
- The initialization should be performed in a separate subsystem.
- The initialization subsystem should have a name that indicates that initialization is performed by the subsystem.

jc_0351: Methods of initialization

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



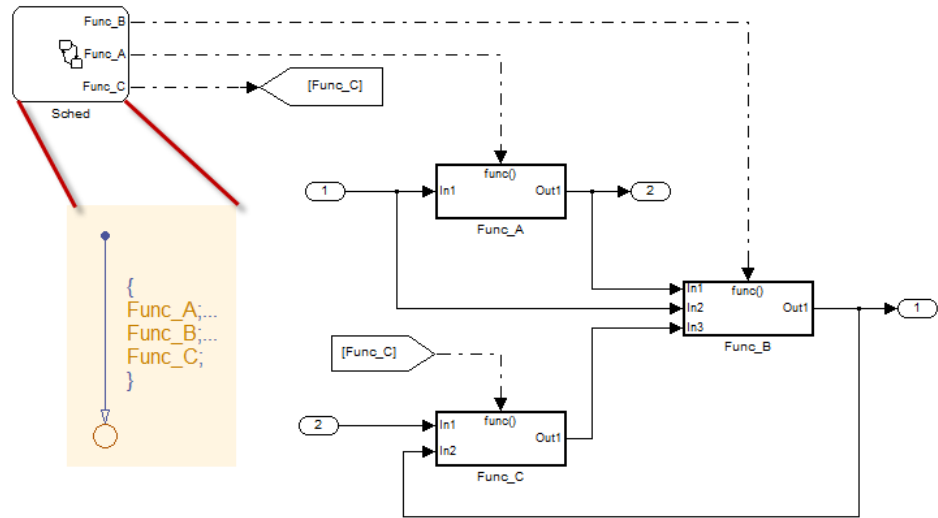
Example A



Example B

Or

jc_0351: Methods of initialization



Rationale

- Readability
- Code Generation

Last Changed

V2.2

Model Advisor Check

Not applicable

jc_0111: Direction of Subsystem

ID: Title jc_0111: Direction of Subsystem

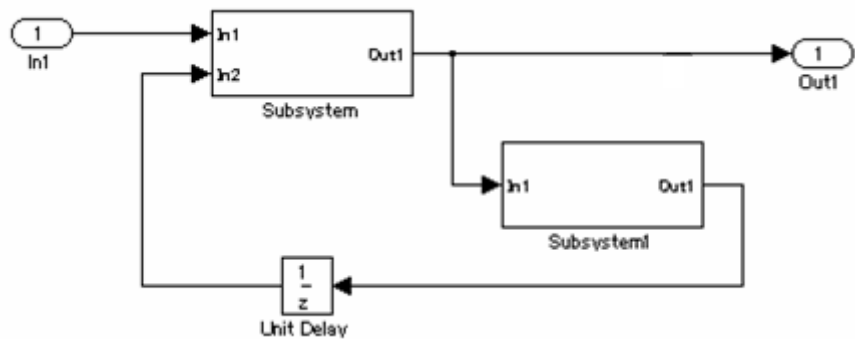
Priority Strongly recommended

Scope J-MAAB

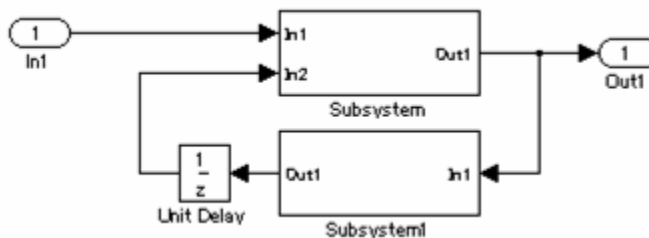
MATLAB Versions All

Prerequisites None

Description Subsystem must not be reversed.



Correct



Incorrect

jc_0111: Direction of Subsystem

Rationale

Readability

**Last
Changed**

V2.0

**Model
Advisor
Check**

By Task > Modeling Standards for MAAB > Simulink > “Check orientation of Subsystem blocks”

Stateflow

- “Chart Appearance” on page 7-2
- “Stateflow Data and Operations” on page 7-20
- “Events” on page 7-42
- “Statechart Patterns” on page 7-47
- “Flowchart Patterns” on page 7-53
- “State Chart Architecture” on page 7-69

Chart Appearance

- db_0123: Stateflow port names
- db_0129: Stateflow transition appearance
- db_0137: States in state machines
- db_0133: Use of patterns for Flowcharts
- db_0132: Transitions in Flowcharts
- 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	<p>The name of a Stateflow input or output should be the same as the corresponding signal.</p> <p>Exception: Reusable Stateflow blocks may have different port names.</p>
Rationale	<ul style="list-style-type: none">• Readability• Code Generation
Last Changed	V1.0
Model Advisor Check	By Task > Modeling Standards for MAAB > Stateflow > “Check for mismatches between names of Stateflow ports and associated signals”

db_0129: Stateflow transition appearance

ID: Title db_0129: Stateflow transition appearance

Priority Strongly recommended

Scope MAAB

**MATLAB
Versions** All

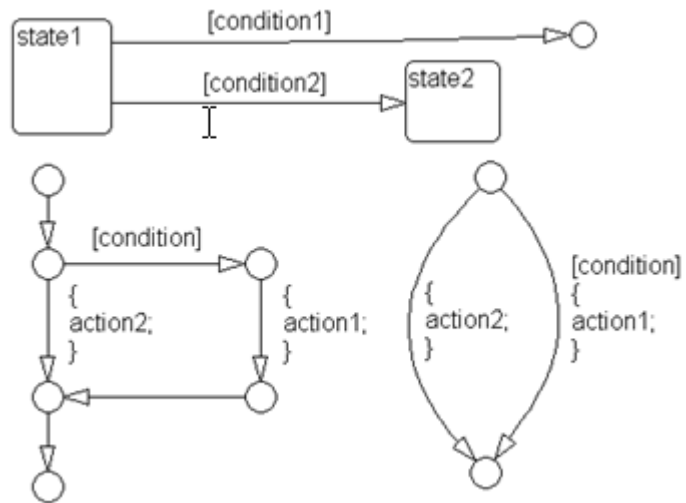
Prerequisites None

Description Transitions in Stateflow:

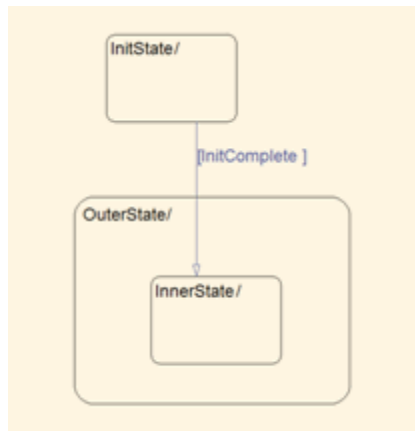
- Do not cross each other, if possible.
- Are not drawn one upon the other.
- Do not cross any states, junctions, or text fields.
- Allowed if transition is to an internal state.

Transition labels may be visually associated to the corresponding transition.

db_0129: Stateflow transition appearance

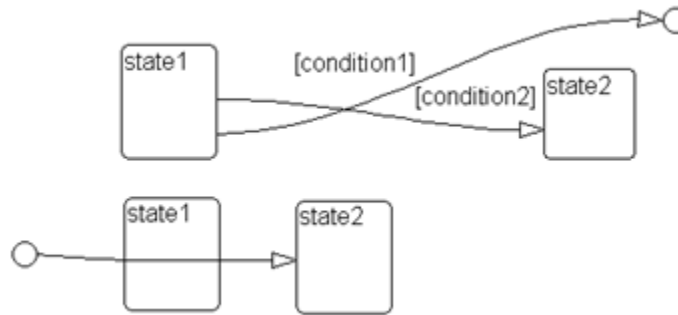


Correct



Correct: Transition crosses state boundary to connect to substate

db_0129: Stateflow transition appearance



Incorrect: Transitions cross each other and transition crosses through state

Rationale

- Readability

Last Changed

V2.2

Model Advisor Check

Not applicable

ID: Title	db_0137: States in state machines
Priority	Mandatory
Scope	MAAB
MATLAB Versions	All
Prerequisites	db_0149: Flowchart patterns for condition actions
Description	<p>For all levels in a state machine, including the root level, for states with exclusive decomposition the following rules apply:</p> <ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Readability• Workflow• Code Generation• Verification and Validation
Last Changed	V3.0
Model Advisor Check	By Task > Modeling Standards for MAAB > Sstateflow > “Check usage of exclusive and default states in state machines”

db_0133: Use of patterns for Flowcharts

ID: Title	db_0133: Use of patterns for Flowcharts
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>A Flowchart is built with the help of Flowchart patterns (for example, if-then-else, for loop, and so on):</p> <ul style="list-style-type: none">• The data flow is oriented from the top to the bottom.• Patterns are connected with empty transitions.
Rationale	<ul style="list-style-type: none">• Readability
Last Changed	V2.2
Model Advisor Check	Not applicable

ID: Title db_0132: Transitions in Flowcharts

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

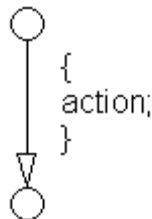
Prerequisites None

Description The following rules apply to transitions in Flowcharts:

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

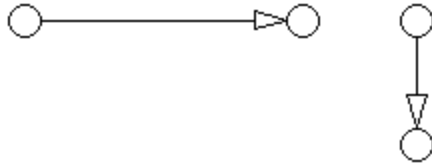


Transition with Condition



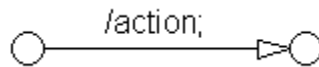
Transition with Condition Action

db_0132: Transitions in Flowcharts



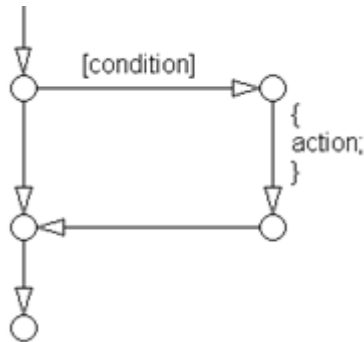
Empty Transition

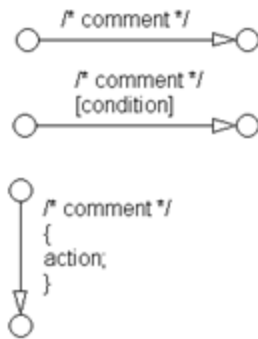
Transition actions are not used in Flowcharts. Transition actions are only valid when used in transitions between states in a state machine, otherwise they are not activated because of the inherent dependency on a valid state to state transition to activate them.



Transition Action

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





Transitions with Comments

Rationale

- Readability

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > “Check Transition orientations in flowcharts”

jc_0501: Format of entries in a State block

ID: Title jc_0501: Format of entries in a State block

Priority Recommended

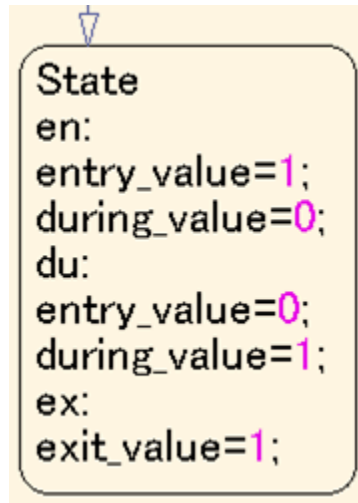
Scope MAAB

MATLAB Versions All

Prerequisites None

Description A new line should:

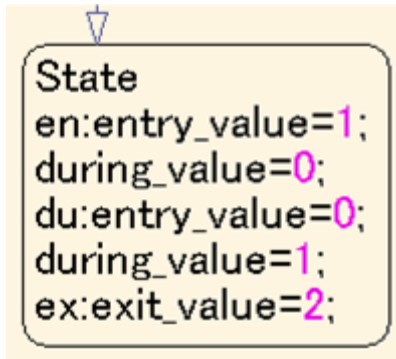
- Start after the entry (en), during (du), and exit (ex) statements.
- Start after the completion of an assignment statement “;”.



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

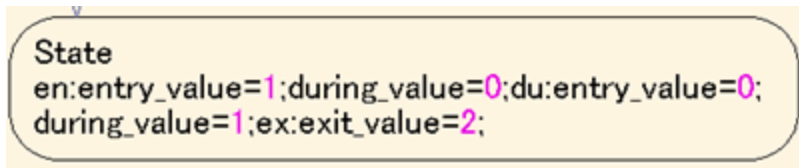
Correct

jc_0501: Format of entries in a State block



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

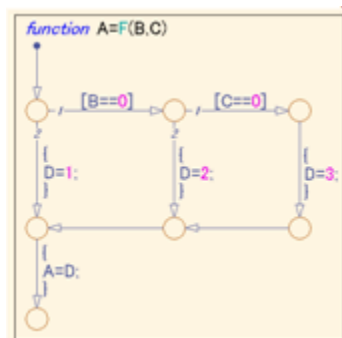
V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > “Check entry formatting in State blocks in Stateflow charts”

jc_0511: Setting the return value from a graphical function

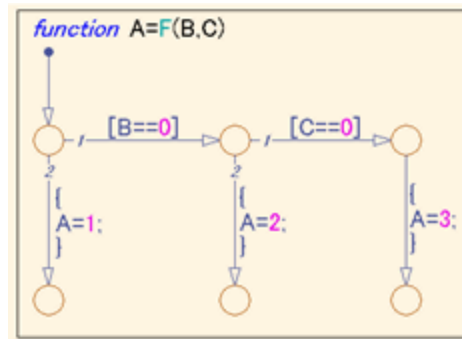
ID: Title	jc_0511: Setting the return value from a graphical function
Priority	Mandatory
Scope	J-MAAB
MATLAB Versions	All
Prerequisites	None
Description	The return value from a graphical function must be set in only one place.



Correct

Return value A is set in one place.

jc_0511: Setting the return value from a graphical function



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 return value assignments of graphical functions in Stateflow charts”

jc_0531: Placement of the default transition

ID: Title jc_0531: Placement of the default transition

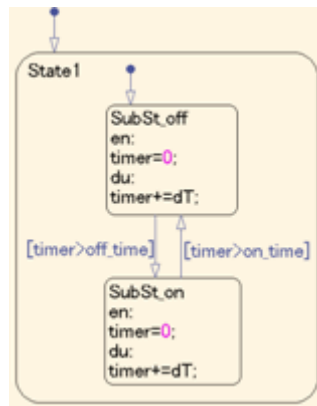
Priority Recommended

Scope J-MAAB

MATLAB Versions All

Prerequisites None

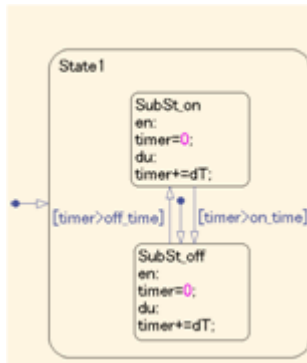
- Description**
- Default transition is connected at the top of the state.
 - The destination state of the default transition is put above the other states in the same hierarchy.



Correct

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

jc_0531: Placement of the default transition



Incorrect

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

Rationale

Readability

Last Changed

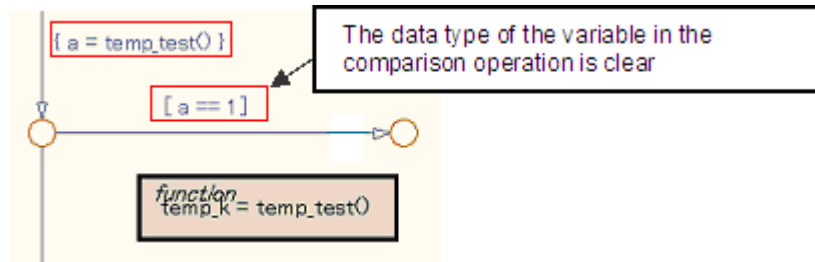
V2.0

Model Advisor Check

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

jc_0521: Use of the return value from graphical functions

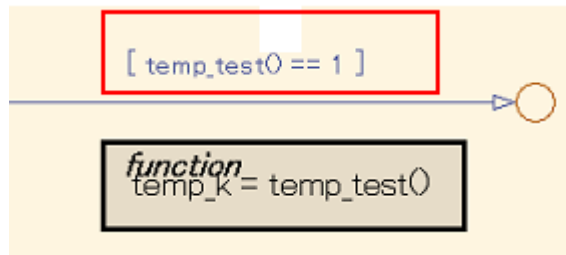
ID: Title	jc_0521: Use of the return value from graphical functions
Priority	Recommended
Scope	J-MAAB
MATLAB Versions	All
Prerequisites	None
Description	The return value from a graphical function should not be used directly in a comparison operation.



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`.

jc_0521: Use of the return value from graphical functions



Incorrect

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

Rationale

- Readability
- 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”

jc_0521: Use of the return value from graphical functions

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

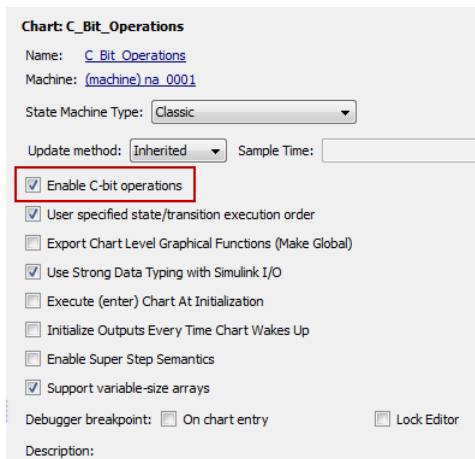
Prerequisites None

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**.

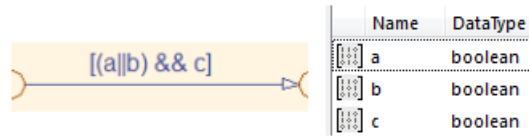
2 Select **Enable C-bit operations**.



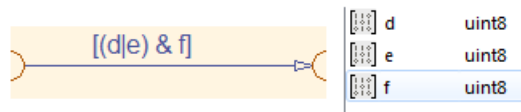
Correct

Use && and || for Boolean operation.

na_0001: Bitwise Stateflow operators

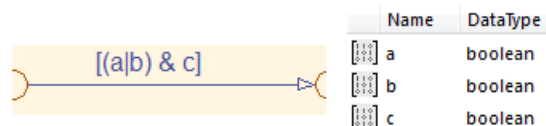


Use & and | for bit operation.



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”

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”

na_0013: Comparison operation in Stateflow

ID: Title na_0013: Comparison operation in Stateflow

Priority Recommended

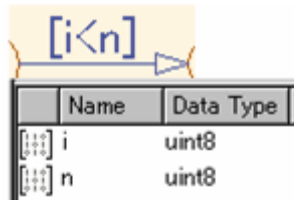
Scope MAAB

MATLAB Versions All

Prerequisites None

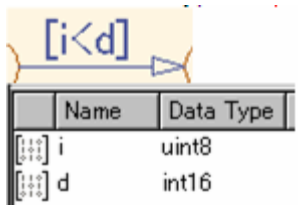
Description

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



Correct

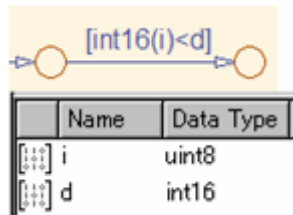
Same data type in “i” and “n”



Incorrect

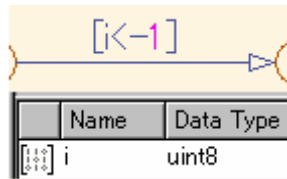
Different data type in “i” and “d”

na_0013: Comparison operation in Stateflow



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”

db_0122: Stateflow and Simulink interface signals and parameters

ID: Title db_0122: Stateflow and Simulink interface signals and parameters

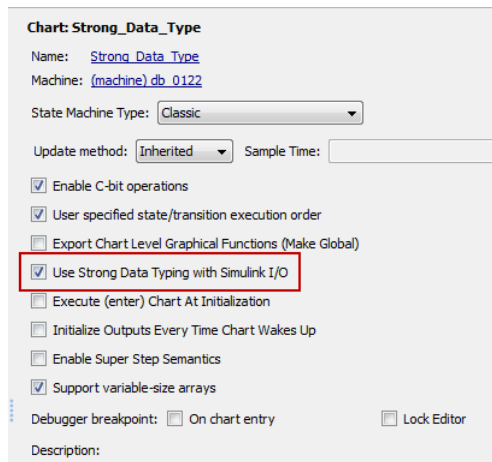
Priority Strongly recommended

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

db_0122: Stateflow and Simulink interface signals and parameters

**Last
Changed**

V2.0

**Model
Advisor
Check**

By Task > Modeling Standards for MAAB > Stateflow > “Check for Strong Data Typing with Simulink I/O”

db_0125: Scope of internal signals and local auxiliary variables

ID: Title db_0125: Scope of internal signals and local auxiliary variables

Priority Strongly recommended

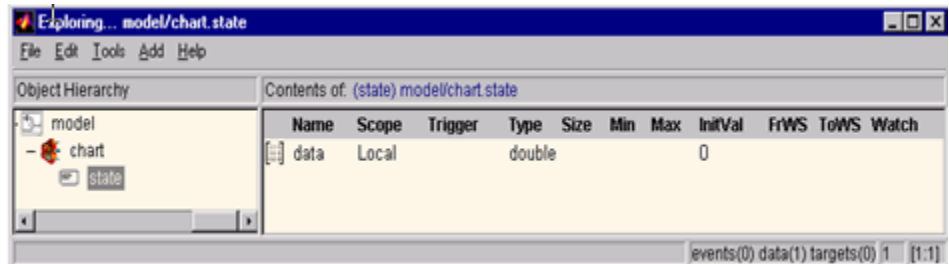
Scope MAAB

MATLAB Versions All

Prerequisites None

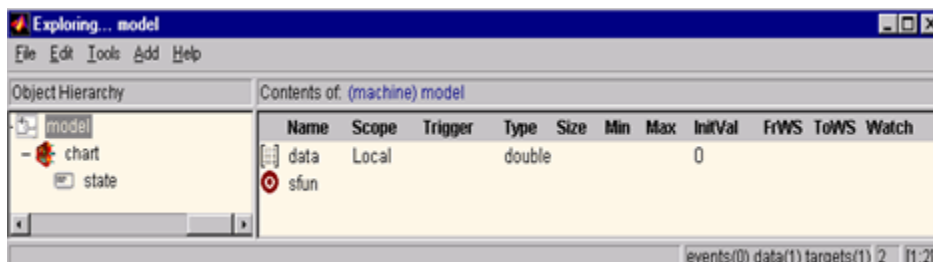
Description Internal signals and local auxiliary variables are "Local data" in Stateflow:

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



Correct

db_0125: Scope of internal signals and local auxiliary variables



Incorrect

Rationale

- Readability
- Code Generation

Last Changed

V2.0

Model Advisor Check

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

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

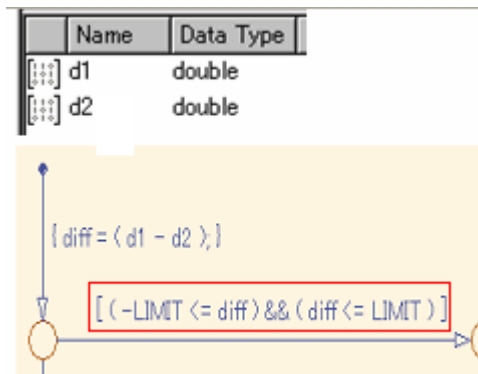
Scope MAAB

MATLAB Versions All

Prerequisites None

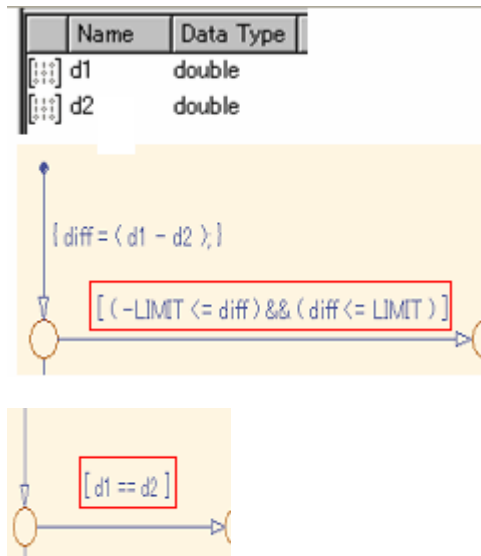
Description

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



Correct

jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow



Incorrect

Rationale

- Verification and Validation
- Code Generation

Last Changed

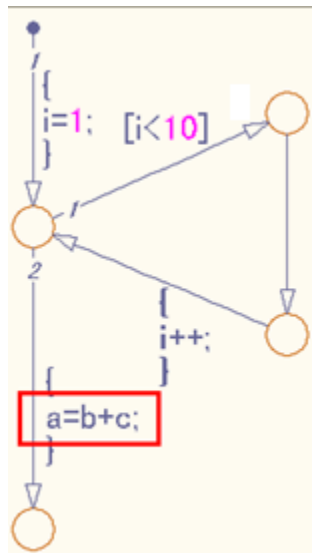
V2.0

Model Advisor Check

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

jc_0491: Reuse of variables within a single Stateflow scope

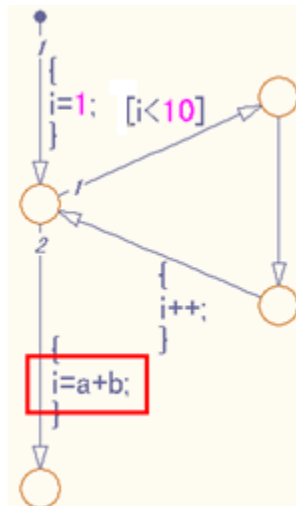
ID: Title	jc_0491: Reuse of variables within a single Stateflow scope
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	The same variable should not have multiple meanings (usages) within a single Stateflow state.



Correct

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

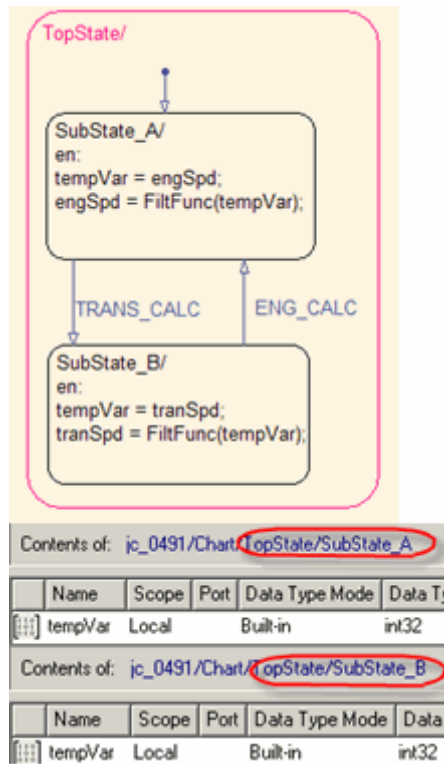
jc_0491: Reuse of variables within a single Stateflow scope



Incorrect

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

jc_0491: Reuse of variables within a single Stateflow scope



Correct

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

Rationale

- Readability
- Verification
- Code Generation

Last Changed

V2.2

jc_0491: Reuse of variables within a single Stateflow scope

**Model
Advisor
Check**

Not applicable

jc_0541: Use of tunable parameters in Stateflow

ID: Title jc_0541: Use of tunable parameters in Stateflow

Priority Strongly recommended

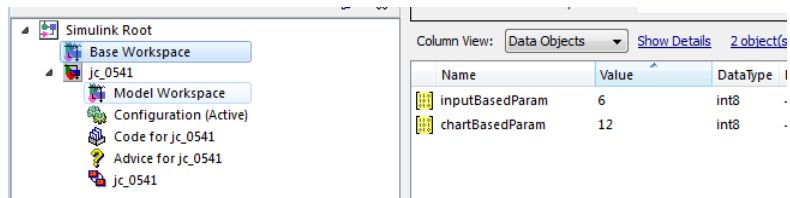
Scope MAAB

MATLAB Versions All

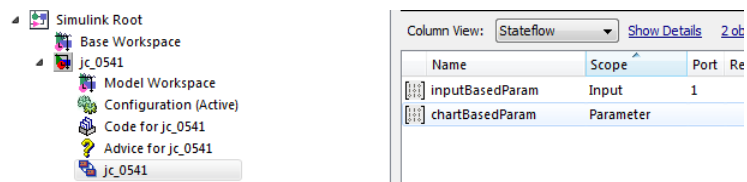
Prerequisites None

Description 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.

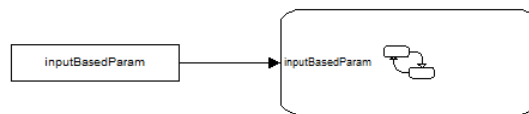


Base Workspace Definitions



Stateflow® Chart Definitions

jc_0541: Use of tunable parameters in Stateflow



Stateflow® Chart

Rationale

- Verification
- Code Generation

Last Changed

V2.2

Model Advisor Check

Not applicable

db_0127: MATLAB commands in Stateflow

ID: Title db_0127: MATLAB commands in Stateflow

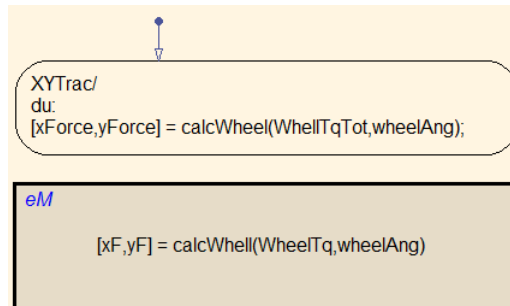
Priority Mandatory

Scope MAAB

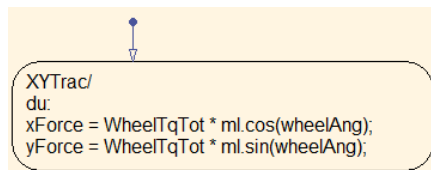
MATLAB Versions All

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

db_0127: MATLAB commands in Stateflow

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”

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

jm_0011: Pointers in Stateflow

Events

- db_0126: Scope of events
- jm_0012: Event broadcasts

ID: Title	db_0126: Scope of events
Priority	Mandatory
Scope	MAAB
MATLAB Versions	Pre R2009b
Prerequisites	None
Description	<p>The following rules apply to events in Stateflow:</p> <ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Readability• Verification and Validation• Workflow• Code Generation• Verification and Validation
Last Changed	V2.2
Model Advisor Check	Not applicable

jm_0012: Event broadcasts

ID: Title jm_0012: Event broadcasts

Priority Strongly recommended

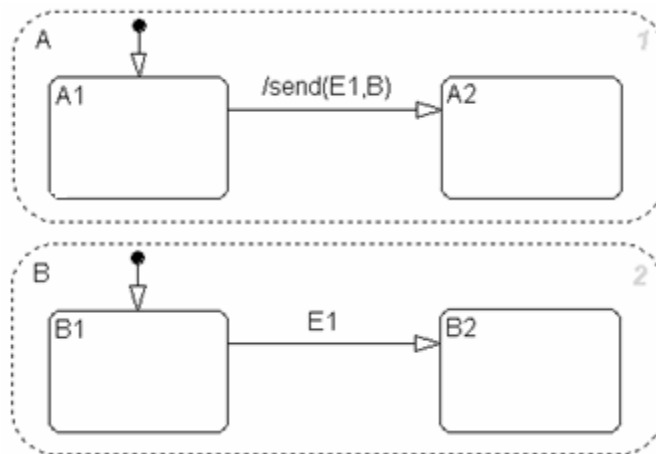
Scope MAAB

MATLAB Versions All

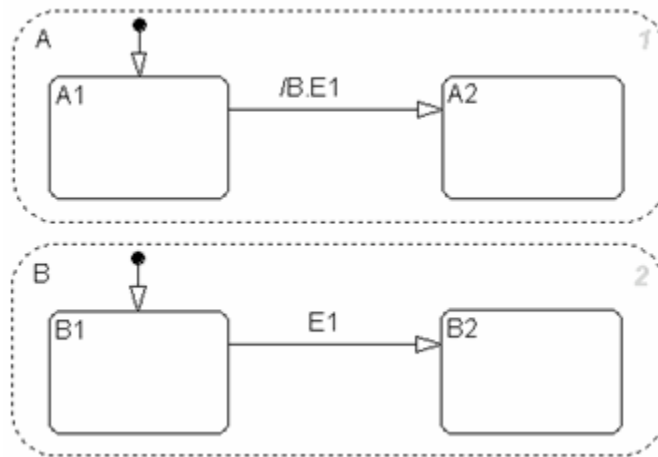
Prerequisites db_0126: Scope of events

Description The following rules apply to event broadcasts in Stateflow:

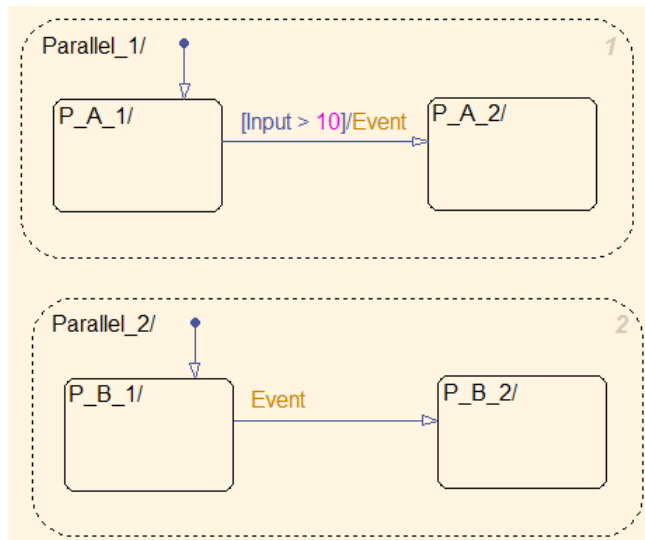
- Directed event broadcasts are the only type of event broadcasts allowed.
- The send syntax or qualified event names are used to direct the event to a particular state.
- Multiple send statements should be used to direct an event to more than one state.



Correct: Example Using Send Syntax



Correct: Example Using Qualified Event Names



Incorrect: Use of a non-directed event

Rationale

- Readability

jm_0012: Event broadcasts

- 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”

Statechart Patterns

- db_0150: State machine patterns for conditions
- db_0151: State machine patterns for transition actions

db_0150: State machine patterns for conditions

ID: Title db_0150: State machine patterns for conditions

Priority Strongly recommended

Scope MAAB

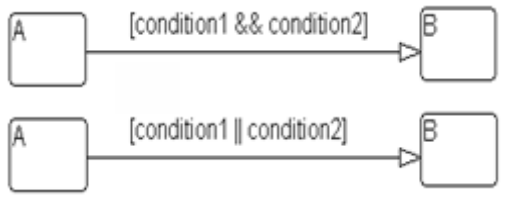
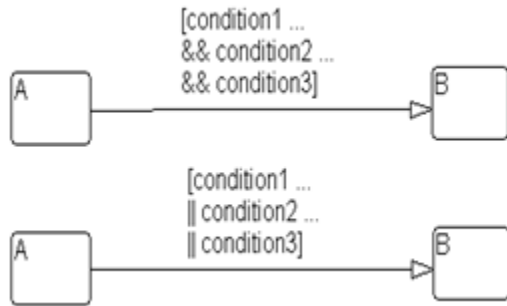
MATLAB Versions All

Prerequisites None

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

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

db_0150: State machine patterns for conditions

Equivalent Functionality	State Machine Pattern
<p>Up to three conditions, short form:</p> <p>(The use of different logical operators in this form is not allowed. Use subconditions instead.)</p> <pre>(condition1 && 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 ... && condition2 ... && condition3) (condition1 ... condition2 ... condition3)</pre>	

Rationale

- Readability

db_0150: State machine patterns for conditions

**Last
Changed**

V2.2

**Model
Advisor
Check**

Not applicable

db_0151: State machine patterns for transition actions

ID: Title db_0151: State machine patterns for transition actions

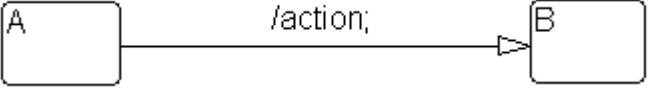
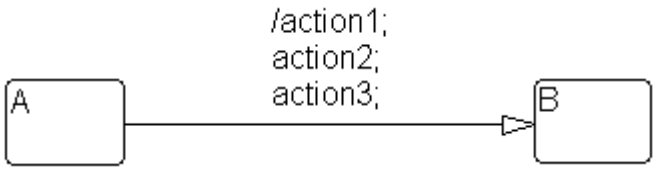
Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

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

Equivalent Functionality	State Machine Pattern
One transition action: <code>action;</code>	 <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 <code>/action;</code>. The arrow ends in a triangular arrowhead pointing towards box B.</p>
Two or more transition actions, multiline form: (Two or more transition actions in one line are not allowed.) <code>action1;</code> <code>action2;</code> <code>action3;</code>	 <p>The diagram shows two state boxes, A and B. A horizontal arrow points from box A to box B. Above the arrow are three lines of text: <code>/action1;</code>, <code>action2;</code>, and <code>action3;</code>. The arrow ends in a triangular arrowhead pointing towards box B.</p>

db_0151: State machine patterns for transition actions

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”

Flowchart Patterns

- db_0148: Flowchart patterns for conditions
- db_0149: Flowchart patterns for condition actions
- db_0134: Flowchart patterns for If constructs
- db_0159: Flowchart patterns for case constructs
- db_0135: Flowchart 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: Flowchart patterns for conditions

ID: Title db_0148: Flowchart patterns for conditions

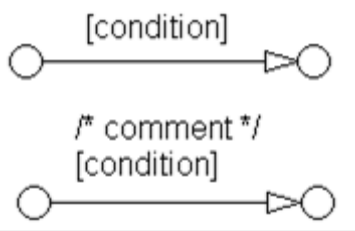
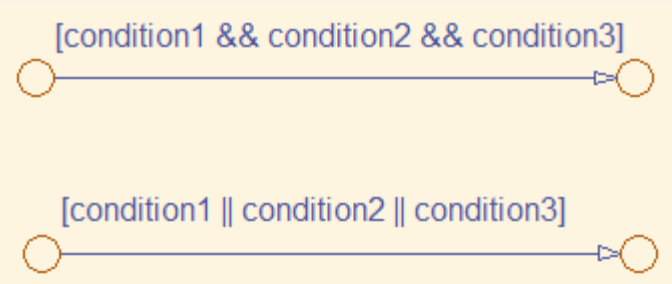
Priority Strongly recommended

Scope MAAB

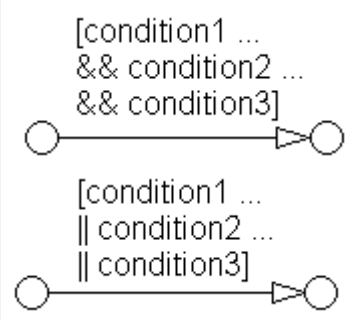
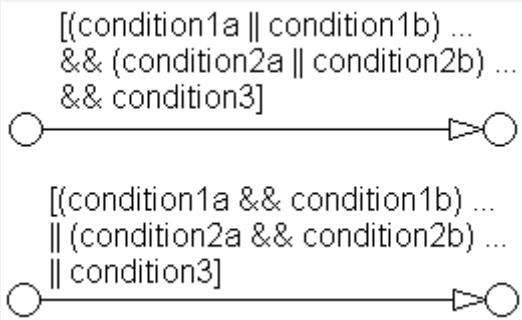
MATLAB Versions All

Prerequisites None

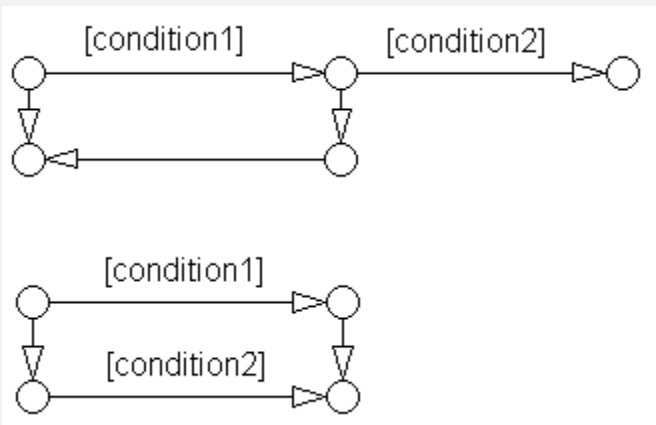
Description Use the following patterns for conditions within Stateflow Flowcharts:

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

db_0148: Flowchart patterns for conditions

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

db_0148: Flowchart patterns for conditions

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

Rationale

- Readability

Last Changed

V2.2

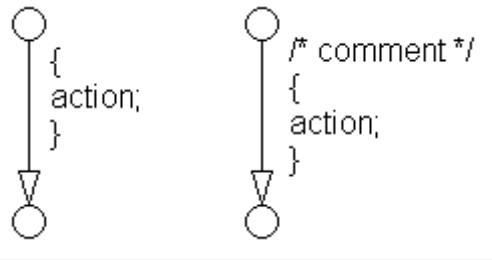
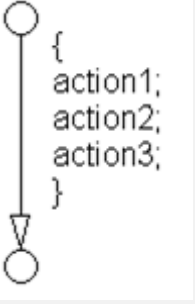

Model Advisor Check

Not applicable

db_0149: Flowchart patterns for condition actions

ID: Title	db_0149: Flowchart 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 Flowcharts:

db_0149: Flowchart patterns for condition actions

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

db_0149: Flowchart patterns for condition actions

Rationale	<ul style="list-style-type: none">• Readability
Last Changed	V2.2
Model Advisor Check	Not applicable

db_0134: Flowchart patterns for If constructs

ID: Title db_0134: Flowchart patterns for If constructs

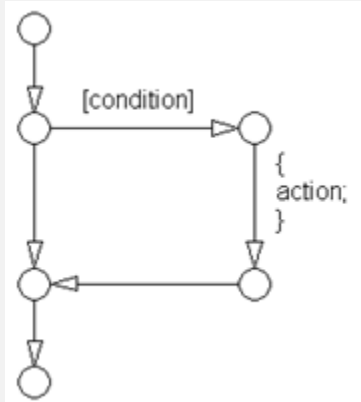
Priority Strongly recommended

Scope MAAB

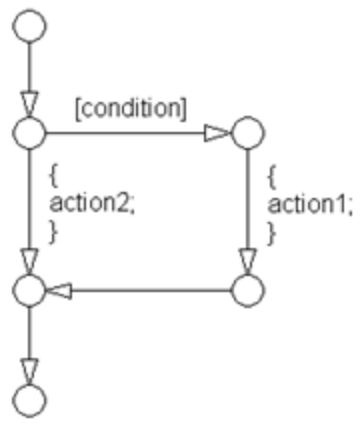
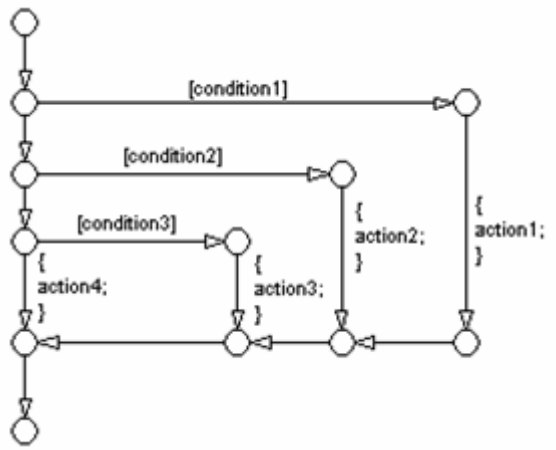
MATLAB Versions All

Prerequisites db_0148: Flowchart patterns for conditions
db_0149: Flowchart patterns for condition actions

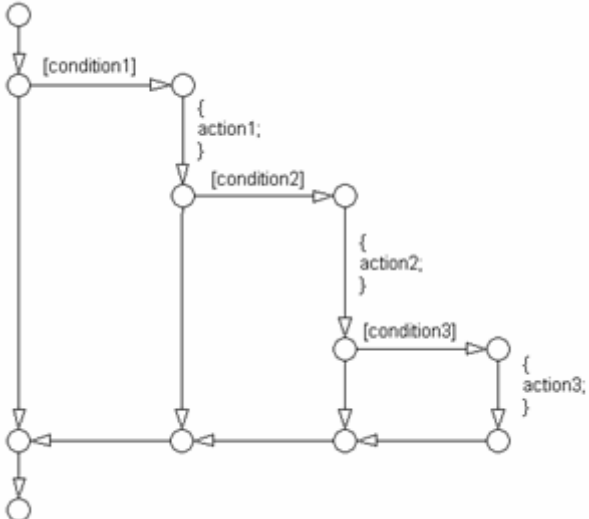
Description Use the following patterns for If constructs within Stateflow Flowcharts:

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

db_0134: Flowchart patterns for If constructs

Equivalent Functionality	Flowchart Pattern
<pre> if then else if (condition){ action1; } else { action2; } </pre>	 <p>The flowchart starts with a start node (circle) leading to a decision node (inverted triangle). From the decision node, a path labeled '[condition]' leads to a right-side action node (circle) containing '{ action1; }'. A return path leads to a left-side action node (circle) containing '{ action2; }'. Both action nodes lead to a common merge node (inverted triangle), which then leads to the final end node (circle).</p>
<pre> if then else if if (condition1){ action1; } else if (condition2) { action2; } else if (condition3){ __action3; } else { action4; } </pre>	 <p>The flowchart starts with a start node (circle) leading to a decision node (inverted triangle). From this node, a path labeled '[condition1]' leads to a right-side action node (circle) containing '{ action1; }'. A return path leads to a second decision node (inverted triangle). From this second node, a path labeled '[condition2]' leads to a right-side action node (circle) containing '{ action2; }'. A return path leads to a third decision node (inverted triangle). From this third node, a path labeled '[condition3]' leads to a right-side action node (circle) containing '{ action3; }'. A return path leads to a fourth decision node (inverted triangle). From this fourth node, a path labeled '{ action4; }' leads to a right-side action node (circle) containing '{ action4; }'. A return path leads to a common merge node (inverted triangle), which then leads to the final end node (circle).</p>

db_0134: Flowchart patterns for If constructs

Equivalent Functionality	Flowchart Pattern
<p>Cascade of if then</p> <pre>if (condition1){ action1; if (condition2){ action2; if (condition3){ action3; } } }</pre>	

Rationale

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

Last Changed

V1.0

Model Advisor Check

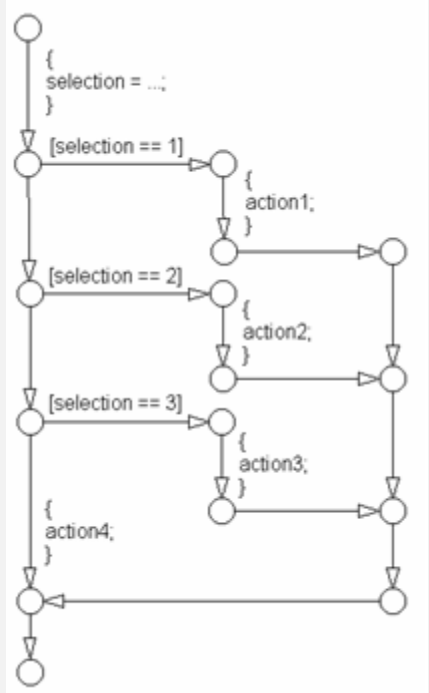
Not applicable

db_0159: Flowchart patterns for case constructs

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

db_0159: Flowchart patterns for case constructs

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

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

db_0159: Flowchart patterns for case constructs

Equivalent Functionality	Flowchart Pattern
<p>case with exclusive conditions</p> <pre> c1 = condition1; c2 = condition2; c3 = condition3; if (c1 && !c2 && !c3) { action1; } elseif (!c1 && c2 && !c3) { action2; } elseif (!c1 && !c2 && c3) { action3; } else { action4; } </pre>	

Rationale • Readability

Last Changed V1.0

Model Advisor Check Not applicable

db_0135: Flowchart patterns for loop constructs

ID: Title	db_0135: Flowchart patterns for loop constructs
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	db_0148: Flowchart patterns for conditions db_0149: Flowchart patterns for condition actions

db_0135: Flowchart patterns for loop constructs

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

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

db_0135: Flowchart patterns for loop constructs

Rationale

- Readability

**Last
Changed**

V1.0

**Model
Advisor
Check**

Not applicable

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

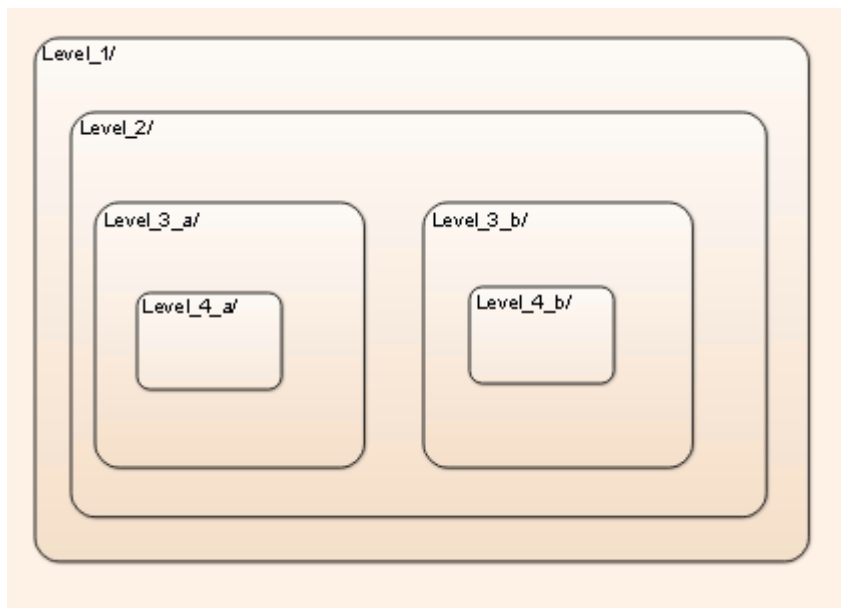
Priority Recommended

Scope NA-MAAB

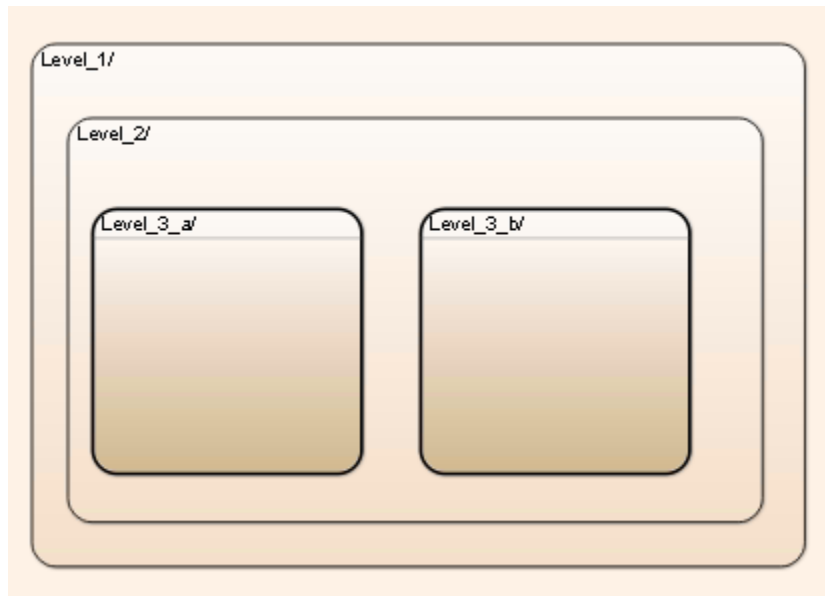
MATLAB Versions All

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

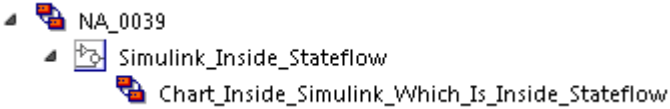
Last Changed

V3.0

Model Advisor Check

Not applicable

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	<p>Do not nest Stateflow charts inside Simulink functions that are included in Stateflow charts.</p> <p>Incorrect</p>  <pre>NA_0039 ├── Simulink_Inside_Stateflow │ └── Chart_Inside_Simulink_Which_Is_Inside_Stateflow</pre>
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Code Generation
Last Changed	V3.0
Model Advisor Check	Not applicable

na_0040: Number of states per container

ID: Title na_0040: Number of states per container

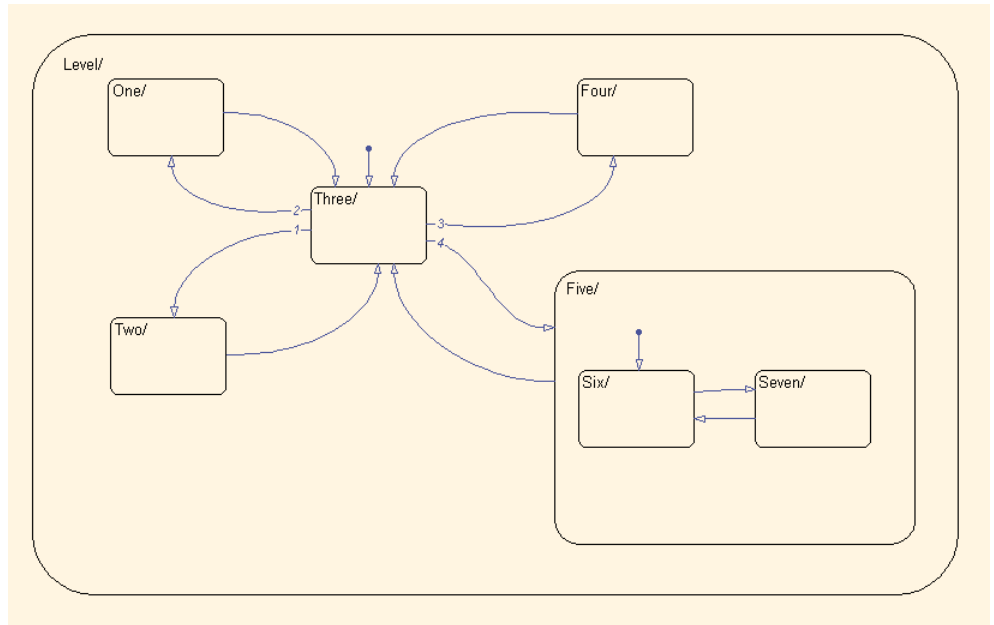
Priority Recommended

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.

na_0040: Number of states per container

Rationale

- Readability
- Verification and Validation
- Code Generation

Last Changed

V3.0

Model Advisor Check

Not applicable

na_0041: Selection of function type

ID: Title	na_0041: Selection of function type
Priority	Recommended
Scope	NA-MAAB
MATLAB Versions	All
Description	<p>Stateflow supports three types of functions: Graphical, MATLAB and Simulink. The appropriate function depends on the type of operations required:</p> <ul style="list-style-type: none">• Simulink<ul style="list-style-type: none">▪ Transfer functions▪ Integrators▪ Table look-ups• MATLAB<ul style="list-style-type: none">▪ Complex equations▪ If / then / else logic• Graphical functions<ul style="list-style-type: none">▪ If / then / else logic
Rationale	<ul style="list-style-type: none">• Workflow• Code Generation
Last Changed	V3.0

na_0041: Selection of function type

**Model
Advisor
Check**

Not applicable

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	<p>When deciding whether to embed Simulink functions inside a Stateflow chart, the following conditions make embedding the preferred option. If the Simulink functions</p> <ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Readability• Workflow
Last Changed	V3.0
Model Advisor Check	Not applicable

na_0042: Location of Simulink functions

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	<p>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.</p>
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Workflow• Code Generation• Simulation
See Also	<ul style="list-style-type: none">• dm_0002: Enumerated type usage
Last Changed	V3.0

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	<ul style="list-style-type: none">• Readability• Verification and Validation• Code Generation
Last Changed	V3.0

MATLAB Functions

- “MATLAB Function Appearance” on page 9-2
- “MATLAB Function Data and Operations” on page 9-7
- “MATLAB Function Patterns” on page 9-11
- “MATLAB Function Usage” on page 9-14

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	<ul style="list-style-type: none">• Readability• Code Generation
See Also	<ul style="list-style-type: none">• jr_0002: Number of nested if/else and case statement blocks
Last Changed	V3.0

na_0019: Restricted Variable Names

ID: Title	na_0019: Restricted Variable Names
Priority	Mandatory
Scope	NA-MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>To improve the readability of the MATLAB code, avoid using reserved C variable names. For example, avoid using <code>const</code>, <code>const</code>, <code>TRUE</code>, <code>FALSE</code>, <code>infinity</code>, <code>nil</code>, <code>double</code>, <code>single</code>, or <code>enum</code>.</p> <p>Avoid using variable names that conflict with MATLAB Functions, for example <code>conv</code>.</p>
Note	Reserved keywords are defined in the Simulink Coder™ documentation.
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation
See Also	<ul style="list-style-type: none">• Derived from jh_0042: Required software
Last Changed	V3.0

ID: Title	na_0025: MATLAB Function Header
Priority	Strongly recommended
Scope	NA-MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>MATLAB Functions must have a descriptive header. Header content may include, but is not limited to, the following types of information:</p>

- 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
%
% $Revision: 1.1.6.2.2.1 $
```

na_0025: MATLAB Function Header

```
% $Author: batserve $  
% $Date: August 27, 2012  
% _____
```

Rationale

- Readability
- Verification and Validation
- Workflow
- Code Generation

See Also

- jh_0073: eML Header version

Last Changed

V3.0

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 <code>assert</code> statements.
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Workflow• Code Generation
Last Changed	V3.0

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)
```

na_0024: Global Variables

```
        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

- ek_0003: Global Variables

Last Changed

V3.0

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
```

na_0022: Recommended patterns for Switch/Case statements

```
    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

- [jh_0026](#): Switch / Case statement

Last Changed

V3.0

na_0022: Recommended patterns for Switch/Case statements

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	<p>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.</p> <p>If sub-functions are used, they may use additional lines of code. Also limit the length of sub-functions to 60 lines of code.</p>
Rationale	<ul style="list-style-type: none">• Readability• Verification and Validation• Workflow• Code Generation
See Also	<ul style="list-style-type: none">• IM_0008: Source lines of eML
Last Changed	V3.0

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	<p>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.</p> <p>This includes functions that are defined within the MATLAB block and those in the separate .m files.</p>
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	<ul style="list-style-type: none">• Readability• Verification and Validation
Last Changed	V3.0

ID: Title	na_0021: Strings
Priority	Strongly recommended
Scope	NA-MAAB
MATLAB Versions	All
Prerequisites	None
Description	<p>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.</p> <p>For example, the following code will produce an error:</p> <pre>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.</pre>
Note	If the string is being used for switch / case behavior, consider using enumerated data types
Rationale	<ul style="list-style-type: none">• Verification and Validation• Workflow• Code Generation
See Also	<ul style="list-style-type: none">• jh_0024: Strings
Last Changed	V3.0

na_0021: Strings

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

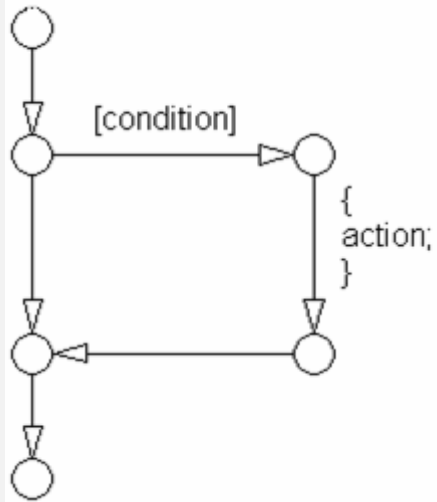
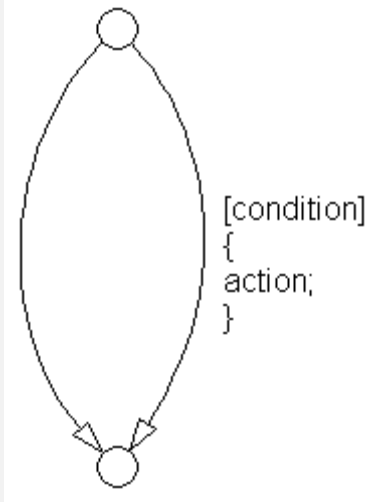
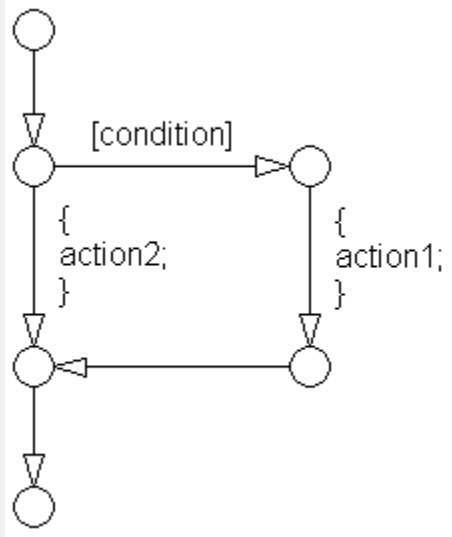
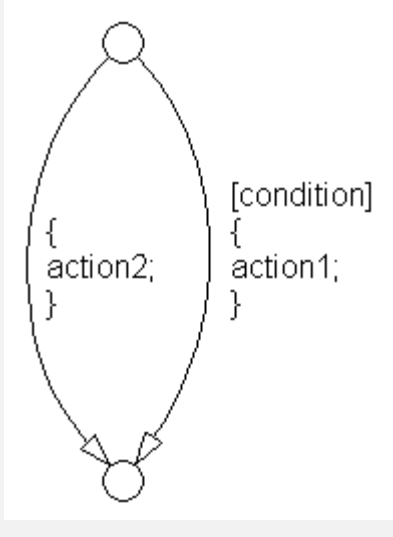
Guidelines with the following characteristics are easier to understand and use. At a minimum, when writing a new guideline, it should be

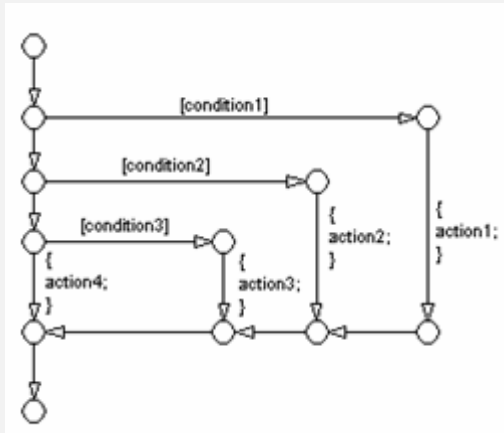
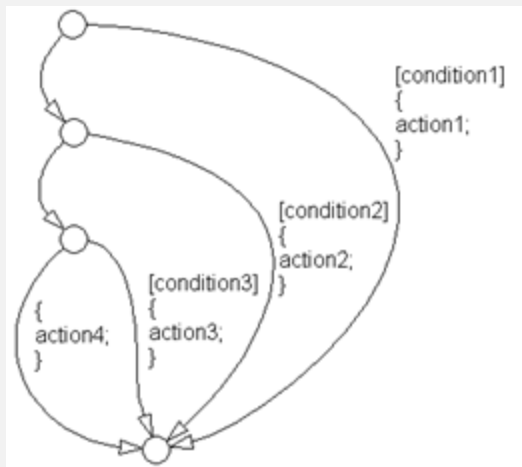
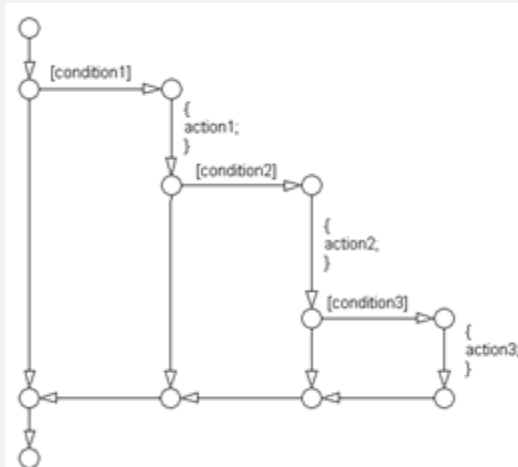
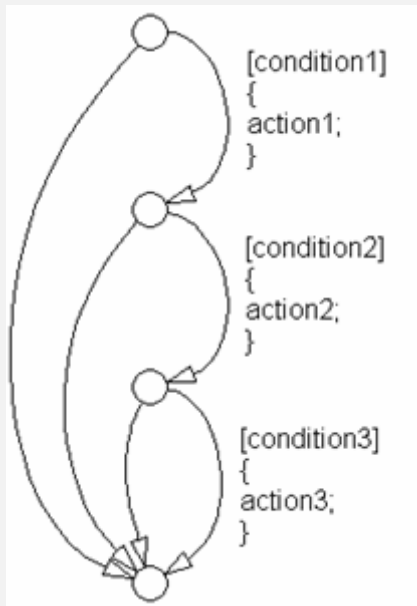
Understandable and unambiguous	<p>A guideline's description should be precise, clearly worded, concise, and should define a characteristic of a model (or part of a model) that a checking tool can evaluate. Use the words "must," "shall," "should," and "may" carefully; they have distinct meanings that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

two guidelines, one that addresses error prevention and one that addresses readability. If appropriate, make one guideline a prerequisite of the other. Also, big guidelines are more likely than small guidelines to require compromises for wide acceptance. Big guidelines may end up being weaker, less specific, and less beneficial. Small, focused guidelines are less likely to change due to compromise and easier adoption.

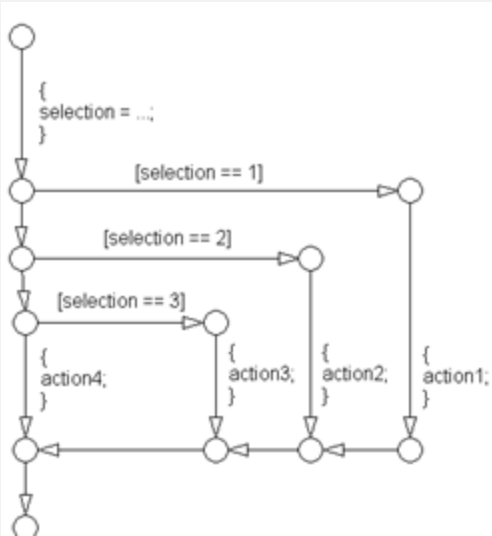
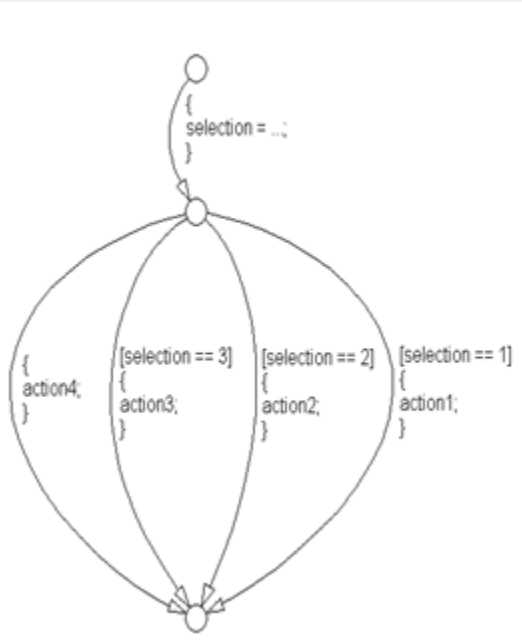
Flowchart Reference

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

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="136 300 244 326">if then</p> 	
<p data-bbox="136 890 319 916">if then else</p> 	

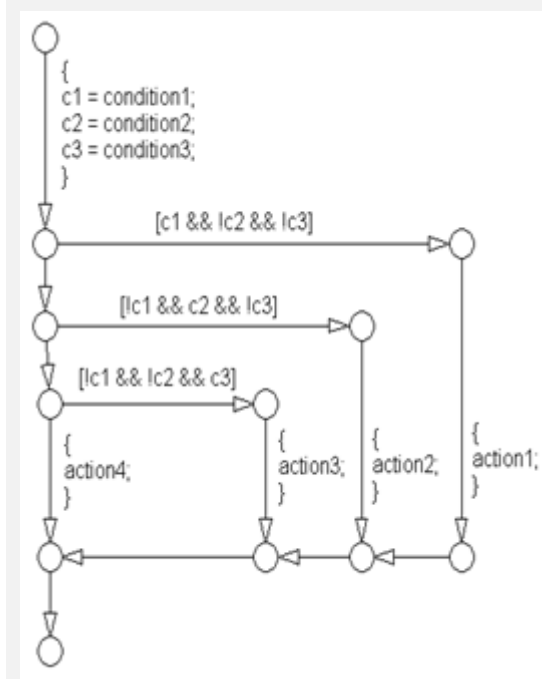
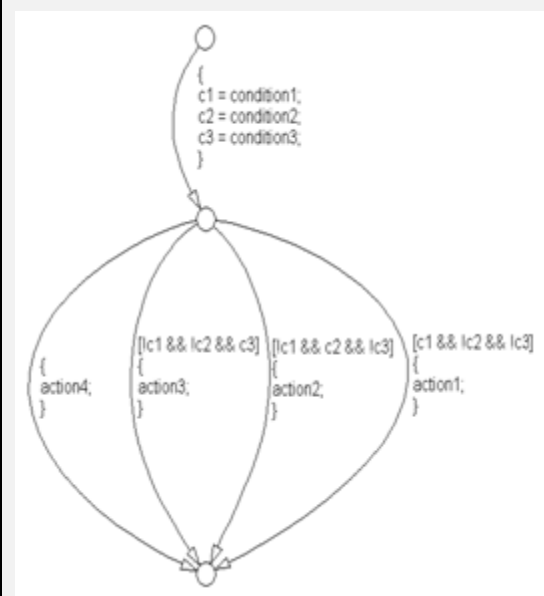
Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
<p data-bbox="136 298 360 326">if then else if</p> 	
<p data-bbox="136 841 382 869">Cascade of if then</p> 	

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

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
case with exclusive selection	
 <p>The diagram shows a vertical sequence of nodes. The top node contains the code <code>{ selection = ...; }</code>. Below it are four nodes, each with a guard condition: <code>[selection == 1]</code>, <code>[selection == 2]</code>, <code>[selection == 3]</code>, and <code>{ action4; }</code>. From the top node, three horizontal arrows branch out to the right, each leading to a node containing an action: <code>{ action1; }</code>, <code>{ action2; }</code>, and <code>{ action3; }</code>. From the bottom node, a horizontal arrow branches out to the left, leading to a node containing <code>{ action4; }</code>. All four action nodes have arrows pointing back to the bottom node, which then has an arrow pointing to a final node at the bottom.</p>	 <p>The diagram shows a single top node with the code <code>{ selection = ...; }</code>. From this node, three curved arrows branch out to the right, each leading to a node containing an action: <code>{ action1; }</code>, <code>{ action2; }</code>, and <code>{ action3; }</code>. From the bottom of these three nodes, three curved arrows branch out to the left, each leading to a node containing an action: <code>{ action4; }</code>, <code>{ action3; }</code>, and <code>{ action2; }</code>. All four action nodes have arrows pointing to a single final node at the bottom.</p>

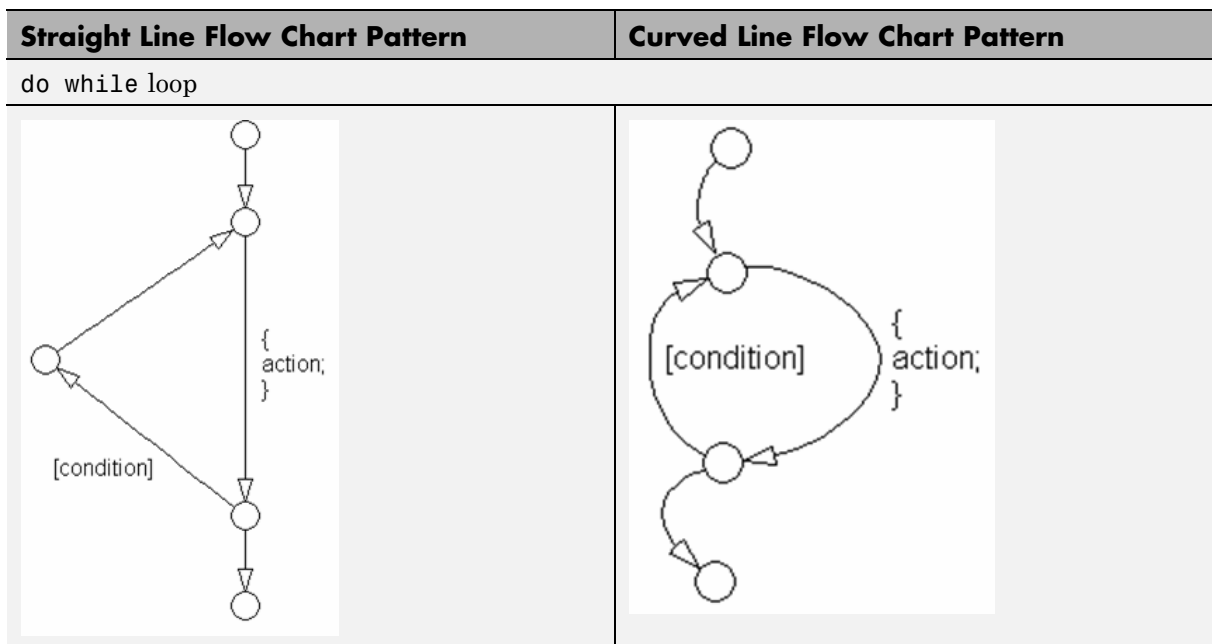
Straight Line Flow Chart Pattern

case with exclusive conditions

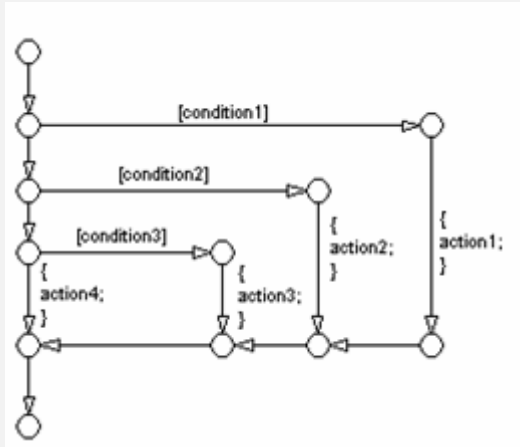
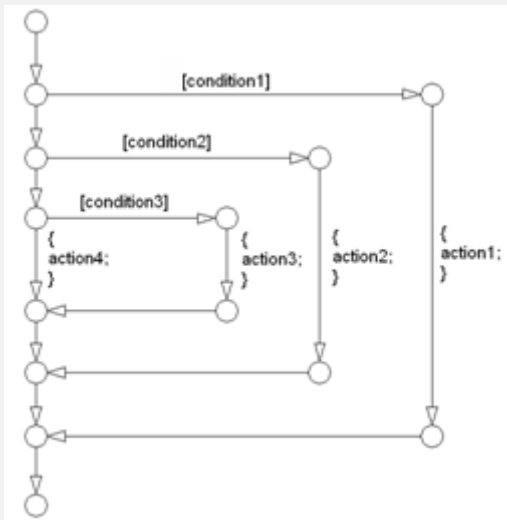
**Curved Line Flow Chart Pattern**

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

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

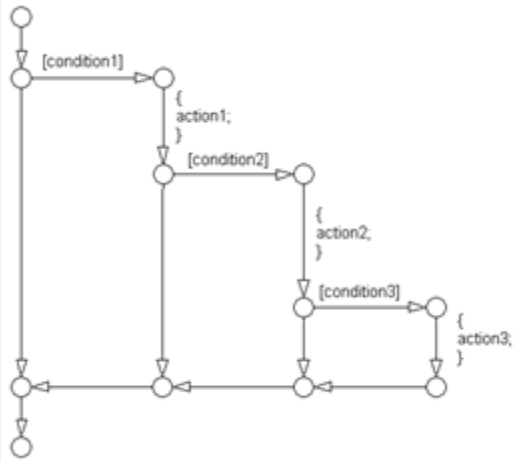
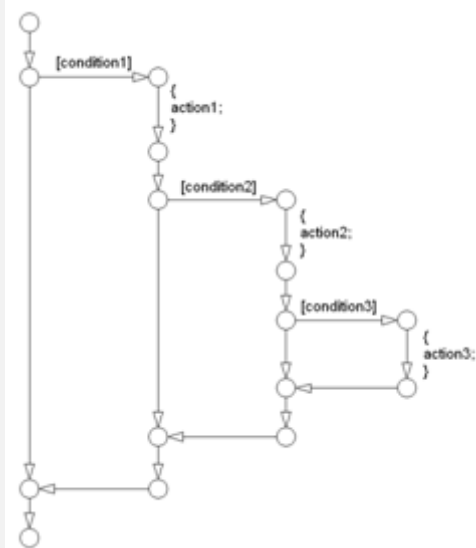


The following patterns are alternately used for If-then-else-if constructs within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
<p data-bbox="136 390 360 418">if then else if</p>  <p>The diagram illustrates the 'Straight Line Flow Chart Pattern' for an 'if then else if' construct. It features a vertical sequence of nodes. The flow starts at the top node and proceeds downwards. Three horizontal transitions, labeled [condition1], [condition2], and [condition3], branch off to the right. Each transition leads to a node containing an action block: {action1;}, {action2;}, and {action3;}. After each action block, the flow returns to the main vertical line. A fourth horizontal transition, labeled {action4;}, branches off to the left from the main vertical line, leading to a node with an empty action block {}. The flow then continues downwards from the bottom of this node.</p>	 <p>The diagram illustrates the 'Alternate Straight Line Flow Chart Pattern' for an 'if then else if' construct. It features a vertical sequence of nodes. The flow starts at the top node and proceeds downwards. Three horizontal transitions, labeled [condition1], [condition2], and [condition3], branch off to the right. Each transition leads to a node containing an action block: {action1;}, {action2;}, and {action3;}. After each action block, the flow returns to the main vertical line. A fourth horizontal transition, labeled {action4;}, branches off to the left from the main vertical line, leading to a node with an empty action block {}. The flow then continues downwards from the bottom of this node.</p>

Straight Line Flow Chart Pattern

Cascade of if then

**Alternate Straight Line Flow Chart Pattern**



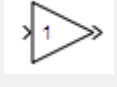


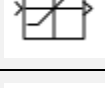
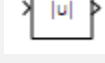
Background Information on Basic Blocks and Signals

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

Basic Blocks

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

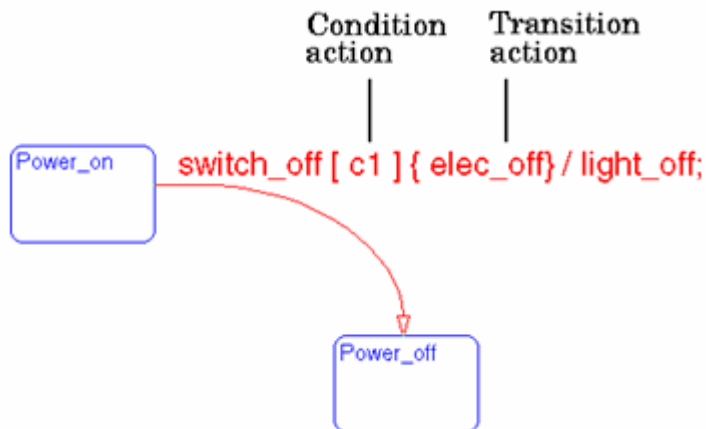
You use signal labels to make model functionality more understandable from the Simulink diagram. You can also use them to control the variable names used in simulation and code generation. Enter signal names only once (at the point of signal origination). Often, you may want to also display the signal 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 6-33 apply to all types of signals.

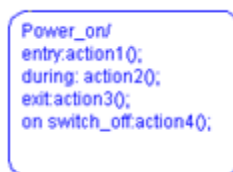
For more information about the representation of signals in Simulink models, see “Signal Basics” in the Simulink documentation.

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.

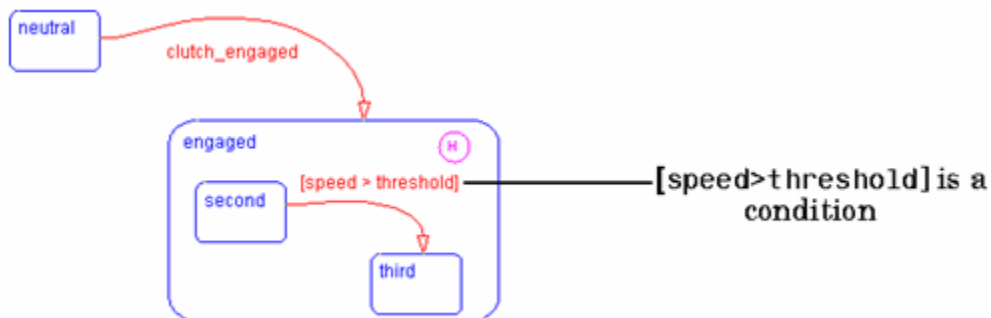
The action language defines the categories of actions you can specify and their associated notations. Violations of the action language notation are flagged as errors by the parser. This section describes the action language notation rules.

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,

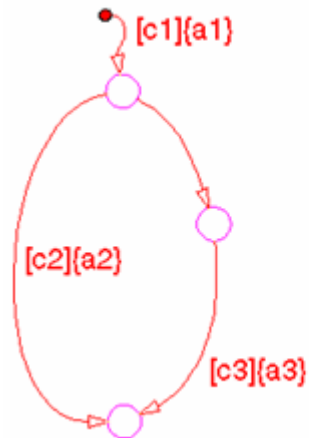


The action language defines the notation to define conditions associated with transitions.

Connective Junction

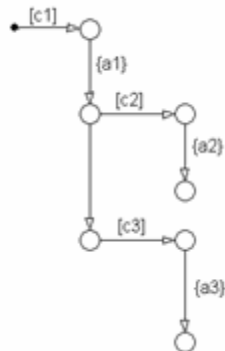
Connective junctions are decision points in the system. A connective junction is a graphical object that simplifies Stateflow diagram

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

Flowchart (also known as Flow Path)

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

Flow Subgraph


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

Hierarchy

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

History Junction

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

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

Inner Transitions

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

Library Link

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

Library Model

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

Parallelism

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

Real-Time System

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

Simulink Coder

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.

Simulink Coder Target

An executable built from code generated by the Simulink Coder product.

S-function

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.

Signal propagation

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

Signal source

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

Simulink

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

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


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

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

State

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

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

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

Stateflow Block

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

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

Stateflow Debugger

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

Stateflow Diagram

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

Stateflow Explorer

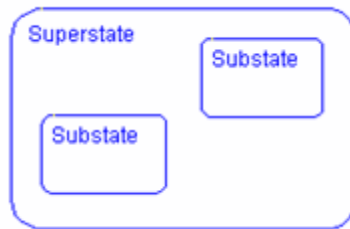
Use the Stateflow Explorer to add, remove, and modify data, event, and target objects.

Stateflow Finder

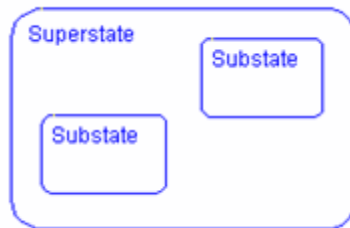
Use the Finder to display a list of objects based on search criteria that you specify. You can directly access the properties dialog box of any object in the search output display by clicking on that object.

Substate

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

**Superstate**

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

**Target**

An executable program built from code generated by Stateflow or 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 Flowchart that starts and ends on a state.

Transition Segment

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

Tunable parameters

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

True Source

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

Virtual Block

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

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

Block Name	Condition Under Which Block Is Virtual
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.