CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB®, Simulink®, and Stateflow®

Version 3.0

MathWorks Automotive Advisory Board (MAAB)

CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB®, SIMULINK®, AND STATEFLOW® ... 1

1. HISTORY ... 6

2. INTRODUCTION ... 7

2.1. MOTIVATION ... 7
2.2. NOTES ON VERSION 3.0 ... 7
2.3. GUIDELINE TEMPLATE ... 7
2.3.1. Guideline ID: ... 8
2.3.2. Guideline Title: ... 8
2.3.3. Priority: ... 8
2.3.4. Scope: ... 9
2.3.5. MATLAB® Versions ... 9
2.3.6. Prerequisites: ... 9
2.3.7. Description: ... 10
2.3.8. Rationale: ... 10
2.3.9. Last change: ... 10
2.4. DOCUMENT USAGE ... 10
2.4.1. Guideline Interaction Semantics ... 10
2.4.2. Masked Subsystems and Readability Rules ... 11

3. SOFTWARE ENVIRONMENT ... 12

3.1. GENERAL GUIDELINES ... 12
3.1.1. na_0026: Consistent software environment ... 12
3.1.2. na_0027: Use of only standard library blocks ... 12

4. NAMING CONVENTIONS ... 14

4.1. GENERAL GUIDELINES ... 14
4.1.1. ar_0001: Filenames ... 14
4.1.2. ar_0002: Directory names ... 14
4.1.3. na_0035: Adoption of naming conventions ... 15
4.2. MODEL CONTENT GUIDELINES ... 16
4.2.1. jc_0201: Usable characters for Subsystem name ... 16
4.2.2. jc_0211: Usable characters for Inport block and Outport block ... 16
4.2.3. jc_0221: Usable characters for signal line name ... 17
4.2.4. na_0030: Usable characters for Simulink Bus names ... 17
4.2.5. jc_0231: Usable characters for block names ... 18
4.2.6. na_0014: Use of local language in Simulink and Stateflow ... 19

5. MODEL ARCHITECTURE ... 21

5.1. SIMULINK® AND STATEFLOW® PARTITIONING ... 21
5.1.1. na_0006: Guidelines for mixed use of Simulink and Stateflow ... 21
5.1.2. na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines ... 27
5.2. SUBSYSTEM HIERARCHIES ... 27
5.2.1. db_0143: Similar block types on the model levels ... 27
5.2.2. db_0144: Use of Subsystems ... 29
5.2.3. db_0040: Model hierarchy ... 30
5.2.4. na_0037: Use of single variable variant conditionals ... 30
5.2.5. na_0020: Number of inputs to variant subsystems ... 31
5.2.6. na_0036: Default Variant ... 31
5.3. J-MAAB MODEL ARCHITECTURE DECOMPOSITION ... 32
5.3.1. jc_0301: Controller model ... 32
5.3.2. jc_0311: Top layer / root level ... 33
5.3.3. jc_0321: Trigger layer ... 34
5.3.4. jc_0331: Structure layer ... 34

5.3.5. jc_0341: Data flow layer *35*

6. MODEL CONFIGURATION OPTIONS **37**

6.1.1. jc_0011: Optimization parameters for Boolean data types *37*
6.1.2. jc_0021: Model diagnostic settings *37*

7. SIMULINK **39**

7.1. DIAGRAM APPEARANCE 39
7.1.1. na_0004: Simulink model appearance *39*
7.1.2. db_0043: Simulink font and font size *40*
7.1.3. db_0042: Port block in Simulink models *40*
7.1.4. na_0005: Port block name visibility in Simulink models *41*
7.1.5. jc_0081: Icon display for Port block *42*
7.1.6. jm_0002: Block resizing *43*
7.1.7. db_0142: Position of block names *43*
7.1.8. jc_0061: Display of block names *44*
7.1.9. db_0146: Triggered, enabled, conditional Subsystems *45*
7.1.10. db_0140: Display of basic block parameters *46*
7.1.11. db_0032: Simulink signal appearance *47*
7.1.12. db_0141: Signal flow in Simulink models *47*
7.1.13. jc_0171: Maintaining signal flow when using Goto and From blocks *48*
7.1.14. na_0032: Use of Merge Blocks *49*
7.1.15. jm_0010: Port block names in Simulink models *50*
7.1.16. jc_0281: Naming of Trigger Port block and Enable Port block *50*
7.2. SIGNALS 51
7.2.1. na_0008: Display of labels on signals *51*
7.2.2. na_0009: Entry versus propagation of signal labels *52*
7.2.3. db_0097: Position of labels for signals and busses *53*
7.2.4. db_0081: Unconnected signals, block inputs and block outputs *54*
7.3. BLOCK USAGE 54
7.3.1. na_0003: Simple logical expressions in If Condition block *54*
7.3.2. na_0002: Appropriate implementation of fundamental logical and numerical operations *56*
7.3.3. jm_0001: Prohibited Simulink standard blocks inside controllers *57*
7.3.4. hd_0001: Prohibited Simulink sinks *59*
7.3.5. na_0011: Scope of Goto and From blocks *59*
7.3.6. jc_0141: Use of the Switch block *60*
7.3.7. jc_0121: Use of the Sum block *61*
7.3.8. jc_0131: Use of Relational Operator block *63*
7.3.9. jc_0161: Use of Data Store Read/Write/Memory blocks *63*
7.4. BLOCK PARAMETERS 64
7.4.1. db_0112: Indexing *64*
7.4.2. na_0010: Grouping data flows into signals *64*
7.4.3. db_0110: Tunable parameters in basic blocks *65*
7.5. SIMULINK PATTERNS 66
7.5.1. na_0012: Use of Switch vs. If-Then-Else Action Subsystem *66*
7.5.2. db_0114: Simulink patterns for If-then-else-if constructs *67*
7.5.3. db_0115: Simulink patterns for case constructs *68*
7.5.4. na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches *69*
7.5.5. db_0116: Simulink patterns for logical constructs with logical blocks *70*
7.5.6. db_0117: Simulink patterns for vector signals *71*
7.5.7. jc_0351: Methods of initialization *73*
7.5.8. jc_0111: Direction of Subsystem *75*

8. STATEFLOW **77**

8.1. CHART APPEARANCE 77
8.1.1. db_0123: Stateflow port names *77*

8.1.2. db_0129: Stateflow transition appearance 77
8.1.3. db_0137: States in state machines 78
8.1.4. db_0133: Use of patterns for Flowcharts 79
8.1.5. db_0132: Transitions in Flowcharts 79
8.1.6. jc_0501: Format of entries in a State block 81
8.1.7. jc_0511: Setting the return value from a graphical function 82
8.1.8. jc_0531: Placement of the default transition 83
8.1.9. jc_0521: Use of the return value from graphical functions 84
8.2. STATEFLOW DATA AND OPERATIONS 85
8.2.1. na_0001: Bitwise Stateflow operators 85
8.2.2. jc_0451: Use of unary minus on unsigned integers in Stateflow 87
8.2.3. na_0013: Comparison operation in Stateflow 87
8.2.4. db_0122: Stateflow and Simulink interface signals and parameters 88
8.2.5. db_0125: Scope of internal signals and local auxiliary variables 89
8.2.6. jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow 90
8.2.7. jc_0491: Reuse of variables within a single Stateflow scope 91
8.2.8. jc_0541: Use of tunable parameters in Stateflow 93
8.2.9. db_0127: MATLAB commands in Stateflow 93
8.2.10. jm_0011: Pointers in Stateflow 94
8.3. EVENTS 95
8.3.1. db_0126: Scope of events 95
8.3.2. jm_0012: Event broadcasts 95
8.4. STATECHART PATTERNS 97
8.4.1. db_0150: State machine patterns for conditions 97
8.4.2. db_0151: State machine patterns for transition actions 98
8.5. FLOWCHART PATTERNS 98
8.5.1. db_0148: Flowchart patterns for conditions 98
8.5.2. db_0149: Flowchart patterns for condition actions 100
8.5.3. db_0134: Flowchart patterns for If constructs 101
8.5.4. db_0159: Flowchart patterns for case constructs 103
8.5.5. db_0135: Flowchart patterns for loop constructs 105
8.6. STATE CHART ARCHITECTURE 106
8.6.1. na_0038: Levels in Stateflow charts 106
8.6.2. na_0039: Use of Simulink in Stateflow charts 107
***8.6.3.** na_0040: Number of states per container 108*
***8.6.4.** na_0041: Selection of function type 108*
8.6.5. na_0042: Location of functions 109

9. ENUMERATED DATA 111

9.1.1. na_0033: Enumerated Types Usage 111
9.1.2. na_0031: Definition of default enumerated value 111

10. MATLAB FUNCTIONS 112

10.1. MATLAB FUNCTION APPEARANCE 112
10.1.1. na_0018: Number of nested if/else and case statement 112
10.1.2. : na_0019: Restricted Variable Names 112
10.1.3. na_0025: MATLAB Function Header 113
10.2. MATLAB FUNCTION DATA AND OPERATIONS 113
10.2.1. na_0034: MATLAB Function block input/output settings 113
10.2.2. na_0024: Global Variables 114
10.3. MATLAB FUNCTION PATTERNS 115
10.3.1. na_0022: Recommended patterns for Switch / Case statements 115
10.4. MATLAB FUNCTION USAGE 116
10.4.1. na_0016: Source lines of MATLAB Functions 116
10.4.2. na_0017: Number of called function levels 116
10.4.3. na_0021: Strings 117

11. APPENDIX A: RECOMMENDATIONS FOR AUTOMATION TOOLS 119

12. APPENDIX B: GUIDELINE WRITING 120

13. APPENDIX C: FLOWCHART REFERENCE 121

14. OBSOLETE RULES 127

14.1. REMOVED IN VERSION 2.2 127
14.2. REMOVED IN VERSION 3.0 127

15. GLOSSARY 128

1.History

Date	Change
02.04.2001	Initial document Release, Version 1.00
04.27.2007	Version 2.00 Update release
07.30.2011	Version 2.20 Update release
08.31.2012	Version 3.0 Update release

2.Introduction

2.1. Motivation

The MAAB guidelines are an important basis 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
- Handing over of research or predevelopment projects to product development

2.2. 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 (http://www.mathworks.com/aerospace-defense/standards/nasa.html). Rules that are referenced from the NASA Orion guideline are noted with a "See also" field that provides the original rule number.

2.3. Guideline template

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

ID: Title	**XX_nnnn: Title of the guideline (unique, short)**
Priority	One of mandatory / strongly recommended / recommended
Scope	MAAB, NA-MAAB, J-MAAB, Specific Company (for optional local company usage)
MATLAB® Version	all RX, RY, RZ RX and earlier RX and later RX through RY
Prerequisites	Links to guidelines, which are prerequisite to this guideline (ID+title)
Description	Description of the guideline (text, images)
Rationale	Motivation for the guideline
Last Change	Version number of last change

Note: The elements of this template are the minimum required items that must be present for proper understanding and exchange of guidelines. The addition of project- or vendor fields to this template is possible as long as their meaning does not overlap with any of the existing fields. In fact, such additions are even encouraged if they help to integrate other guideline templates and lead to a wider acceptance of the core template itself.

2.3.1. 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 will not be changed.
- 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.
- No new rules will be written with these legacy prefixes.

2.3.2. 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. The description of the guideline might change over time, but the title should remain stable.

2.3.3. 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		
• Guidelines that all companies agree to that are absolutely essential • Guidelines that all companies conform to 100%	• Guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming to the guideline 100% • Models should conform to these guidelines to the greatest extent	• Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model • Guidelines where conformance is preferred, but not required

	possible; however 100% compliance is not required	
CONSEQUENCES If the guideline is violated		
• Essential items are missing • The model might not work properly	• The quality and the appearance deteriorates • There may be an adverse effect on maintainability, portability, and reusability	• The appearance will not conform with other projects
WAIVER POLICY If the guideline is intentionally ignored,		
• The reasons must be documented		

2.3.4. Scope:

The scope can be set to one of the following
MAAB (MathWorks Automotive Advisory Board)
J-MAAB (Japan MAAB)
NA-MAAB (North American MAAB)

"MAAB" is a group of automotive manufacturers and suppliers that work closely together with MathWorks. MAAB includes the sub-groups J-MAAB, and NA-MAAB.

“J-MAAB” is 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” is a subgroup of MAAB that includes automotive manufacturers and suppliers in USA and Europe and works closely with MathWorks. That rule is local rule in USA and Europe. Coverage is USA and Europe.

2.3.5. MATLAB® Versions

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

- All : All versions of MATLAB
- RX, RY, 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

2.3.6. Prerequisites:

- This field is for links to other guidelines that are prerequisite to 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.

2.3.7. Description:

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

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

2.3.8. Rationale:

The guidelines can be recommended for one or more of the following reasons.

- Readability: Easily understood algorithms
  - Readable models
  - Uniform appearance of models, code, and documentation
  - Clean interfaces
  - Professional documentation
- Workflow: Effective development process and workflow
  - Ease of maintenance
  - Rapid model changes
  - Reusable components
  - Problem-free exchange of models
  - Model portability
- Simulation: Efficient simulation and analysis
  - Simulation speed
  - Simulation memory
  - Model instrumentation
- Verification & Validation: Ability to verify and validate a model and generated code with:
  - Requirements Traceability
  - Testing
  - Problem-free system integration
  - Clean interfaces
- Code generation: Generation of code that is efficient and effective for embedded systems
  - Fast software changes
  - Robustness of generated code

2.3.9. Last change:

The “Last Change” field contains the document version number.

2.4. Document Usage

The following paragraphs provide information on using this document as reference and for compiling a project-specific guideline document. Information on automated checking of the guidelines can be found in Appendix A.

2.4.1. Guideline Interaction Semantics

The initial sections of the document, naming conventions and model architecture, provide basic guidelines that apply to all types of models. The later sections, Simulink and Stateflow, provide specific rules for those environments. Some guidelines are dependent on other guidelines and are explicitly listed throughout the template.

2.4.2. Masked Subsystems and Readability Rules

If users do not view the content of masked subsystems within a model, the guidelines for readability are not applicable.

3.Software Environment

3.1. General Guidelines

3.1.1. na_0026: Consistent software environment

ID: Title	na_0026: Consistent software environment
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	See Description
Prerequisites	
Description	During software development, it is recommended that a consistent software environment is used across the project. Software includes, but is not limited, to: • MATLAB • Simulink • C Compiler (for simulation) • C Compiler (for target hardware) Consistent software environment implies that the same version of the software is used across the full project. The version number applies to any patches or extensions to the software used by a group.
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☑ Code Generation
See also	jh_0042: Required software
Last Change	V3.00

3.1.2. na_0027: Use of only standard library blocks

ID: Title	na_0027: Use of only standard library blocks
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	Companies should specify a subset of Simulink blocks for use when developing models. The block list can include custom block libraries developed by the company or third parties. Models should be built only from these blocks. Non-compliant blocks can be used during development. If non-compliant blocks are used, they should be marked either with a color, icon and / or annotation. These blocks must be removed prior to use in production code generation.

Rationale	☑ Readability ☐ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
See also	hyl_0201: Use of standard library blocks only
Last Change	V3.00

4.Naming Conventions

4.1. General Guidelines

4.1.1. ar_0001: Filenames

<table>
<tr><td>ID: Title</td><td>ar_0001: Filenames</td></tr>
<tr><td>Priority</td><td>Mandatory</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>A filename conforms to the following constraints:
<table>
<tr><td>FORM</td><td>filename = name.extension
name: no leading digits, no blanks
extension: no blanks</td></tr>
<tr><td>UNIQUENESS</td><td>☐ all filenames within the parent project directory
☐ cannot conflict with C / C++ or MATLAB keywords</td></tr>
<tr><td>ALLOWED CHARACTERS</td><td>name
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _
extension:
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9</td></tr>
<tr><td>UNDERSCORES</td><td>name:
can use underscores to separate parts
cannot have more than one consecutive underscore
cannot start with an underscore
cannot end with an underscore
extension:
should not use underscores</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☑ Readability
☑ Workflow
☑ Simulation
☐ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V3.00</td></tr>
</table>

4.1.2. ar_0002: Directory names

ID: Title	**ar_0002: Directory names**
Priority	mandatory
Scope	MAAB

<table>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>A directory name conforms to the following constraints:
<table>
<tr><td>FORM</td><td>directory name = name
name: no leading digits, no blanks</td></tr>
<tr><td>UNIQUENESS</td><td>all directory names within the parent project directory</td></tr>
<tr><td>ALLOWED CHARACTERS</td><td>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 _</td></tr>
<tr><td>UNDERSCORES</td><td>name:
underscores can be used to separate parts
cannot have more than one consecutive underscore
cannot start with an underscore
cannot end with an underscore</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☑ Readability ☐ Verification and Validation
☑ Workflow ☑ Code Generation
☑ Simulation</td></tr>
<tr><td>Last Change</td><td>V1.00</td></tr>
</table>

4.1.3. na_0035: Adoption of naming conventions

<table>
<tr><td>ID: Title</td><td>na_0035: Adoption of naming conventions</td></tr>
<tr><td>Priority</td><td>Recommended</td></tr>
<tr><td>Scope</td><td>NA-MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>Adoption of a naming convention is recommended. A naming convention provides guidance for naming blocks, signals, parameters and data types. Naming conventions frequently cover issues such as:

Compliance with the programing language and downstream tools
Length
Use of symbols
Readability
Use of underscores
Use of capitalization
Encoding information
Use of “meaningful” names
Standard abbreviations and acronyms
Data type
Engineering units
Data ownership
Memory type
</td></tr>
<tr><td>Rationale</td><td>☑ Readability ☐ Verification and Validation
☑ Workflow ☑ Code Generation</td></tr>
</table>

	☑ Simulation
Last Change	V3.00

4.2. Model Content Guidelines

4.2.1. jc_0201: Usable characters for Subsystem name

ID: Title	**jc_0201: Usable characters for Subsystem names**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The names of all Subsystem blocks should conform to the following constraints: FORM — **name**: • should not start with a number • should not have blank spaces • should not have 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**: • underscores can be used to separate parts • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

4.2.2. jc_0211: Usable characters for Inport block and Outport block

ID: Title	**jc_0211: Usable characters for Inport block and Outport block**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The names of all Inport blocks and Outport blocks should conform to the following constraints: FORM — **name**: • should not start with a number • should not have blank spaces • should not include carriage returns

	<table><tr><td>ALLOWED CHARACTERS</td><td>**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 _</td></tr><tr><td>UNDERSCORES</td><td>**name**: • underscores can be used to separate parts • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore</td></tr></table>
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

4.2.3. jc_0221: Usable characters for signal line name

ID: Title	jc_0221: Usable characters for signal line names
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	All named signals should conform to the following constraints: <table><tr><td>FORM</td><td>**name**: • should not start with a number • should not have blank spaces • should not have any control characters • should not include carriage returns</td></tr><tr><td>ALLOWED CHARACTERS</td><td>**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 _</td></tr><tr><td>UNDERSCORES</td><td>**name**: • underscores can be used to separate parts • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore</td></tr></table>
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

4.2.4. na_0030: Usable characters for Simulink Bus names

ID: Title	na_0030: Usable characters for Simulink Bus Names
Priority	strongly recommended

<table>
<tr><td>Scope</td><td>NA-MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>All Simulink Bus names should conform to the following constraints:
<table>
<tr><td>FORM</td><td>name:
Should not start with a number
Should not have blank spaces
Carriage returns are not allowed</td></tr>
<tr><td>ALLOWED CHARACTERS</td><td>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 _</td></tr>
<tr><td>UNDERSCORES</td><td>name:
Can use underscores to separate parts
Cannot have more than one consecutive underscore
Cannot start with an underscore
Cannot end with an underscore</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>See Also</td><td>jh_0040: Usable characters for Simulink Bus Names</td></tr>
<tr><td>Last Change</td><td>V3.00</td></tr>
</table>

4.2.5. jc_0231: Usable characters for block names

<table>
<tr><td>ID: Title</td><td>jc_0231: Usable characters for block names</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>jc_0201: Usable characters for Subsystem names</td></tr>
<tr><td>Description</td><td>All named blocks should conform to the following constraints:
<table>
<tr><td>FORM</td><td>name:
should not start with a number
should not start with a blank space
may not use double byte characters
carriage returns are allowed</td></tr>
<tr><td>ALLOWED CHARACTERS</td><td>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 _</td></tr>
</table>
Note: this rule does not apply to Subsystem blocks.</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Verification and Validation</td></tr>
</table>

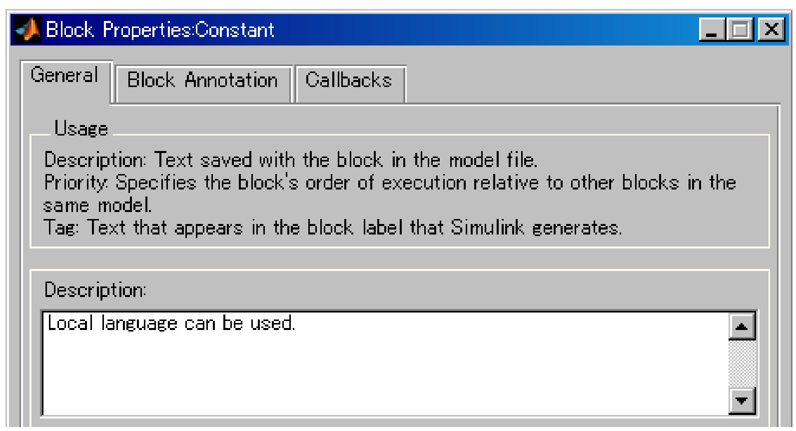
	☐ Simulation ☐ Code Generation
Last Change	V2.00

4.2.6. na_0014: Use of local language in Simulink and Stateflow

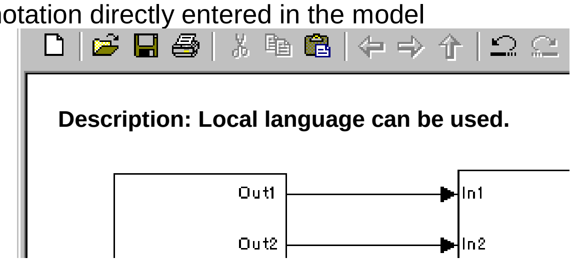
<table>
<tr><td>ID: Title</td><td>na_0014: Use of local language in Simulink and Stateflow</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>J-MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>
The local language should be used only in descriptive fields. Descriptive fields are text entry points that do not affect code generation or simulation. Examples of descriptive fields include

Simulink Example

- The Description field in the Block Properties

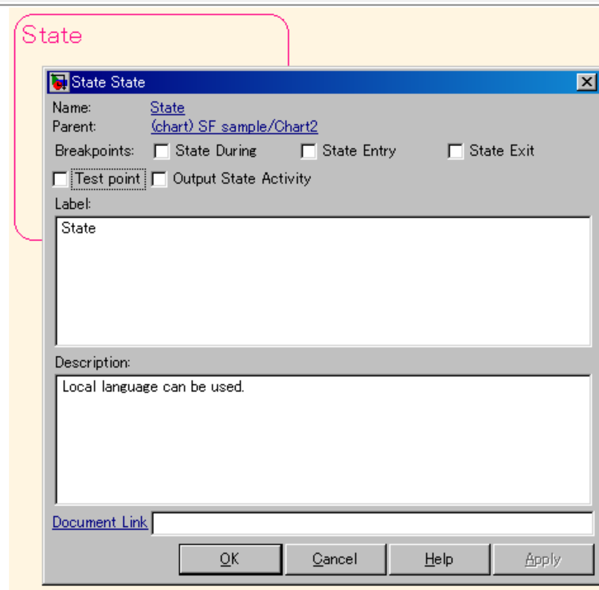
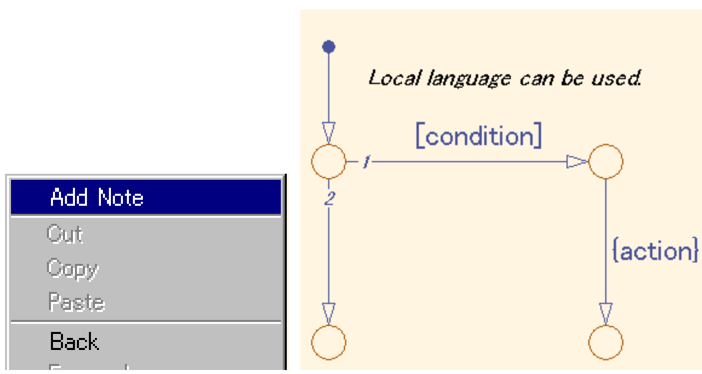


- Text annotation directly entered in the model



Stateflow Example

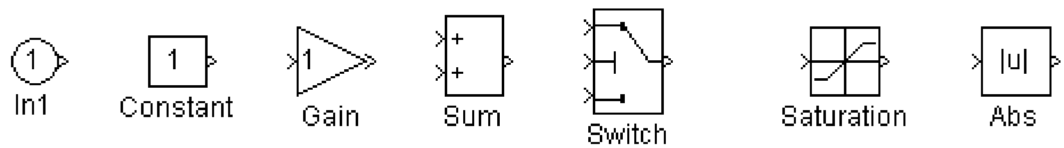
- The Description field of the chart or state Properties
</td></tr>
</table>

	 • Annotation description added using Add Note  Note: It is possible that Simulink can't open a model that includes local language on the different character encoding systems; thus, it is important to pay attention when using local characters in case of exchanging models between overseas.
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

5.Model Architecture

Basic Blocks

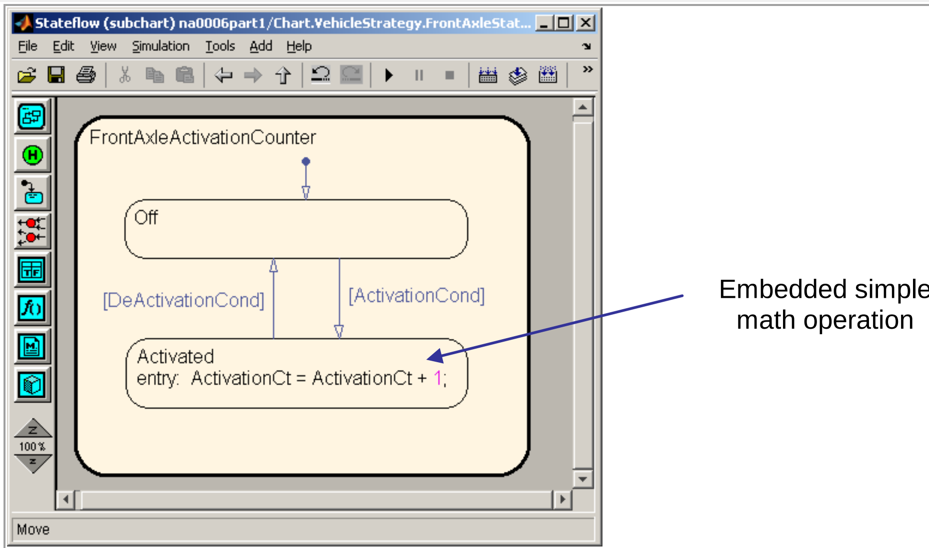
This document uses the term “Basic Blocks” to refer to blocks from the base Simulink library. Examples of basic blocks:



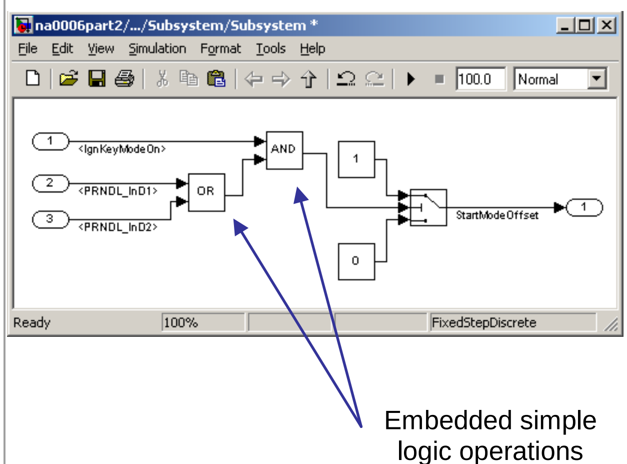
5.1. Simulink® and Stateflow® Partitioning

5.1.1. na_0006: Guidelines for mixed use of Simulink and Stateflow

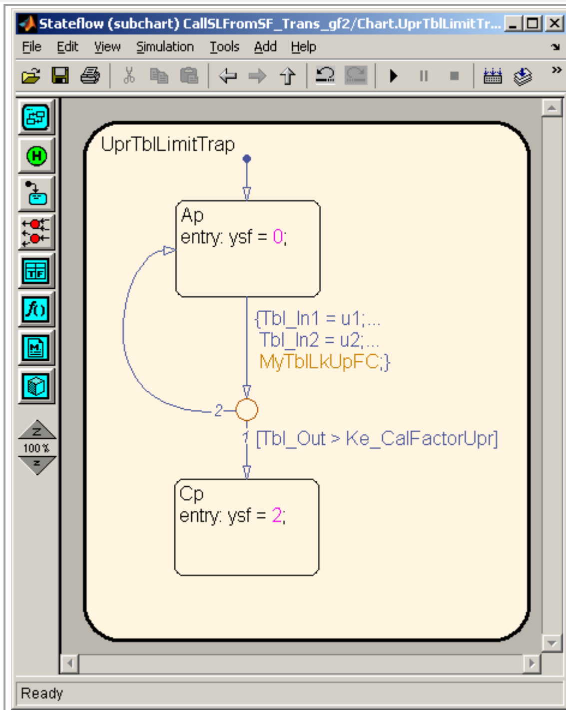
ID: Title	**na_0006: Guidelines for mixed use of Simulink and Stateflow**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled. • If the function primarily involves complicated logical operations, use Stateflow diagrams. • Stateflow should be used to implement modal logic – where the control function to be performed at the current time depends on a combination of *past and present logical conditions.* • If the function primarily involves numerical operations, use Simulink features. Specifics: • If the primary nature of the function is logical, but some simple numerical calculations are done to support the logic, implement the simple numerical functions using the Stateflow action language.



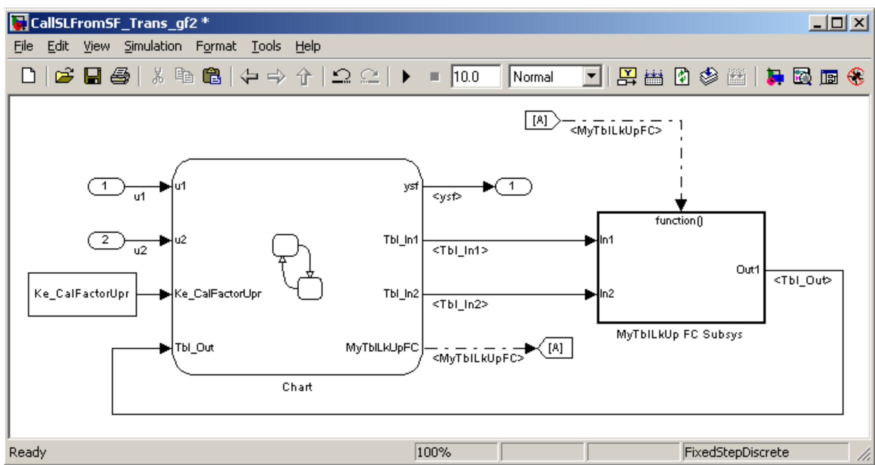
- If the primary nature of the function is numeric, but some simple logical operations are done to support the arithmetic, implement the simple logical functions with Simulink blocks.



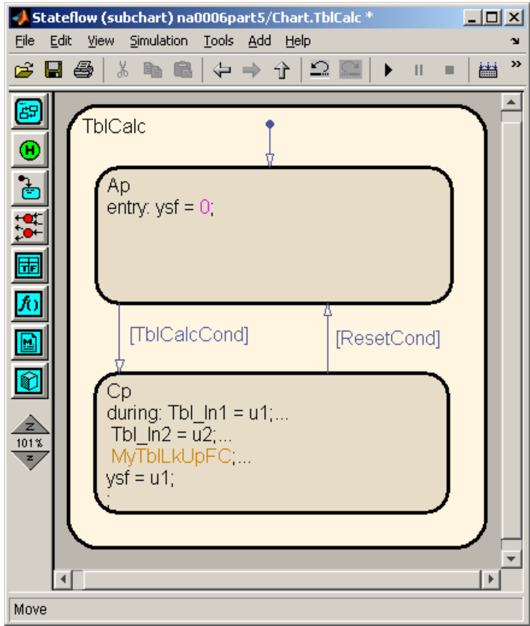
- If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, use a Simulink subsystem to implement the numerical calculations. The Stateflow software should invoke the execution of this subsystem, using a function-call.

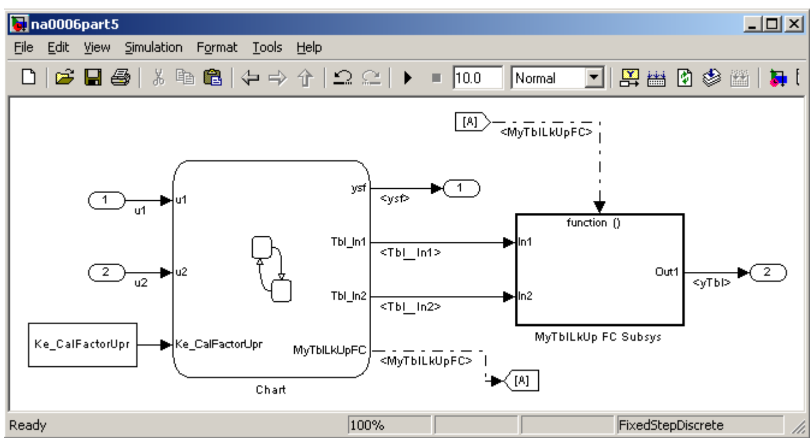


Stateflow (subchart) CallSLFromSF_Trans_gf2/Chart.UprTblLimitTr...
File Edit View Simulation Tools Add Help
UprTblLimitTrap
Ap
entry: ysf = 0;
{Tbl_In1 = u1;...
Tbl_In2 = u2;...
MyTblLkUpFC;}
2
1
[Tbl_Out > Ke_CalFactorUpr]
Cp
entry: ysf = 2;
100%
Ready



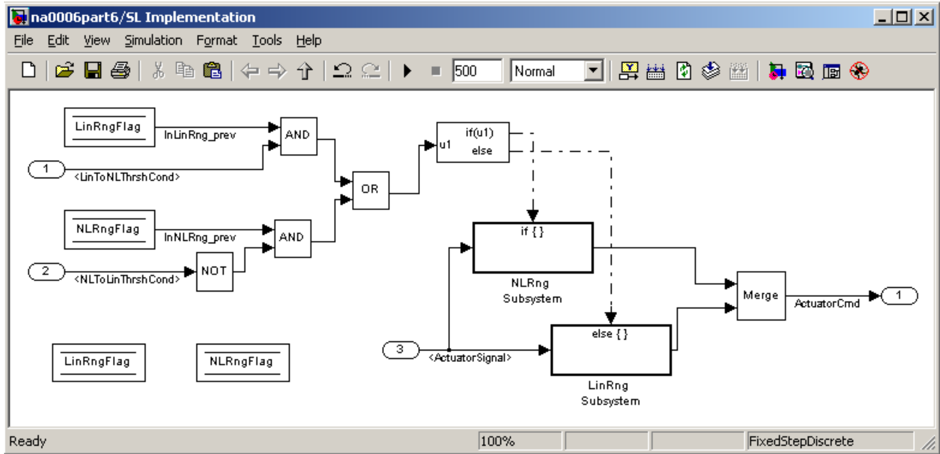
CallSLFromSF_Trans_gf2 *
File Edit View Simulation Format Tools Help
10.0
Normal
[A]
<MyTblLkUpFC>
1
u1
2
u2
Ke_CalFactorUpr
u1
u2
Ke_CalFactorUpr
Tbl_Out
ysf
Tbl_In1
Tbl_In2
MyTblLkUpFC
Chart
<ysf>
1
<Tbl_In1>
<Tbl_In2>
<MyTblLkUpFC>
[A]
function()
In1
In2
Out1
MyTblLkUp FC Subsys
<Tbl_Out>
Ready
100%
FixedStepDiscrete

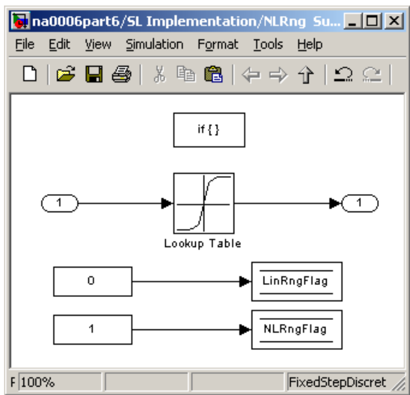




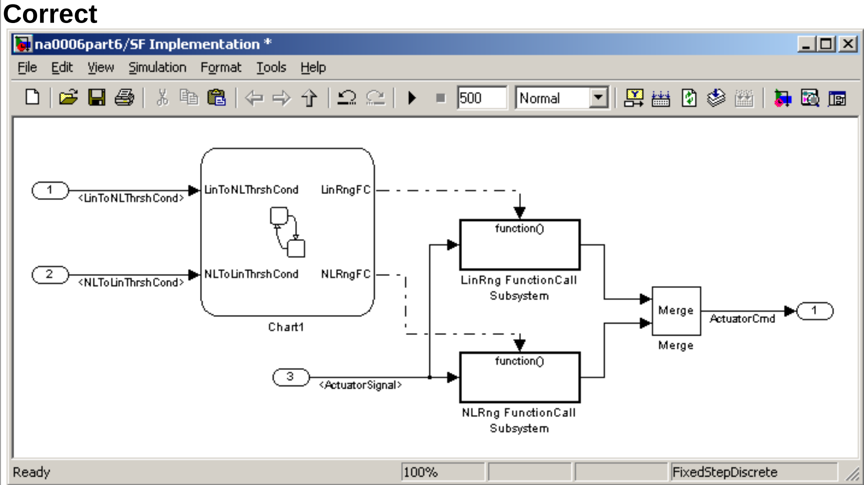
- Use the Stateflow product to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*. (If there is a need to store the result of a logical condition test in Simulink, for example, by storing a flag, this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

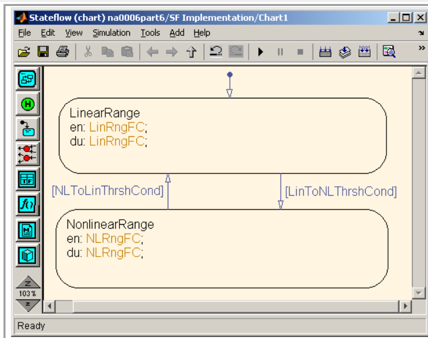
Incorrect


na0006part6/SL Implementation
File Edit View Simulation Format Tools Help
500
Normal
LinRngFlag
InLinRng_prev
AND
1
<LinToNLThrshCond>
OR
if(u1)
u1
else
NLRngFlag
InNLRng_prev
AND
2
<NLToLinThrshCond>
NOT
if { }
NLRng
Subsystem
Merge
ActuatorCmd
1
LinRngFlag
NLRngFlag
3
<ActuatorSignal>
else { }
LinRng
Subsystem
Ready
100%
FixedStepDiscrete


na0006part6/SL Implementation/NLRng Su...
File Edit View Simulation Format Tools Help
if { }
1
1
Lookup Table
0
LinRngFlag
1
NLRngFlag
100%
FixedStepDiscret

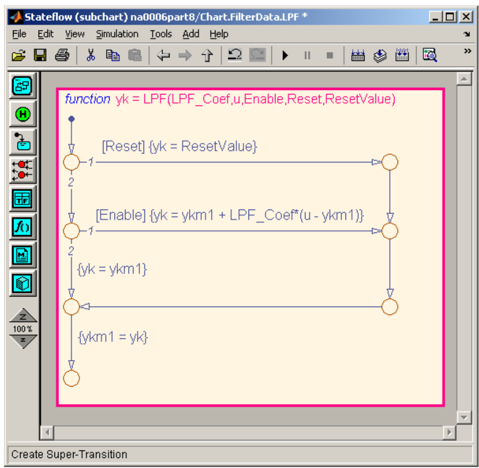
Correct


na0006part6/SF Implementation *
File Edit View Simulation Format Tools Help
500
Normal
1
<LinToNLThrshCond>
LinToNLThrshCond
LinRngFC
function()
LinRng FunctionCall
Subsystem
2
<NLToLinThrshCond>
NLToLinThrshCond
NLRngFC
Chart1
Merge
Merge
ActuatorCmd
1
function()
3
<ActuatorSignal>
NLRng FunctionCall
Subsystem
Ready
100%
FixedStepDiscrete

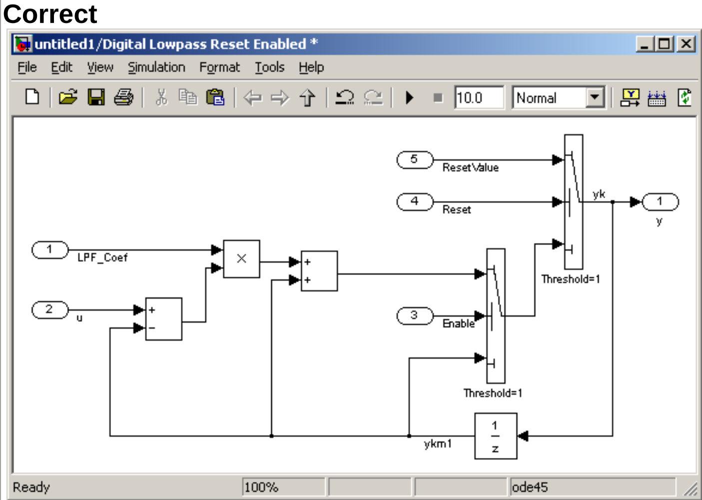


- Simulink should be used to implement numerical expressions containing continuously-valued states, e.g., difference equations, integrals, derivatives, and filters.

Incorrect



Correct



Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☑ Simulation
Last Change	V2.00

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

<table>
<tr><td>ID: Title</td><td>na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>na_0006: Guidelines for Mixed use of Simulink and Stateflow</td></tr>
<tr><td>Description</td><td>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 model 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.

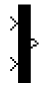
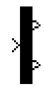
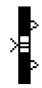
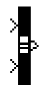
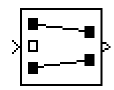
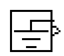
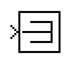
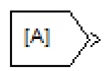
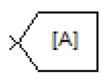
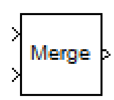
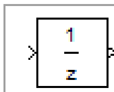
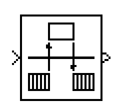
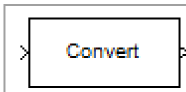
Specifics:

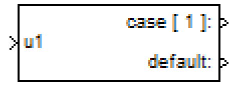
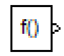
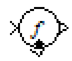
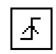
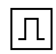
If the primary nature of the function segment is to calculate modes or states, but if-then-else statements are required, add a flow chart to a state within the state chart. (See 7.5 Flowchart Patterns)
</td></tr>
<tr><td>Rationale</td><td>☑ Readability ☑ Verification and Validation
☑ Workflow ☑ Code Generation
☑ Simulation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

5.2. Subsystem Hierarchies

5.2.1. db_0143: Similar block types on the model levels

<table>
<tr><td>ID: Title</td><td>db_0143: Similar block types on the model levels</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>NA-MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>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 Subsystem or only basic blocks). The blocks listed in this rule are used for signal routing. You can place them at any level of the model.
<table>
<tr><td colspan="2">Blocks which can be placed on every model level:</td></tr>
<tr><td>Inport</td><td></td></tr>
</table></td></tr>
</table>

Outport	
Mux	
Demux	
Bus Selector	
Bus Creator	
Selector	
Ground	
Terminator	
From	
Goto	
Merge	
Unit Delay	
Rate Transition	
Data Type Conversion	
Data Store Memory	

	If	
	Case	
	Function-Call Generator	
	Function-Call Split	
	Trigger(1)	
	Enable(2)	
	Action port(3)	

Note	1.) Starting in R2009a, the Trigger block is allowed at the root level of the model. 2.) Starting in R2011b, the Enabled block is allowed at the root level of the model. 3.) Action ports are not allowed at the root level of a model. 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 ☑ Verification and Validation ☑ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.20

5.2.2. db_0144: Use of Subsystems

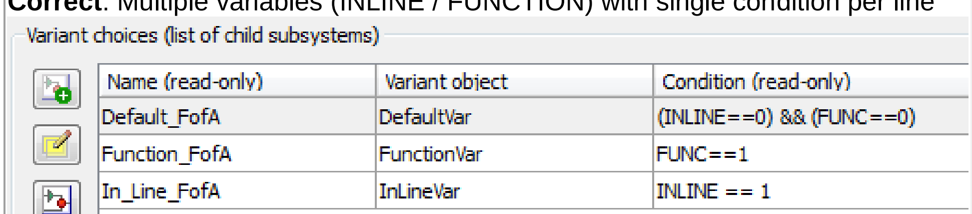
ID: Title	**db_0144: Use of Subsystems**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Blocks in a Simulink diagram should be grouped together into subsystems based on functional decomposition of the algorithm, or portion thereof, represented in the diagram. Avoid grouping blocks into subsystems primarily for the purpose of saving space in the diagram. Each subsystem in the diagram should represent a unit of functionality required to accomplish the purpose of the model or submodel. Blocks

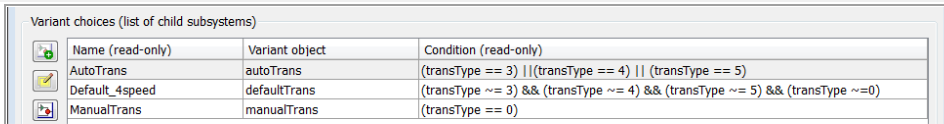
	can also be grouped together based on behavioral variants or timing. If creation of a subsystem is required for readability issues, then a virtual subsystem should be used.
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.20

5.2.3. db_0040: Model hierarchy

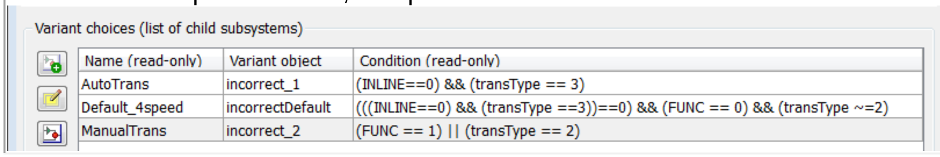
ID: Title	**db_0040: Model hierarchy**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The model hierarchy should correspond to the functional structure of the control system.
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.00

5.2.4. na_0037: Use of single variable variant conditionals

ID: Title	na_0037: Use of single variable variant conditionals
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
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  **Correct**: Single variable compound conditions



Incorrect: Multiple variables, compound conditions



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 ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☑ Simulation
See also	na_0036 Default variant
Last Change	V3.00

5.2.5. na_0020: Number of inputs to variant subsystems

ID: Title	na_0020: Number of inputs to variant subsystems
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	Simulink requires variant subsystems to have the same number of inputs. However, the variant subsystem might not use all of the inputs. In these instances, terminate the unused inputs with the Terminator block.
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☑ Simulation
Last Change	V3.00

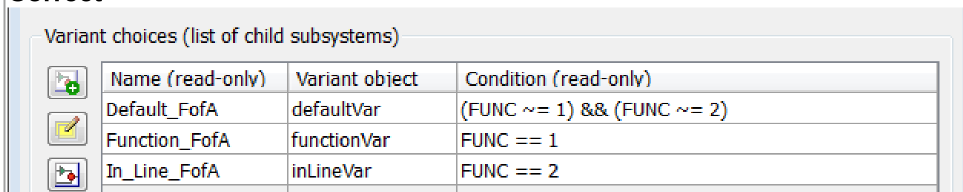
5.2.6. na_0036: Default variant

ID: Title	**na_0036 Default variant**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	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.

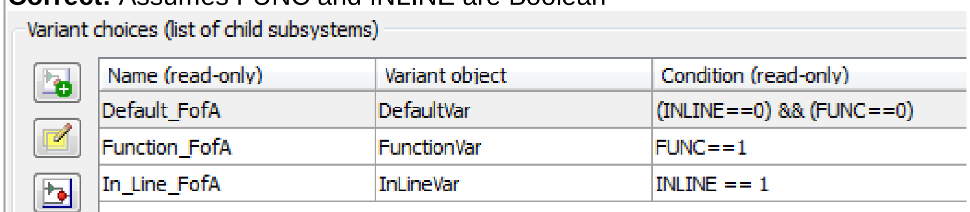
Correct



Variant choices (list of child subsystems)

Name (read-only)	Variant object	Condition (read-only)
Default_FofA	defaultVar	(FUNC ~= 1) && (FUNC ~= 2)
Function_FofA	functionVar	FUNC == 1
In_Line_FofA	inLineVar	FUNC == 2

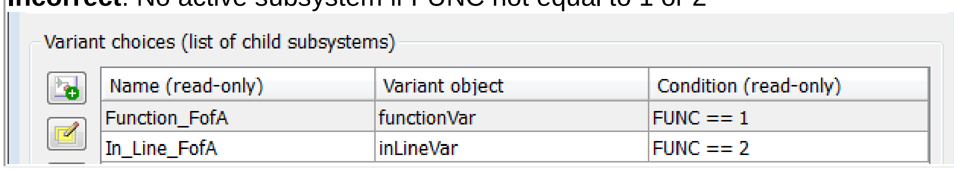
Correct: Assumes FUNC and INLINE are Boolean



Variant choices (list of child subsystems)

Name (read-only)	Variant object	Condition (read-only)
Default_FofA	DefaultVar	(INLINE==0) && (FUNC==0)
Function_FofA	FunctionVar	FUNC==1
In_Line_FofA	InLineVar	INLINE == 1

Incorrect: No active subsystem if FUNC not equal to 1 or 2



Variant choices (list of child subsystems)

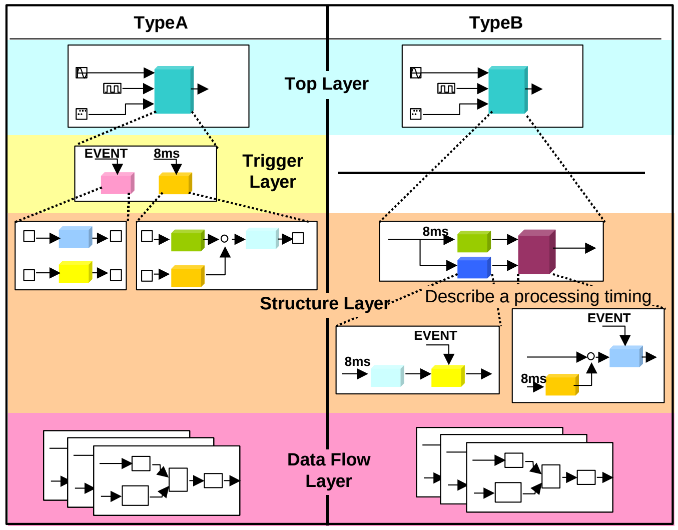
Name (read-only)	Variant object	Condition (read-only)
Function_FofA	functionVar	FUNC == 1
In_Line_FofA	inLineVar	FUNC == 2

Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☑ Simulation
Last Change	V3.00

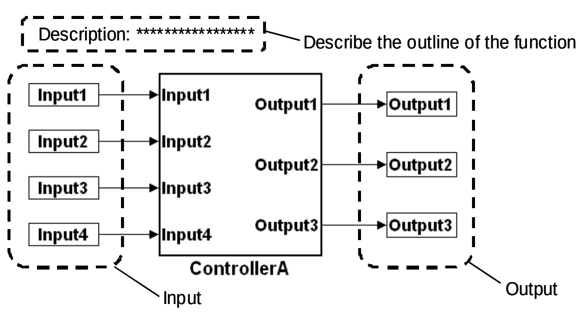
5.3. J-MAAB Model Architecture Decomposition

5.3.1. jc_0301: Controller model

ID: Title	jc_0301: Controller model
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	Control models are organized using the following hierarchical structure. Details on each layer are provided in the latter rules. • Top layer / root level • Trigger layer • Structure layer • Data flow layer

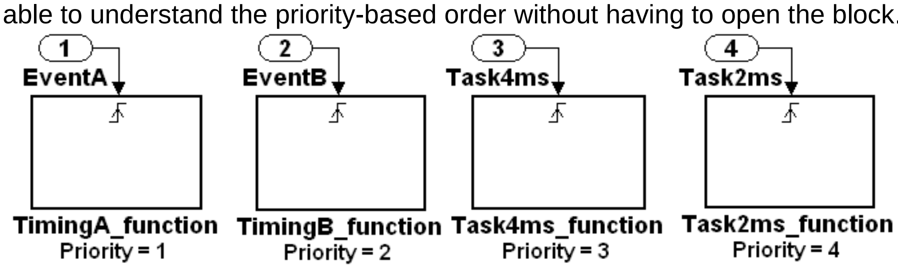
	Use of the Trigger level is optional. In the diagram below “Type A” shows the use of a trigger level while “Type B“ shows a model without a trigger level. 
Rationale	☐ Readability ☑ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

5.3.2. jc_0311: Top layer / root level

ID: Title	**jc_0311: Top layer / root level**
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
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	☐ Readability ☑ Workflow ☐ Verification and Validation

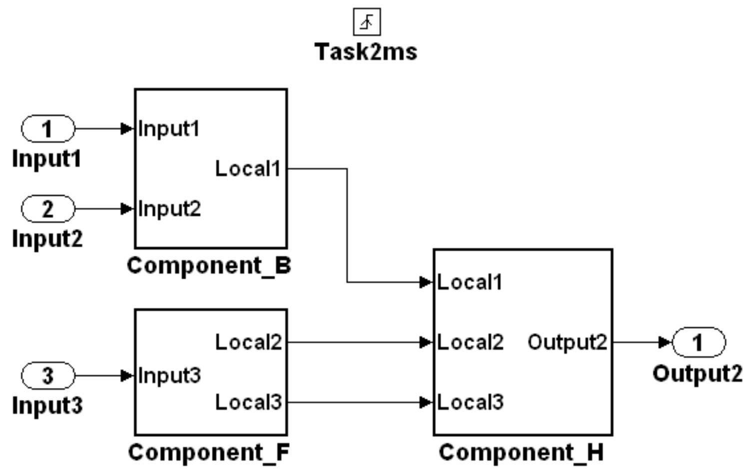
	☐ Simulation ☐ Code Generation
Last Change	V2.00

5.3.3. jc_0321: Trigger layer

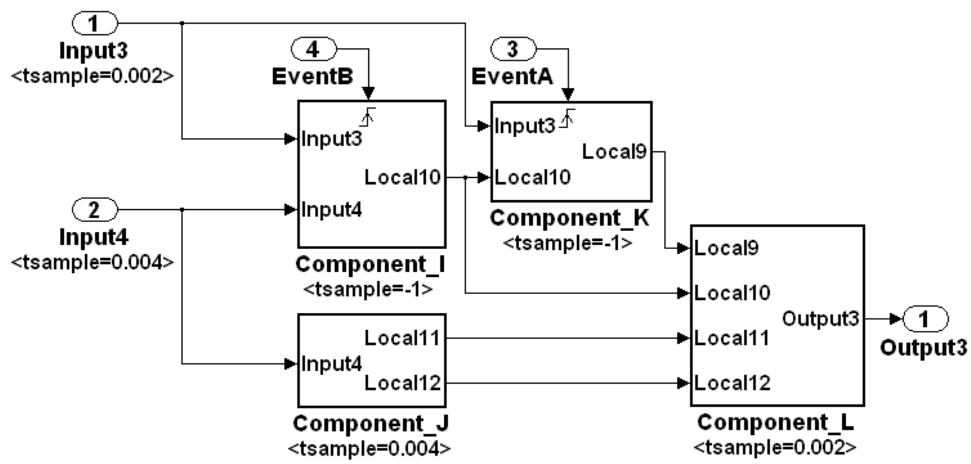
ID: Title	**jc_0321: Trigger layer**
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	A trigger layer indicates the processing timing by using Triggered Subsystem or Function-Call Subsystem. • The blocks should set Priority, if needed. • The priority value must be displayed as a Block Annotation. The user should be able to understand the priority-based order without having to open the block.  Trigger Layer Example
Rationale	☑ Readability ☐ Verification and Validation ☑ Workflow ☐ Simulation ☑ Code Generation
Last Change	V2.00

5.3.4. jc_0331: Structure layer

ID: Title	**jc_0331: Structure layer**
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	Describe a structure layer like the following description example. • In case of Type B, specify sample time at an Inport block or a Subsystem to define task time of the Subsystem. • In case of Type B, use a Block Annotation at an Inport block or a Subsystem and display sample time to clarify task time of the Subsystem A subsystem of a structure layer should be an atomic subsystem.



Structured Layer Example (Type A: No description of processing timing)

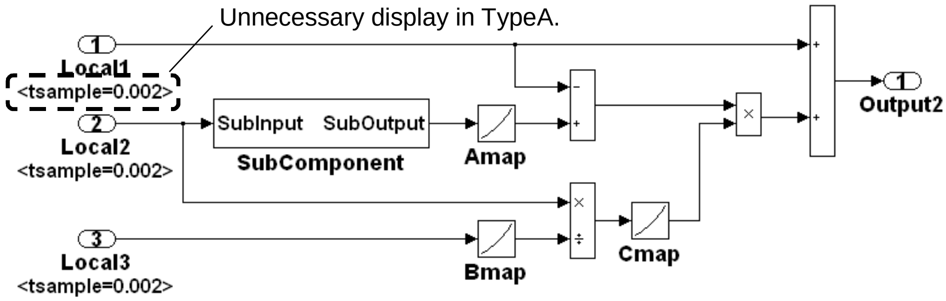


Structured Layer Example (Type B: Description of processing timing)

Rationale	☑ Readability ☑ Workflow ☐ Simulation ☐ Verification and Validation ☑ Code Generation
Last Change	V2.00

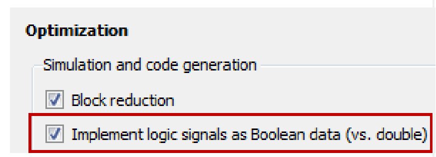
5.3.5. jc_0341: Data flow layer

ID: Title	jc_0341: Data flow layer
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	Describe a data flow layer as in the following example. • In 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	☐ Readability ☑ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

6.Model Configuration Options

6.1.1. jc_0011: Optimization parameters for Boolean data types

<table>
<tr><td>ID:Title</td><td>jc_0011: Optimization parameters for Boolean data types</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>na_0002: Appropriate implementation of fundamental logical and numerical operations</td></tr>
<tr><td>Description</td><td>The optimization option for Boolean data types must be enabled (on).
<table>
<tr><td>Path</td><td>Parameter</td><td>Image</td></tr>
<tr><td>Configuration Parameters > Optimization > Simulation and code generation > Implement logic signals as Boolean data (vs. double)</td><td>BooleanDataType</td><td>
</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☐ Readability
☑ Workflow
☐ Simulation
☐ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

6.1.2. jc_0021: Model diagnostic settings

ID:Title	**jc_0021: Model diagnostic settings**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	

<table>
<tr><td>Description</td><td>The following diagnostics must be enabled. An enabled diagnostic is set to either “warning” or “error”. Setting the diagnostic option to “none” is not permitted. Diagnostics that are not listed can 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</td></tr>
<tr><td>Rationale</td><td>☐ Readability
☑ Workflow
☐ Simulation
☐ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

7. Simulink

7.1. Diagram Appearance

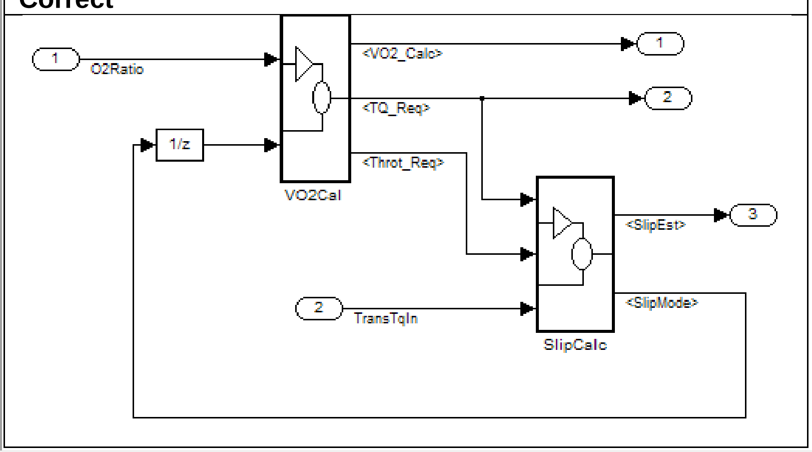
7.1.1. na_0004: Simulink model appearance

ID: Title	**na_0004 Simulink model appearance**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The model appearance settings should conform to the following guidelines when the model is released. The user is free to 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 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 Non-scalar Lines: checked
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

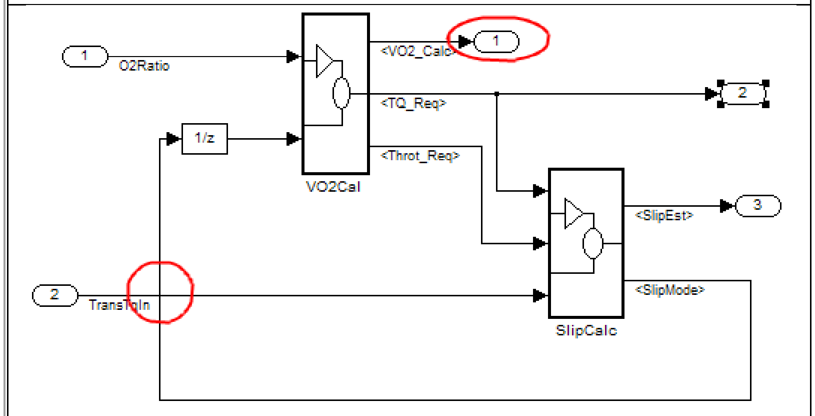
7.1.2. db_0043: Simulink font and font size

ID: Title	**db_0043: Simulink font and font size**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	All text elements (block names, block annotations and signal labels) except free text annotations within a model must have the same font style and font size. Fonts and font size should be selected for legibility. Note: The selected font should be directly portable (e.g. Simulink/Stateflow default font) or convertible between platforms (e.g. Arial/Helvetica 12pt).
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.1.3. db_0042: Port block in Simulink models

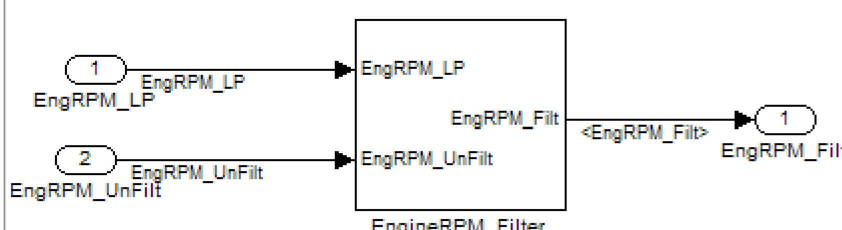
<table>
<tr><td>ID: Title</td><td>db_0042: Port block in Simulink models</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>In a Simulink model, the ports comply with the following rules:
• Inports should be placed on the left side of the diagram, but they can be moved in to prevent signal crossings.
• Outports should be placed on the right side, but they can be moved in to prevent signal crossings.
• Duplicate Inports can be used at the subsystem level if required, but should be avoided, if possible.
○ Do not use duplicate Inports at the root level.
Correct


</td></tr>
</table>

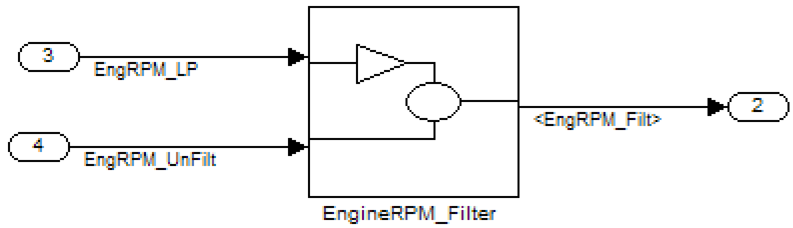
<table>
<tr><td></td><td>Incorrect


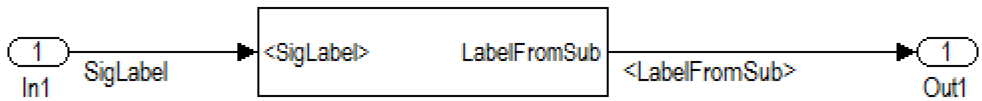
Notes on the incorrect model
Inport 2 should be moved in so it does not cross the feed back loop lines.
Outport 1 should be moved to the right hand side of the diagram.</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

7.1.4. na_0005: Port block name visibility in Simulink models

<table>
<tr><td>ID: Title</td><td>na_0005: Port block name visibility in Simulink models</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>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 select one of the following alternatives to enforce.
Organizationally-Scoped Alternatives (follow one practice):
The name of an Inport or Outport is not hidden. ("Format / Hide Name" is not allowed.)


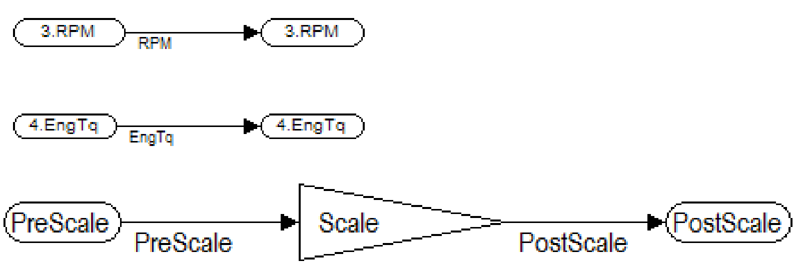
<ol start="2">
The name of an Inport or Outport must be hidden. ("Format / Hide Name" is used.)
Exception: inside library subsystem blocks, the names may not be hidden.</td></tr>
</table>

<table>
<tr><td></td><td>

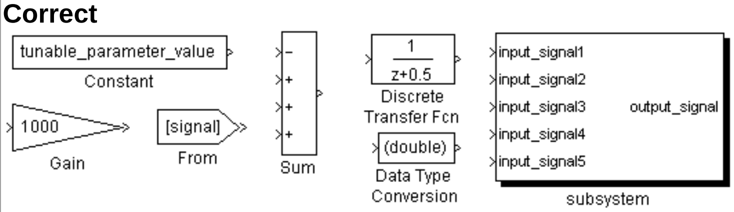
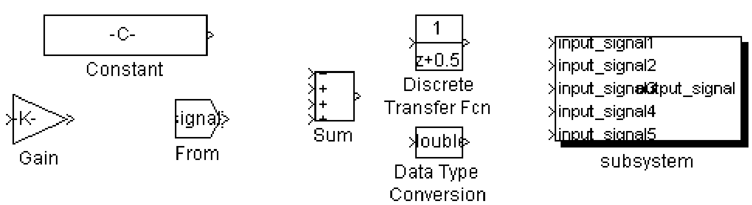
Correct: Use of signal label

</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

7.1.5. jc_0081: Icon display for Port block

<table>
<tr><td>ID: Title</td><td>jc_0081: Icon display for Port block</td></tr>
<tr><td>Priority</td><td>recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>R14 and later</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The Icon display setting should be set to Port number for Inport and Outport blocks.
Correct

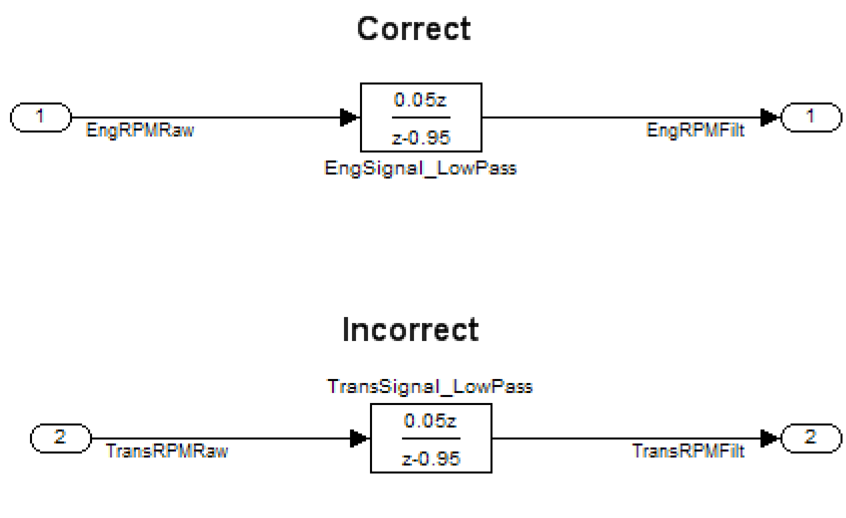

Incorrect

</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

7.1.6. jm_0002: Block resizing

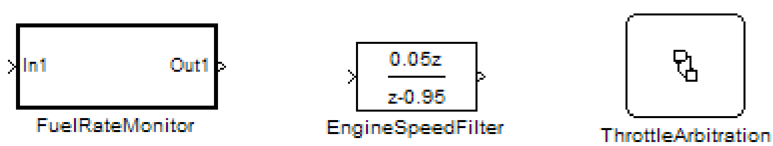
ID: Title	**jm_0002: Block resizing**
Priority	mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	All blocks in a model must be sized such that their icon is completely visible and recognizable. In particular, any text displayed (for example, tunable parameters, filenames, or equations) in the icon must be readable. This guideline requires resizing of 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 block icon of a subsystem block so that all of the input and output names within it are readable. In such cases, you may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. If you do this, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block. **Correct**  **Incorrect** 
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

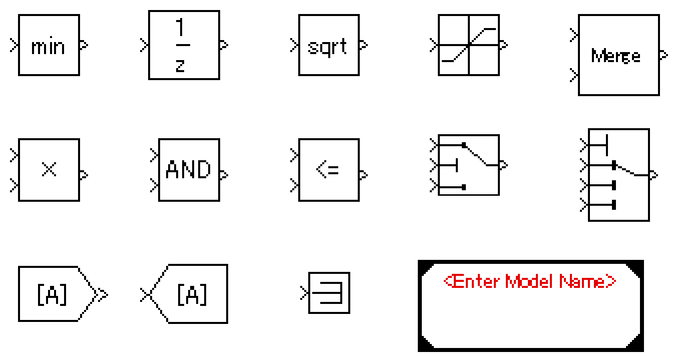
7.1.7. db_0142: Position of block names

ID: Title	**db_0142: Position of block names**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	If shown the name of each block should be placed below the block.

	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.1.8. jc_0061: Display of block names

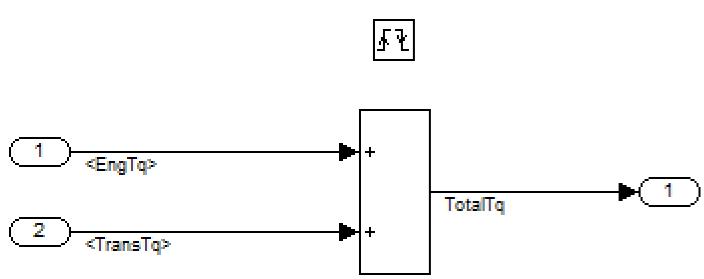
ID: Title	**jc_0061: Display of block names**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	• Display a block name when it provides descriptive information.  • The block name should not be displayed if the block function is known and understood from the block appearance.

<table>
<tr><td></td><td>
</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

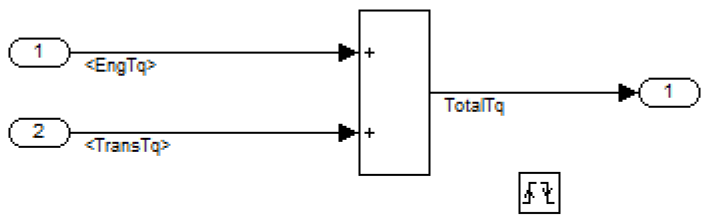
7.1.9. db_0146: Triggered, enabled, conditional Subsystems

<table>
<tr><td>ID: Title</td><td>db_0146: Triggered, Enabled, Conditional Subsystems</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>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:

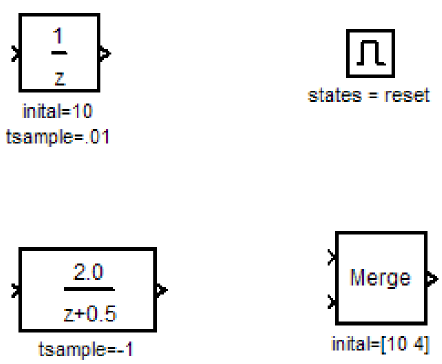
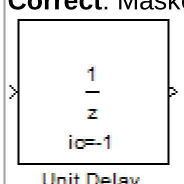
Enable
For Iterator
Action Port
Switch Case Action
Trigger
While Iterator

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


Incorrect</td></tr>
</table>

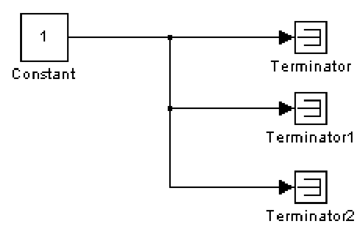
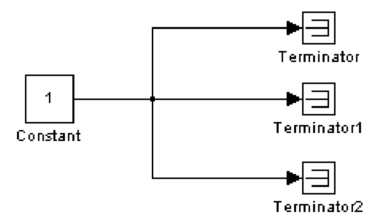
	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

7.1.10. db_0140: Display of basic block parameters

ID: Title	**db_0140: Display of basic block parameters**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Important block parameters modified from the default values should be displayed. Note: The attribute string is one method to support the display of block parameters. The block annotation tab allows you to add the desired attribute information. As of R2011b, masking basic blocks is a supported method for displaying the information. This method is allowed if the base icon is distinguishable. **Correct**  **Correct**: Masked block 
Rationale	☑ Readability ☐ Workflow ☐ Verification and Validation

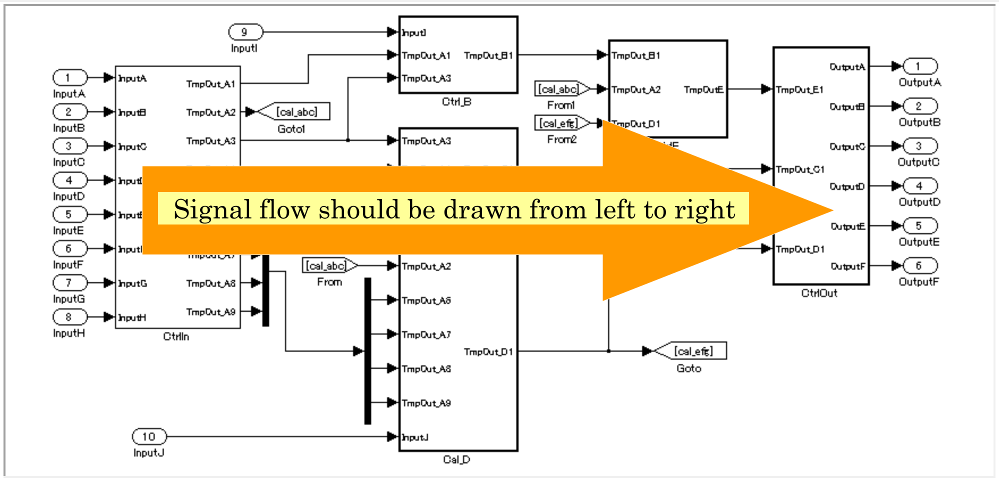
	☐ Simulation ☐ Code Generation
Last Change	V2.20

7.1.11. db_0032: Simulink signal appearance

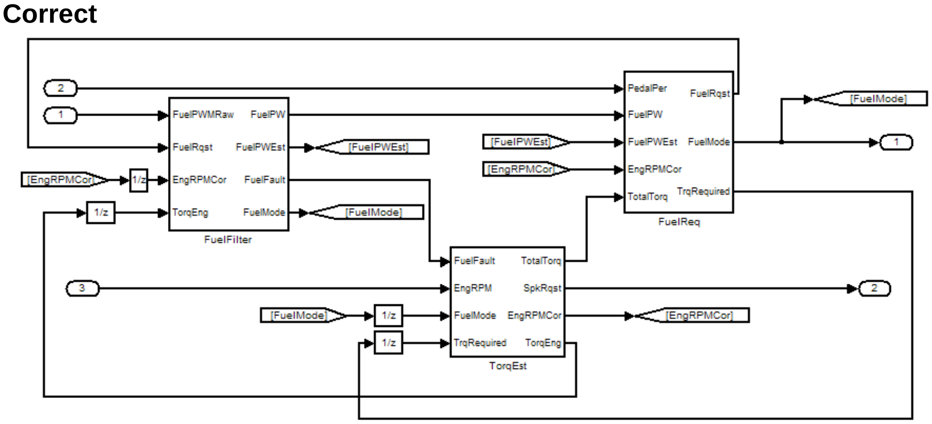
ID: Title	**db_0032: Simulink signal appearance**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Signal lines • Should not cross each other, if possible. • Are drawn with right angles. • Are not drawn one upon the other. • Do not cross any blocks. • Should not split into more than two sub lines at a single branching point. **Correct**  **Incorrect** 
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

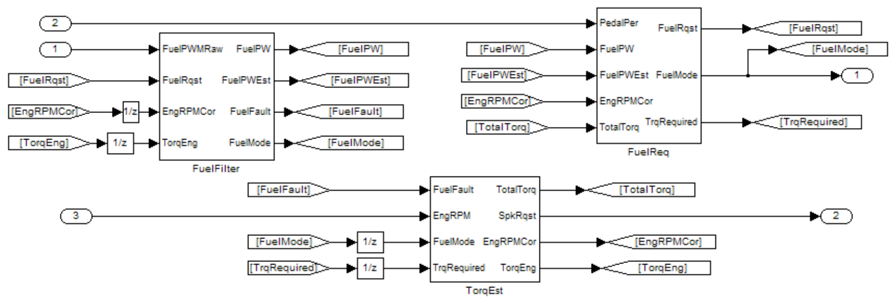
7.1.12. db_0141: Signal flow in Simulink models

ID: Title	**db_0141: Signal flow in Simulink models**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	• Signal flow in a model is from left to right. • Exception: Feedback loops • Sequential blocks or subsystems are arranged from left to right. • Exception: Feedback loops • Parallel blocks or subsystems are arranged from top to bottom.

	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.1.13. 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 Version	All
Prerequisites	
Description	• Visual depiction of signal flow must be maintained between subsystems. • Use of Goto and From blocks is allowed if: • At least one signal line is used between connected subsystems. • Subsystems connected in a feed-forward and feedback loop have at least one signal line for each direction. • Using Goto and From blocks to create buses or connect inputs to merge blocks are exceptions to this rule. **Correct** 

	Incorrect 
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.1.14. na_0032: Use of Merge Blocks

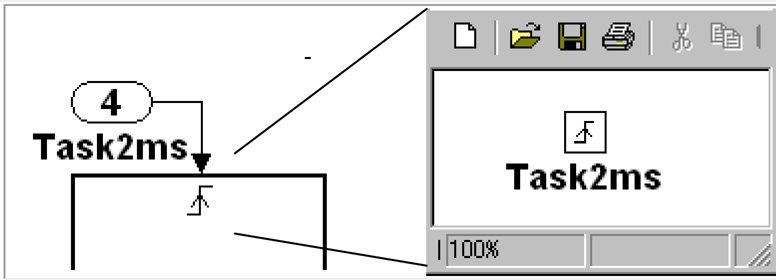
ID: Title	**na_0032: Use of merge blocks**
Priority	Strongly Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	None
Description	When using merge blocks: • Signals entering a merge block must not branch off to any other block. • With buses: ○ All buses must be identical. This includes: ▪ Number of elements ▪ Element names ▪ Element order ▪ Element data type ▪ Element size ○ Buses must be either all virtual or all non-virtual. ○ All bus lines entering a merge block must not branch off to any other block.
Rationale	☐ Readability ☑ Workflow ☐ Simulation ☐ Verification and Validation ☑ Code Generation
See Also	jh_0109: Merge blocks
Last Change	V3.00

7.1.15. jm_0010: Port block names in Simulink models

ID: Title	jm_0010: Port block names in Simulink models
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	db_0042: Ports in Simulink models na_0005: Port block name visibility in Simulink models
Description	For some items, it is not possible to define a single approach applicable to all organizations' internal processes However, within a given organization, it is important to follow a **single** consistent approach is followed. An organization applying the guidelines must select **one** of these alternatives. 1. Names of Inport blocks and Outport blocks must match the corresponding signal or bus names. **Exceptions:** ○ When any combination of an Inport block, an Outport block, and any other block have the same block name, a suffix or prefix should be used on the Inport and Outport blocks. ○ One common suffix / prefix is "_in" for Inports and "_out" for Outports. ○ Any suffix or prefix can be used on the ports, however the selected prefix should be consistent. ○ Library blocks and reusable subsystems that encapsulate generic functionality. 2. When the names of Inport and Outport blocks are hidden, apply a consistent naming practice for the blocks. Suggested practices include leaving the names as their default names (for example, Out1), giving them the same name as the associated signal or giving them a shortened or mangled version of the name of the associated signal.
Rationale	☑ Readability ☐ Verification and Validation ☑ Workflow ☑ Code Generation ☑ Simulation
Last Change	V2.00

7.1.16. 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 Version	All
Prerequisites	
Description	For Trigger port blocks and Enable port blocks, match the name of the signal triggering the subsystem. • The block name should match the name of the signal triggering the subsystem.

	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☑ Code Generation
Last Change	V2.00

7.2. Signals

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

You use signal labels to make model functionality more understandable from the Simulink diagram. You can also use them to control the variable names used in simulation and code generation. Enter signal labels only once (at the point of signal origination). Often, you may also 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, a new name should be associated with it.

Unless explicitly stated otherwise, the following naming rules apply to all types of signals.

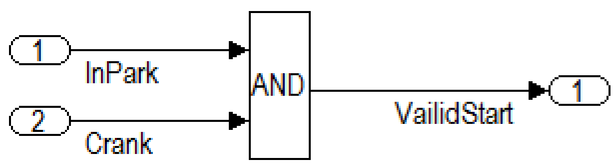
7.2.1. na_0008: Display of labels on signals

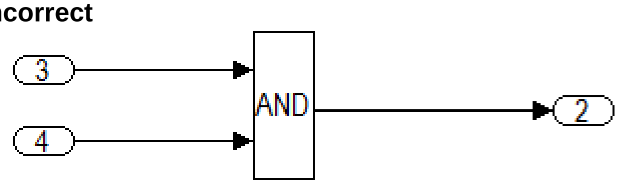
ID: Title	**na_0008: Display of labels on signals**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	A label must be displayed on a signal originating from the following blocks: • Inport block • From block (block icon exception applies – see Note below) • Subsystem block 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 non transformative

<table>
<tr><td></td><td>operation):

• Outport block
• Goto block
• Data Store Write block
• Bus Creator block
• Mux block
• Subsystem block
• Chart block

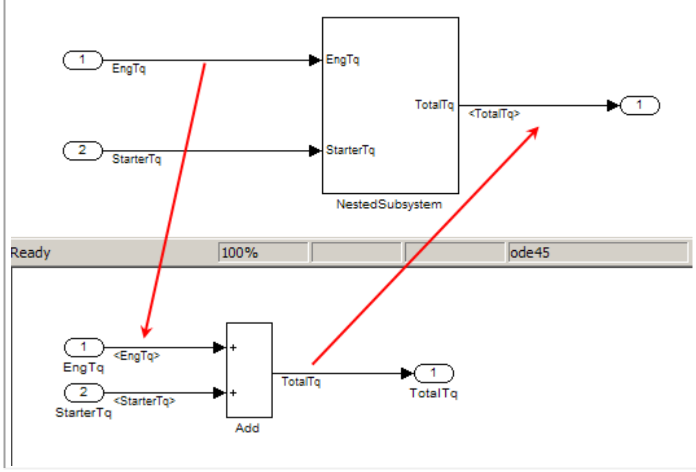
Note: Block icon exception (applicable only where called out above): If the signal label is visible in the originating block icon display, the connected signal does not need not to have the label displayed, unless the signal label is needed elsewhere due to a destination-based rule.

Correct


Incorrect

</td></tr>
<tr><td>Rationale</td><td>☑ Readability ☑ Verification and Validation
☑ Workflow ☑ Code Generation
☑ Simulation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

7.2.2. na_0009: Entry versus propagation of signal labels

ID: Title	**na_0009: Entry versus propagation of signal labels**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	na_0008: Display of labels on signals

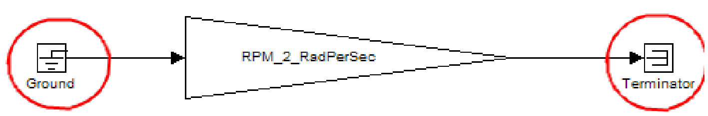
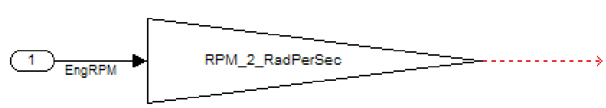
Description	If a label is present on a signal, the following rules define whether that label shall be created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the ‘<’ character). 1. Any displayed signal label must be *entered* for signals that: a. Originate from an Inport at the Root (top) Level of a model b. 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 shall be considered to be included among the blocks that perform transformative operations.) 2. Any displayed signal label must be *propagated* for signals that: a. 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. b. Originate from a basic block that performs a non-transformative operation c. Originate from a Subsystem or Stateflow chart block **Exception:** If the connection originates from the output of a library subsystem block instance, a new label may be *entered* on the signal to accommodate reuse of the library block. 
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☑ Simulation
Last Change	V2.00

7.2.3. 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 Version	All
Prerequisites	
Description	The labels must be visually associated with the corresponding signal and not

	overlap other labels, signals, or blocks. Labels should be located consistently below horizontal lines and close to the corresponding source or destination block.
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.2.4. db_0081: Unconnected signals, block inputs and block outputs

ID: Title	db_0081: Unconnected signals and block inputs / outputs
Priority	Mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	A system must not have any: • Unconnected subsystem or basic block input. • Unconnected subsystem or basic block outputs • Unconnected signal lines In addition: • An otherwise unconnected input should be connected to a ground block • An otherwise unconnected output should be connected to a terminator block **Correct**  **Incorrect** 
Rationale	☑ Readability ☑ Workflow ☐ Simulation ☑ Verification and Validation ☐ Code Generation
Last Change	V2.00

7.3. Block Usage

7.3.1. na_0003: Simple logical expressions in If Condition block

ID: Title	na_0003: Simple logical expressions in If Condition block

<table>
<tr><td>Priority</td><td>mandatory</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>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 here to be one of the following:

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

Exception:

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

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

Examples of Acceptable Exceptions:

u1 || u2 || u3 || u4 || u5
u1 && u2 && u3 && u4

Examples of Primary Expressions:

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

Examples of Acceptable Logical Expressions:

u1 || u2
(u1 > 0) && (u1 < 20)
(u1 > 0) && (u2 < u3)
(u1 > 0) && (~u2)
(EngineState.ENGINE_RUNNING) && (PRNDLState.PRNDL_PARK)

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

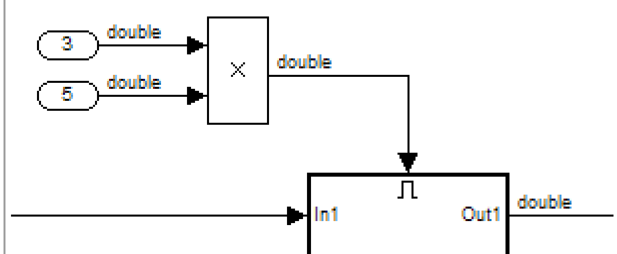
Examples of unacceptable logical expressions include:

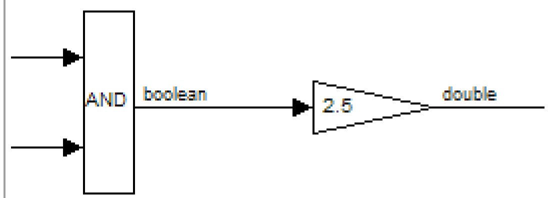
u1 && u2 || u3 (too many primary expressions)

</td></tr>
</table>

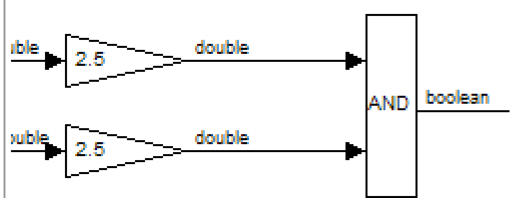
	• u1 && (u2 || u3) (unacceptable operator within primary expression) • (u1 > 0) && (u1 < 20) && (u2 > 5) (too many primary expressions that are not inputs) • (u1 > 0) && ((2*u2) > 6) (unacceptable operator within primary expression)
Rationale	☑ Readability ☐ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.20

7.3.2. 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 Version	All
Prerequisites	
Description	• Blocks that are intended to perform numerical operations must not be used to perform logical operations. **Incorrect**  • A logical output should never be directly connected 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. • This guideline for logical operations also applies to enumerated data types. **Incorrect**

<table>
<tr><td></td><td>

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


</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☑ Workflow
☐ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V3.00</td></tr>
</table>

7.3.3. jm_0001: Prohibited Simulink standard blocks inside controllers

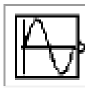
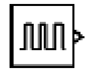
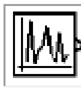
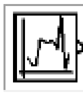
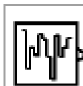
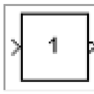
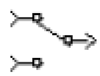
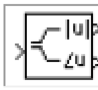
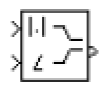
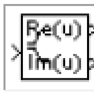
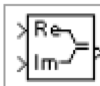
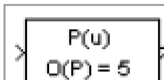
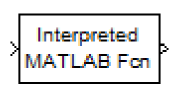
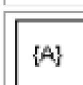
<table>
<tr><td>ID: Title</td><td>jm_0001: Prohibited Simulink standard blocks inside controllers</td></tr>
<tr><td>Priority</td><td>mandatory</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>

Control algorithm models must be designed from discrete blocks.
The MathWorks “Simulink Block Data Type Support” table provides a list of blocks that support production code generation.

Use blocks that are listed as “Code Generation Support”.
Do not use blocks that are 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

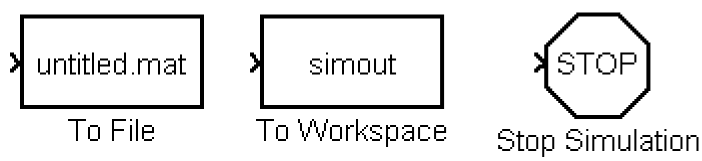
</td></tr>
</table>

Sources are not allowed:	
Sine Wave	
Pulse Generator	
Random Number	
Uniform Random Number	
Band-Limited White Noise	
Additional blocks that are not allowed: The MAAB Style guide group recommends not using the following blocks. The list can be extended by individual companies.	
Slider Gain	
Manual Switch	
Complex to Magnitude-Angle	
Magnitude-Angle to Complex	
Complex to Real-Imag	
Real-Imag to Complex	
Polynomial	
MATLAB Fcn(1)	
Goto Tag Visibility	
Probe	

Notes	(1) In R2011a, the MATLAB Fnc was renamed the Interpreted MATLAB

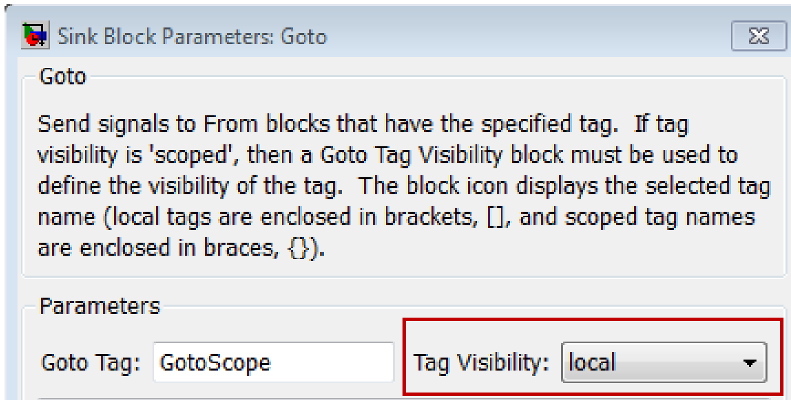
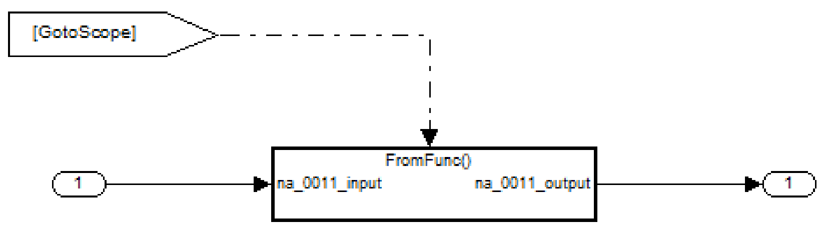
	Function
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

7.3.4. hd_0001: Prohibited Simulink sinks

ID: Title	**hd_0001: Prohibited Simulink sinks**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	**Control algorithm models must be designed from discrete blocks.** **The following sinks blocks are not allowed:** To File To Workspace Stop Simulation 
Note	Simulink Scope and Display blocks are allowed in the model diagram. Consider using the Simulink Signal logging and Signal and Scope Manager for data logging and viewing requirements.
Rationale	☐ Readability ☐ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

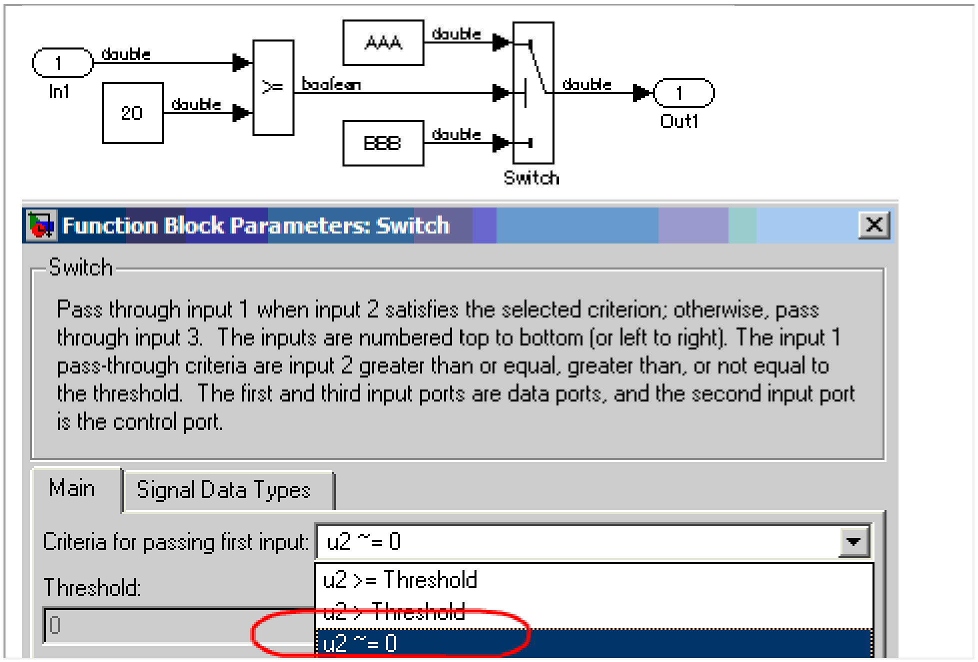
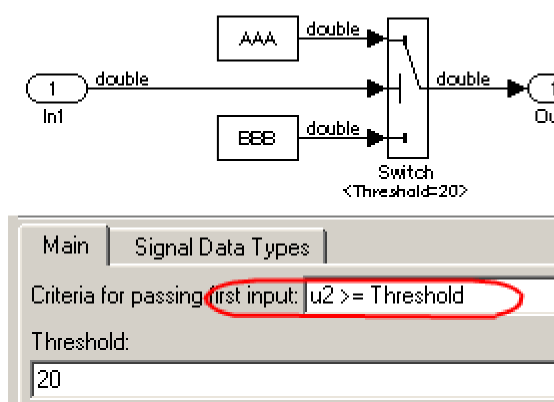
7.3.5. na_0011: Scope of Goto and From blocks

ID: Title	**na_0011: Scope of Goto and From blocks**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	For signal flows, the following rules apply: • From and Goto blocks must use local scope. Note: Control flow signals may use global scope. Control flow signals are output from: • Function-call generators • If and Case blocks

	• Function call outputs from MATLAB and Stateflow blocks Control flow signals are identified as dashed lines in the model after updating a Simulink model.  
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

7.3.6. jc_0141: Use of the Switch block

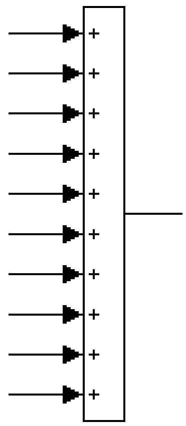
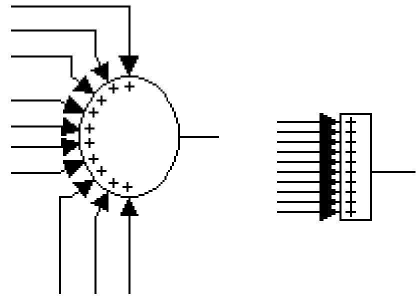
ID: Title	**jc_0141: Use of the Switch block**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	• The switch condition, input 2, must be a Boolean value. • The block parameter “Criteria for passing first input” should be set to u2~=0. **Correct**

	 **Incorrect** 
Rationale	☑ Readability ☑ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

7.3.7. jc_0121: Use of the Sum block

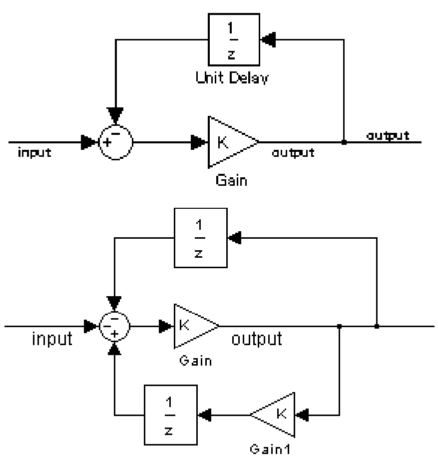
ID: Title	**jc_0121: Use of the Sum block**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Sum blocks should:

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

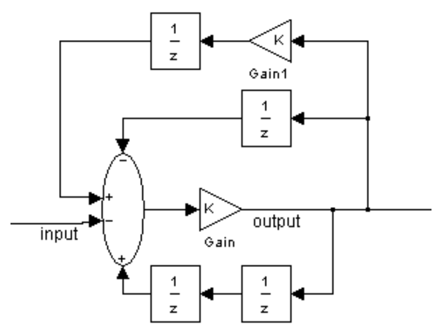
Correct	Incorrect
	

- You may use the round shape in feedback loops.
    - There should be no more then 3 inputs.
    - The inputs may be positioned at 90,180,270 degrees.
    - The output should be positioned at 0 degrees.

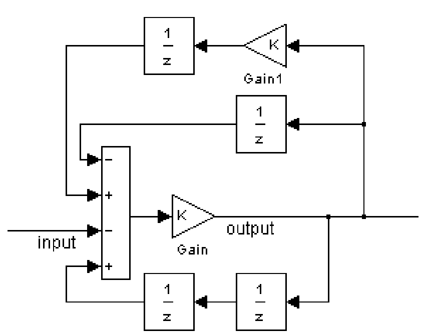
Correct



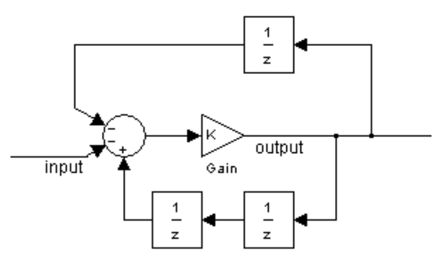
Incorrect



Correct



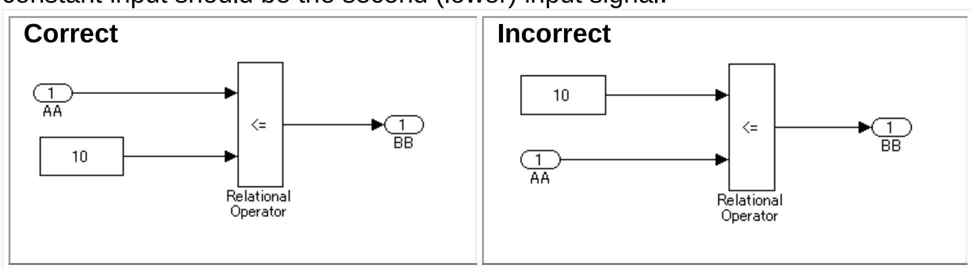
Incorrect



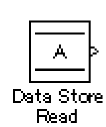
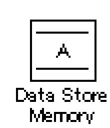
Rationale	☑ Readability ☐ Verification and Validation

	☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

7.3.8. jc_0131: Use of Relational Operator block

ID: Title	**jc_0131: Use of Relational Operator block**
Priority	recommended
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
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. 
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

7.3.9. 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 Version	All
Prerequisites	[jc_0341: Data flow layer]()
Description	Data Store Read —  — Data Store Write —  — Data Store Memory —  • Prohibited in a data flow layer. • Allowed between subsystems running at different rates.
Rationale	• Readability ☐ Verification and Validation ☑ Workflow

	☐ Simulation ☐ Code Generation
Last Change	V2.00

7.4. Block Parameters

7.4.1. db_0112: Indexing

ID: Title	**db_0112: Indexing**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Use a consistent vector indexing method for all blocks. When possible, use zero-based indexing to improve code efficiency. However, since MATLAB blocks do not support zero-based indexing, one-based indexing can be used for models containing MATLAB blocks.
See Also	• cgsl_0101: Zero-based indexing • hisl_0021: Consistent vector indexing
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.20

7.4.2. na_0010: Grouping data flows into signals

ID: Title	**na_0010: Grouping data flows into signals**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	<u>Vectors</u> 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. <u>Busses</u> Signals that do not meet criteria for us as a vector, as described above, must only be grouped into bus signals. Use Bus selector blocks may only be used with a bus signal input; they must not be used to extract scalar signals from vector signals.

Examples

Some examples of vector signals include:

Vector type	**Size**
Row vector	[1 n]
Column vector	[n 1]
Wheel speed vector	[1 Number of wheels]
Cylinder vector	[1 Number of cylinders]
Position vector based on 2-D coordinates	[1 2]
Position vector based on 3-D coordinates	[1 3]

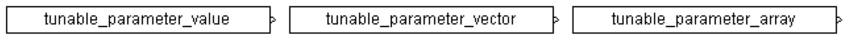
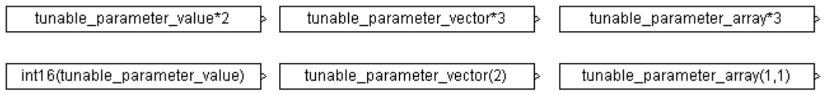
Some examples of bus signals include:

Bus Type	**Elements**
Sensor Bus	Force Vector [Fx, Fy, Fz]
	Position
	Wheel Speed Vector $[\Theta_{lf}, \Theta_{rf}, \Theta_{lr}, \Theta_{rr}]$
	Acceleration
	Pressure
Controller Bus	Sensor Bus
	Actuator Bus
Serial Data Bus	Coolant Temperature
	Engine Speed, Passenger Door Open

Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.00

7.4.3. db_0110: Tunable parameters in basic blocks

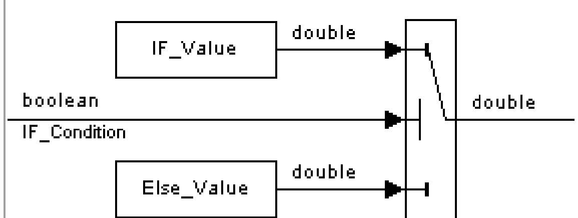
ID: Title	**db_0110: Tunable parameters in basic blocks**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	To insure that a parameter is tunable, it must be entered in a block dialog field: • Without any expression. • Without a data type conversion. • Without selection of rows or columns. **Correct**

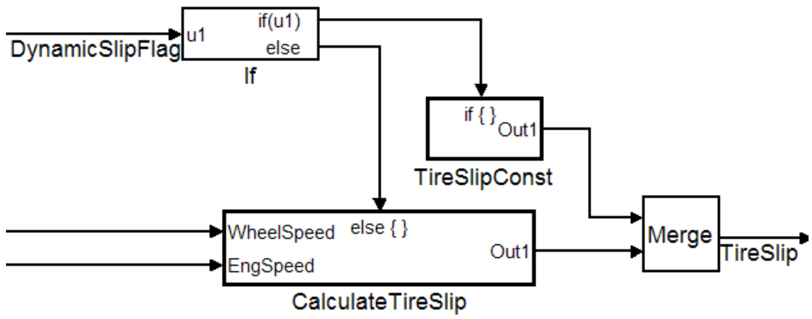
	 **Incorrect** 
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

7.5. Simulink Patterns

The following rules illustrate sample patterns used in Simulink diagrams. As such, they would normally be part of a much larger Simulink diagram.

7.5.1. 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 Version	All
Prerequisites	
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 will maximize simulation efficiency and the efficiency of generated code (Note that even a basic block, for example a table look-up, may require fairly complicated computations.)

<table>
<tr><td></td><td>

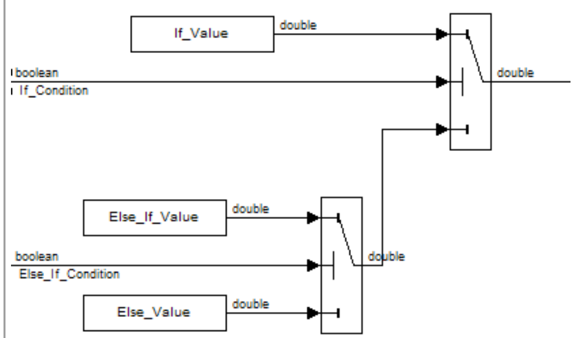
Must be used for modeling if-then-else structures, if the purpose of the construct is to avoid an undesirable numerical computation, such as division by zero.
Should be used for modeling if-then-else structures, if the explicit or implied then or the else action is just to hold the associated output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

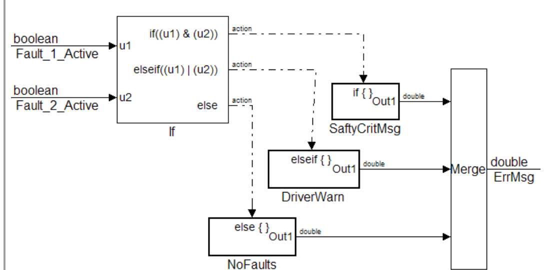
In other cases, the degree of complexity of the then and/or else action computations and the intelligence of the Simulink simulation and code generation engines determine the appropriate construct.
These statements also apply to more complicated nested and cascaded if-then-else structures and case structure implementations. </td></tr>
<tr><td>Rationale</td><td>☑ Readability ☑ Verification and Validation
☐ Workflow ☑ Code Generation
☐ Simulation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

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

<table>
<tr><td>ID: Title</td><td>db_0114: Simulink patterns for If-then-else-if constructs</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns should be used for If-then-else-if constructs within a Simulink model:
<table><tr><td>Equivalent Functionality</td><td>Simulink pattern</td></tr></table></td></tr>
</table>

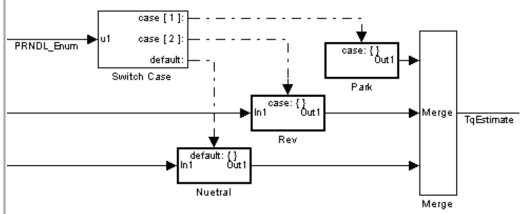
<table>
<tr><td></td><td>
<table>
<tr><td>IF THEN ELSE IF with blocks

if (If_Condition) {
output_signal = If_Value;
}
else if (Else_If_Condition) {
output_signal =
Else_If_Value;
}
else {
output_signal =
Else_Value;
}</td><td>
</td></tr>
<tr><td>IF THEN ELSE IF
with if/then/else
subsystems:
if(Fault_1_Active &
Fault_2_Active)
{
  ErrMsg = SaftyCrit;
}
else if (Fault_1_Active |
Fault_2_Active)

{
  ErrMsg = DriveWarn;
}
else
{
  ErrMsg = NoFaults;
}</td><td>
</td></tr>
</table>
</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

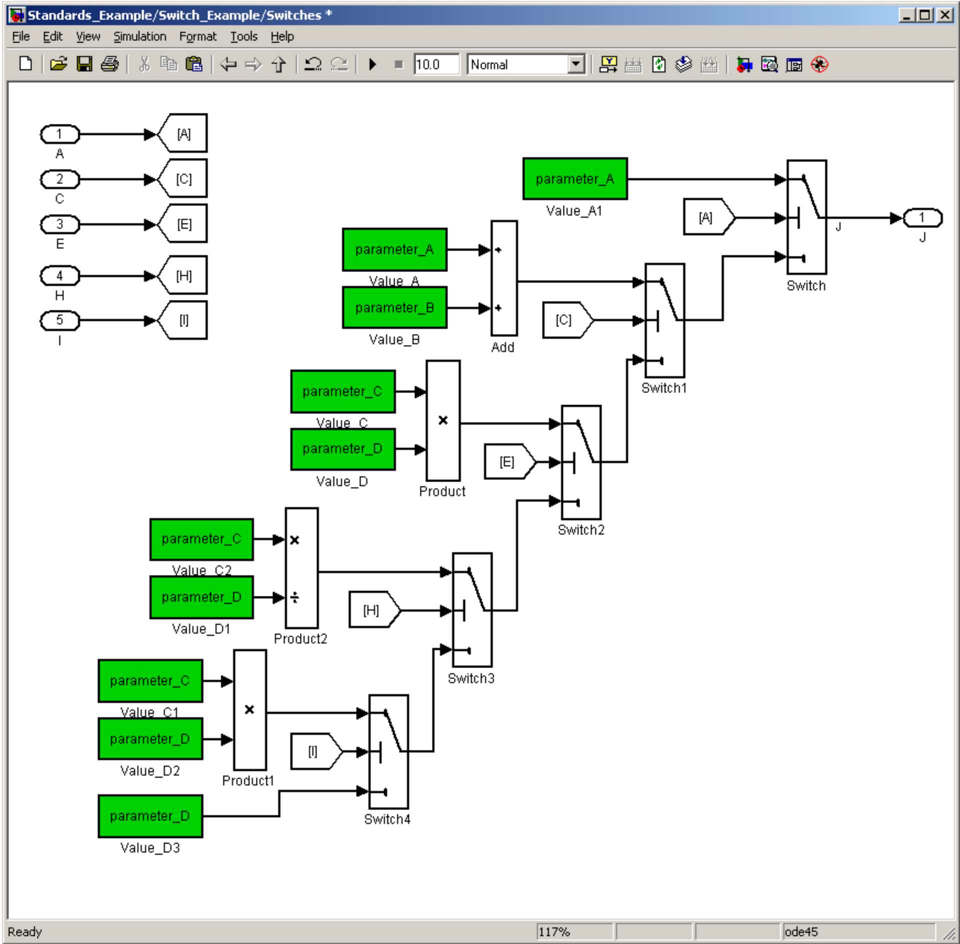
7.5.3. db_0115: Simulink patterns for case constructs

<table>
<tr><td>ID: Title</td><td>db_0115: Simulink patterns for case constructs</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for case constructs within Simulink:
<table><tr><th>Equivalent Functionality</th><th>Simulink Pattern</th></tr></table></td></tr>
</table>

	Case With switch case block *switch (PRNDL_Enum)* *{* *case 1* *TqEstimate = ParkV;* *break;* *case 2* *TqEstimae = RevV;* *break;* *default* *TqEstimate = NeutralV;* *break;* *}*	
Rationale	☑ Readability ☐ Workflow ☐ Simulation	☐ Verification and Validation ☐ Code Generation
Last Change	V2.20	

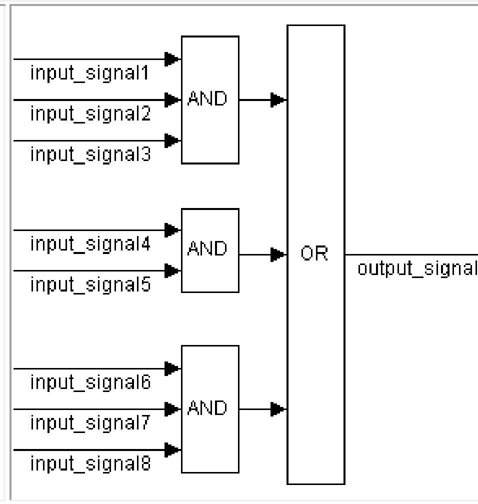
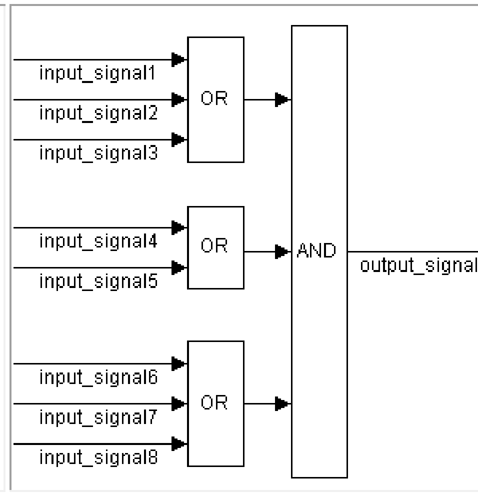
7.5.4. na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches

ID: Title	**na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
MA Check	No
Prerequisites	[na_0012: Use of Switch vs. If-Then-Else Action Subsystem] [db_0114: Simulink patterns for If-then-else-if constructs]
Description	The use of switch constructs should be limited, typically to 3 levels. Replace switch constructs that have more than 3 levels with an If-Then-Else action subsystem construct. **Incorrect**

	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
See also	bn_0003: Use of If-Then-Else Action Subsystem to Replace Multiple Switches
Last Change	V3.00

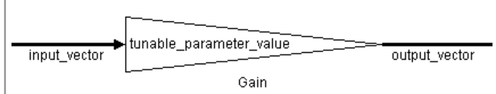
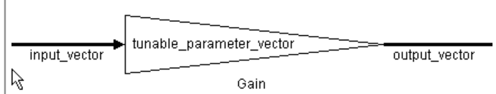
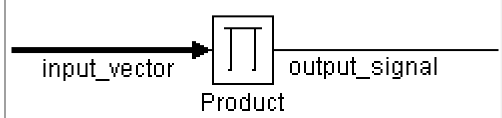
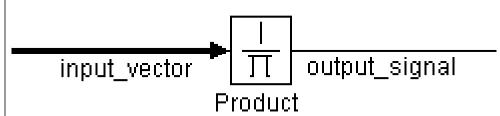
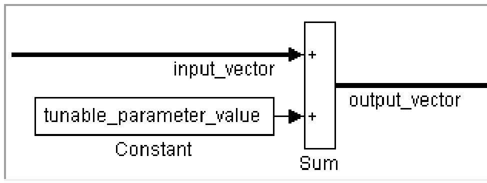
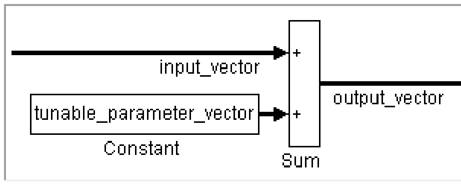
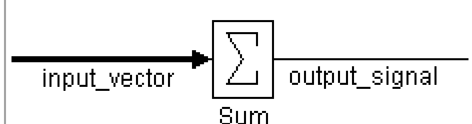
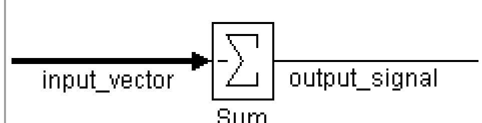
7.5.5. db_0116: Simulink patterns for logical constructs with logical blocks

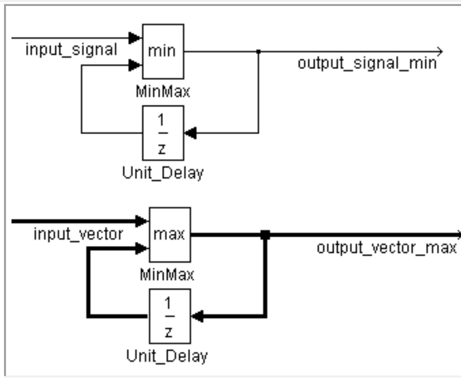
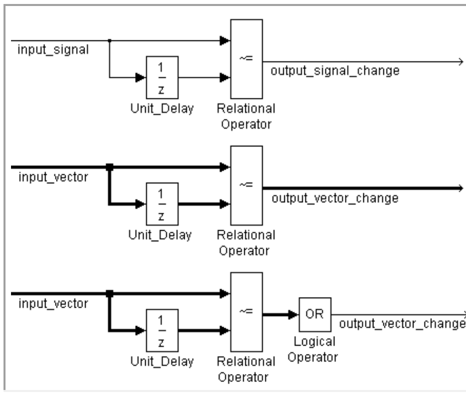
ID: Title	**db_0116: Simulink patterns for logical constructs with logical blocks**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Use the following patterns for logical combinations within a Simulink model: **Equivalent Functionality** | **Simulink pattern**

	Combination of logical signals: conjunctive	
	Combination of logical signals: disjunctive	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation	
Last Change	V1.00	

7.5.6. db_0117: Simulink patterns for vector signals

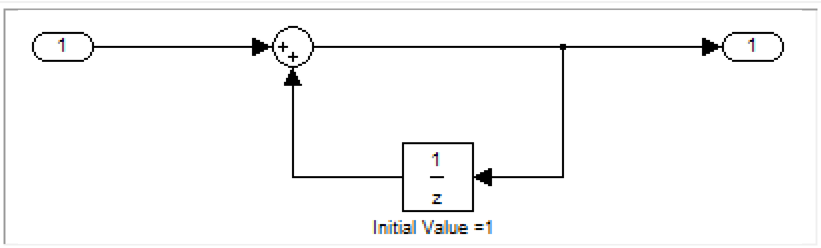
ID: Title	**db_0117: Simulink patterns for vector signals**	
Priority	strongly recommended	
Scope	MAAB	
MATLAB Version	All	
Prerequisites		
Description	Simulink is a vectorizeable modeling language allowing for the direct processing of vector data. The following patterns are used for vector signals within Simulink model:	
	Equivalent Functionality	**Simulink Pattern**

Vector loop: for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_value; }	
Vector loop: for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_vector(i); }	
Vector loop: output_signal = 1; for (i=0; i<input_vector_size; i++) { output_signal = output_signal * input_vector(i); }	
Vector loop: output_signal = 1; for (i=0; i<input_vector_size; i++) { output_signal = output_signal / input_vector(i); }	
Vector loop: for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_value; }	
Vector loop: for (i=0; i<input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_vector(i); }	
Vector loop: output_signal = 0; for (i=0; i<input_vector_size; i++) { output_signal = output_signal + input_vector(i); }	
Vector loop: output_signal = 0; for (i=0; i<input_vector_size; i++) { output_signal = output_signal - input_vector(i); }	

	Minimum or maximum of a signal or a vector over time:	
	Change event of a signal or a vector:	
Rationale	☑ Readability ☐ Workflow ☐ Simulation	☑ Verification and Validation ☑ Code Generation
Last Change	V2.20	

7.5.7. jc_0351: Methods of initialization

ID: Title	**jc_0351: Methods of initialization**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	db_0140: Display of block parameters
Description	**Simple initialization:** • Blocks such as the 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 db_0140. **Example**



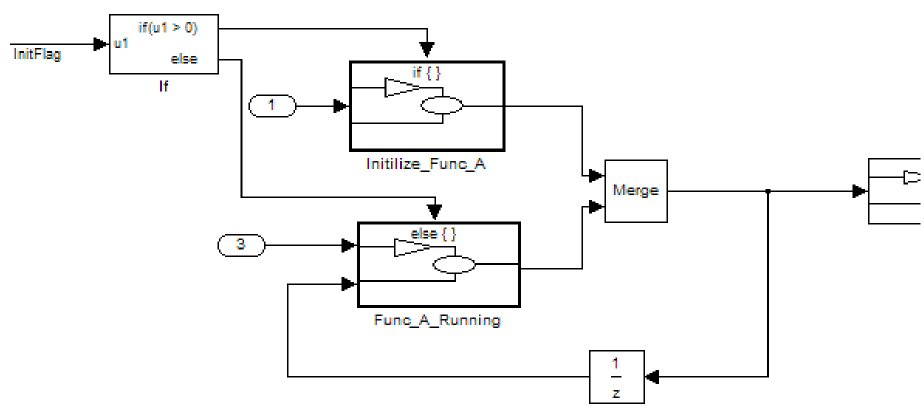
Initialization that requires computation:

The following rules apply for complex initializations:

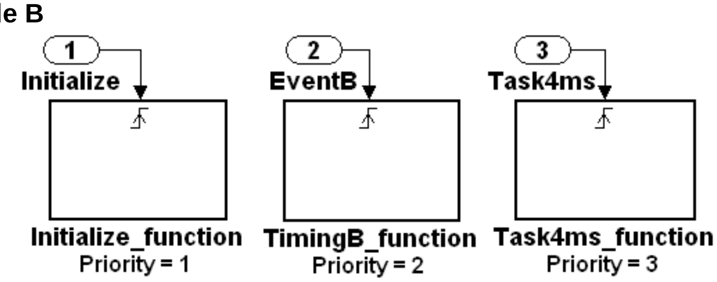
- Initialization should be performed in a separate subsystem.
- Initialization subsystem should have a name that indicates that initialization is performed by the subsystem.

Complex initializations can either be done at a local level (Example A), at a global level (Example B), or a combination of local and global.

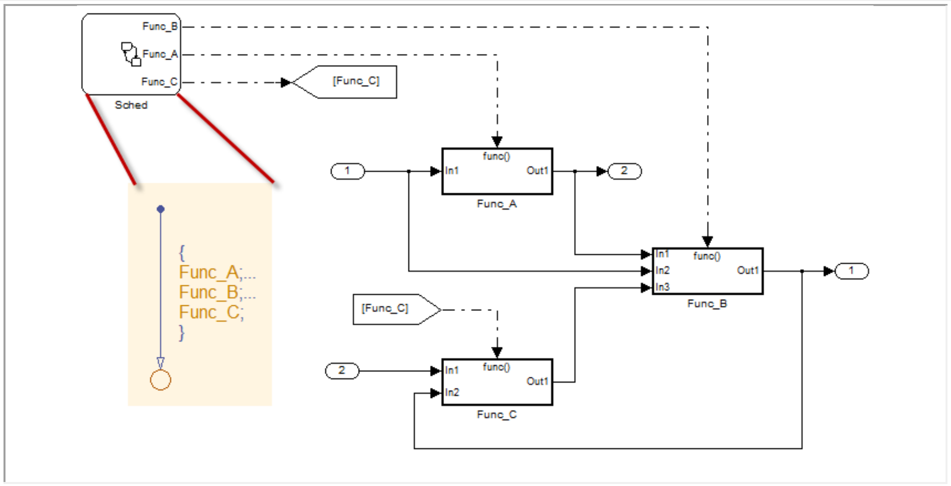
Example A



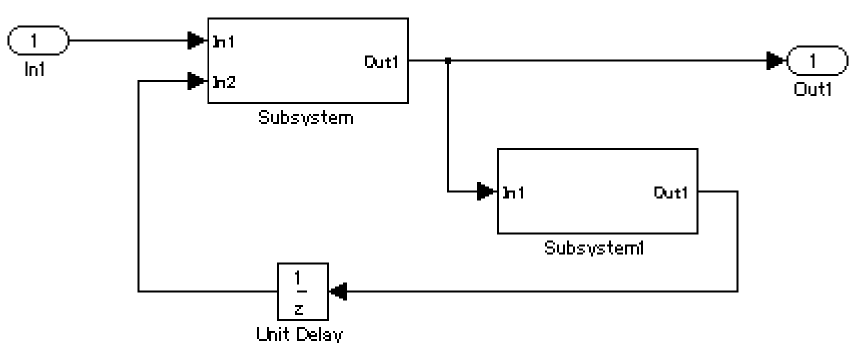
Example B

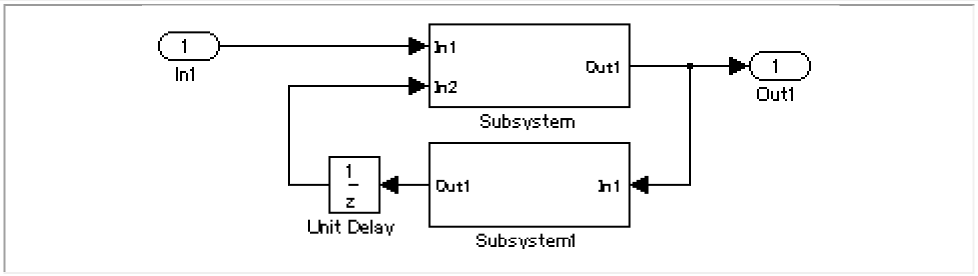


Or

	
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.00

7.5.8. jc_0111: Direction of Subsystem

ID: Title	**jc_0111: Direction of Subsystem**
Priority	strongly recommended
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	Subsystem must not be reversed. **Correct**  **Incorrect**

	
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

8.Stateflow

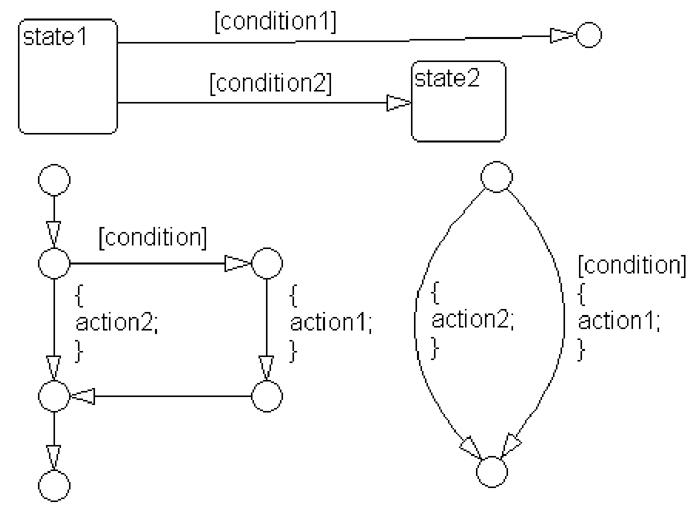
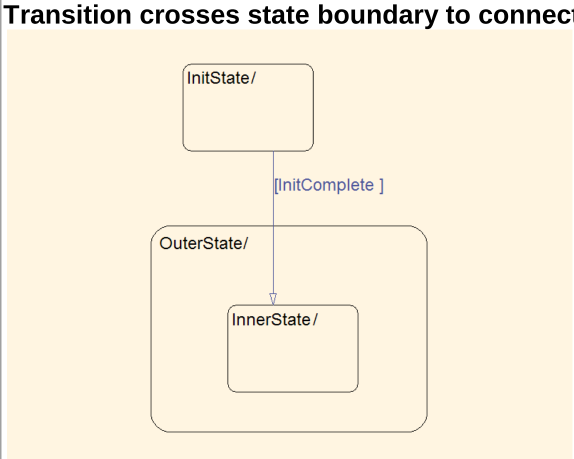
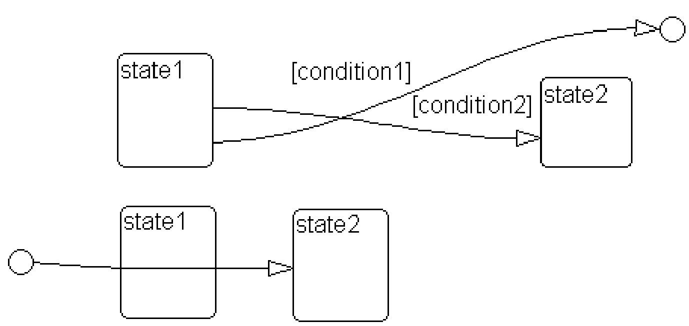
8.1. Chart Appearance

8.1.1. db_0123: Stateflow port names

ID: Title	**db_0123: Stateflow port names**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The name of a Stateflow input/output should be the same as the corresponding signal. Exception: Reusable Stateflow blocks may have different port names.
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☑ Code Generation
Last Change	V1.00

8.1.2. db_0129: Stateflow transition appearance

ID: Title	**db_0129: Stateflow transition appearance**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
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 transitioning to an internal state. Transition labels can be visually associated to the corresponding transition. **Correct**

	 **Correct** **Transition crosses state boundary to connect to substate**  **Incorrect** **Transition crosses each other and transition crosses through state.** 
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

8.1.3. db_0137: States in state machines

ID: Title	db_0137: States in state machines

Priority	mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	db_0149: Flowchart patterns for condition actions
Description	For all levels in a state machine, including the root level, for states with exclusive decomposition, the following rules apply: • At least two exclusive states must exist. • A state cannot have only one substate. • The initial state of every hierarchical level with exclusive states is clearly defined by a default transition. In the case of multiple default transitions, there must always be an unconditional default transition.
Rationale	☑ Readability ☑ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V3.00

8.1.4. db_0133: Use of patterns for Flowcharts

ID: Title	db_0133: Use of patterns for Flowcharts
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	A Flowchart is built with the help of Flowchart patterns (for example, IF-THEN-ELSE, FOR LOOP, and so on): • The data flow is oriented from the top to the bottom. • Patterns are connected with empty transitions.
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

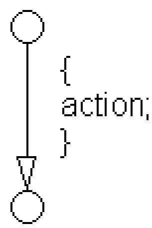
8.1.5. db_0132: Transitions in Flowcharts

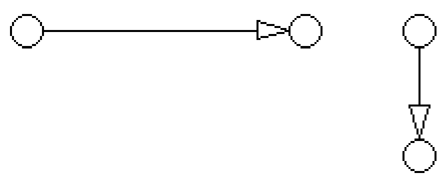
ID: Title	db_0132: Transitions in Flowcharts
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	

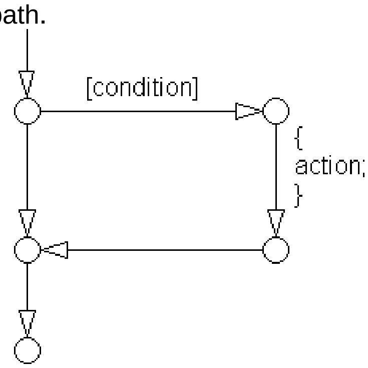
<table>
<tr>
<td>Description</td>
<td>
The following rules apply to transitions in Flowcharts:

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.

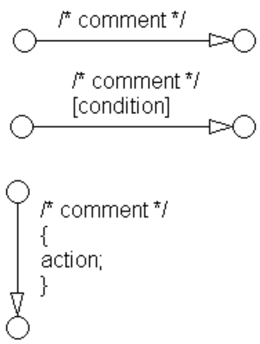
Transition with condition:


Transition with condition action:


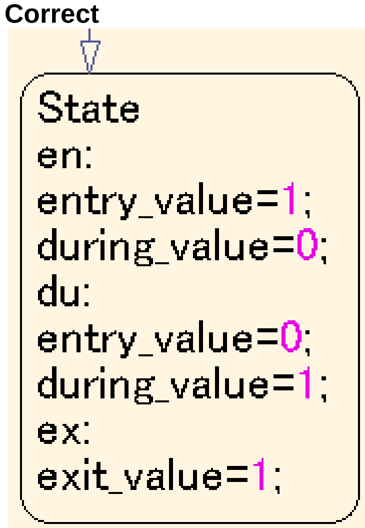
Empty transition:

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:

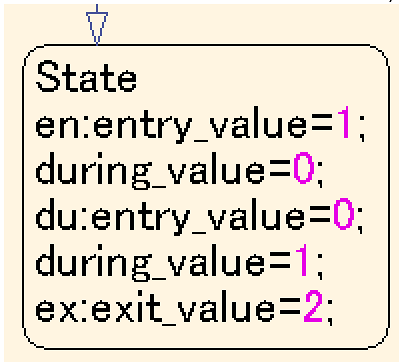
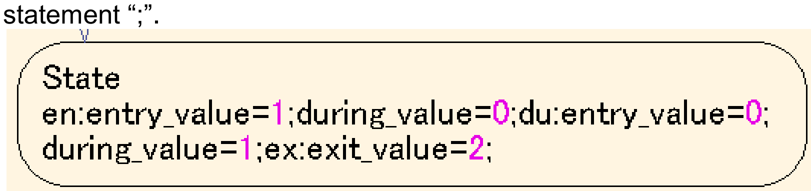

At every junction, except for the last junction of a flow diagram, exactly one unconditional transition begins. Every decision point (junction) must have a default path.


A transition may have a comment:
</td>
</tr>
</table>

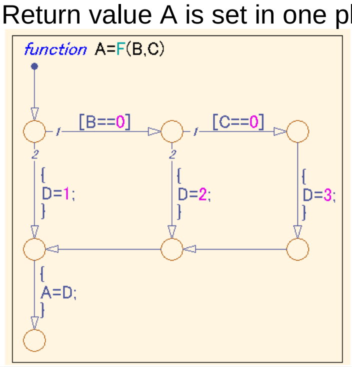
	
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.00

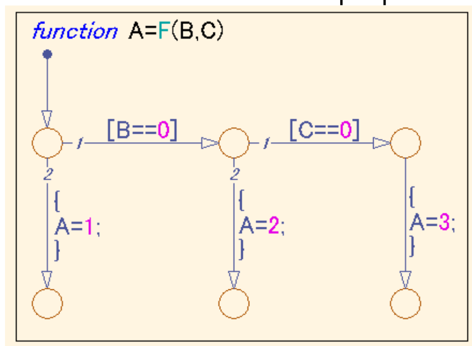
8.1.6. jc_0501: Format of entries in a State block

ID: Title	jc_0501: Format of entries in a State block
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	A new line should: • Start after the entry (en) during (du), and exit (ex) statements. • Start after the completion of an assignment statement ";". **Correct** 

	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 ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

8.1.7. jc_0511: Setting the return value from a graphical function

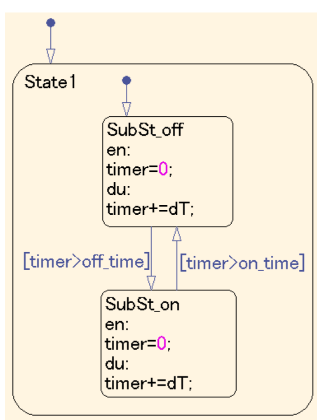
ID: Title	**jc_0511: Setting the return value from a graphical function**
Priority	mandatory
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	The return value from a graphical function must be set in only one place. **Correct** Return value A is set in one place 

<table>
<tr><td></td><td><table><tr><td>Incorrect
Return value A is set in multiple places.

</td></tr></table></td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

8.1.8. jc_0531: Placement of the default transition

<table>
<tr><td>ID: Title</td><td>jc_0531: Placement of the default transition</td></tr>
<tr><td>Priority</td><td>recommended</td></tr>
<tr><td>Scope</td><td>J-MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>

- 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.

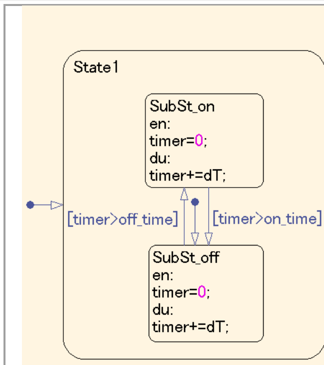
<table>
<tr><td>Correct

</td><td>

- 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.

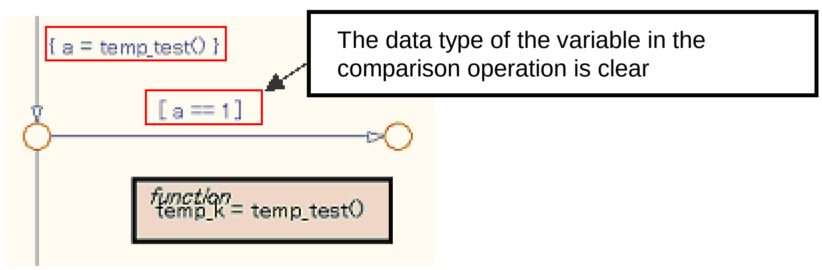
</td></tr>
<tr><td>Incorrect</td><td>

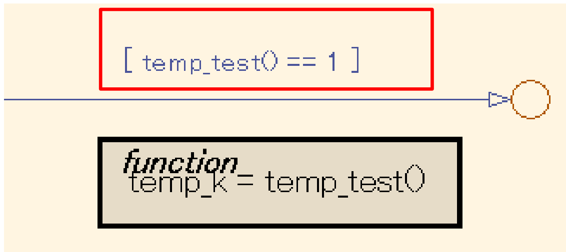
- Default transition is connected at the side of the state (State 1).
- The destination state of

</td></tr>
</table>
</td></tr>
</table>

	 the default transition is lower than the other states in the same hierarchy (SubSt_off).
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V2.00

8.1.9. jc_0521: Use of the return value from graphical functions

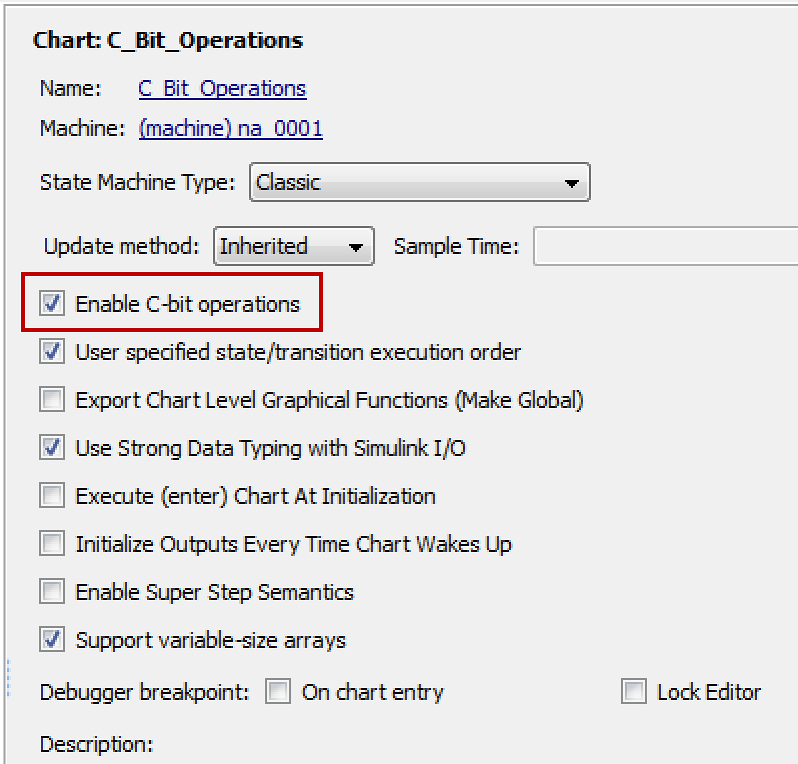
ID: Title	**jc_0521: Use of the return value from graphical functions**
Priority	recommended
Scope	J-MAAB
MATLAB Version	All
Prerequisites	
Description	The return value from a graphical function should not be used directly in a comparison operation. **Correct** An intermediate variable is used in the conditional expression after the assignment of the return value from the function "temp_test" to the intermediate variable "a".  **Incorrect** Return value of the function “temp_test” is used in the conditional expression.

	
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.00

8.2. Stateflow data and operations

8.2.1. na_0001: Bitwise Stateflow operators

ID: Title	**na_0001: Bitwise Stateflow operators**
Priority	strongly recommended
Scope	MAAB
Prerequisites	
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 “II” for Boolean operation.

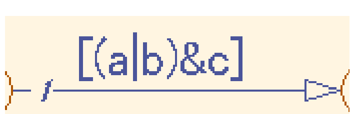
Name	Data Type
a	boolean
b	boolean
c	boolean

Use “&” and “I” for bit operation.

Name	Data Type
d	uint8
e	uint8
f	uint8

Incorrect

Use “&” and “I” for Boolean operation.



Name	Data Type
a	boolean
b	boolean
c	boolean

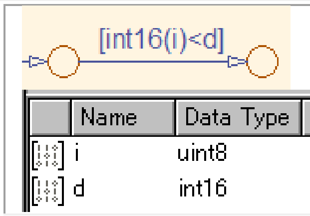
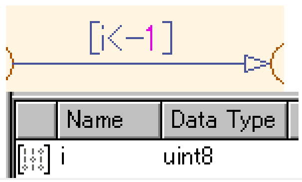
Rational	☑ Readability ☐ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

8.2.2. 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 Version	All
Prerequisites	
Description	Do not perform unary minus on unsigned integers. **Correct** `{ si16_var1=-si16_var2; }` Name: si_var2 — Data Type: int16 **Incorrect** `{ ui16_var1=-ui16_var2; }` Name: ui_var2 — Data Type: uint16
Rationale	☐ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V2.00

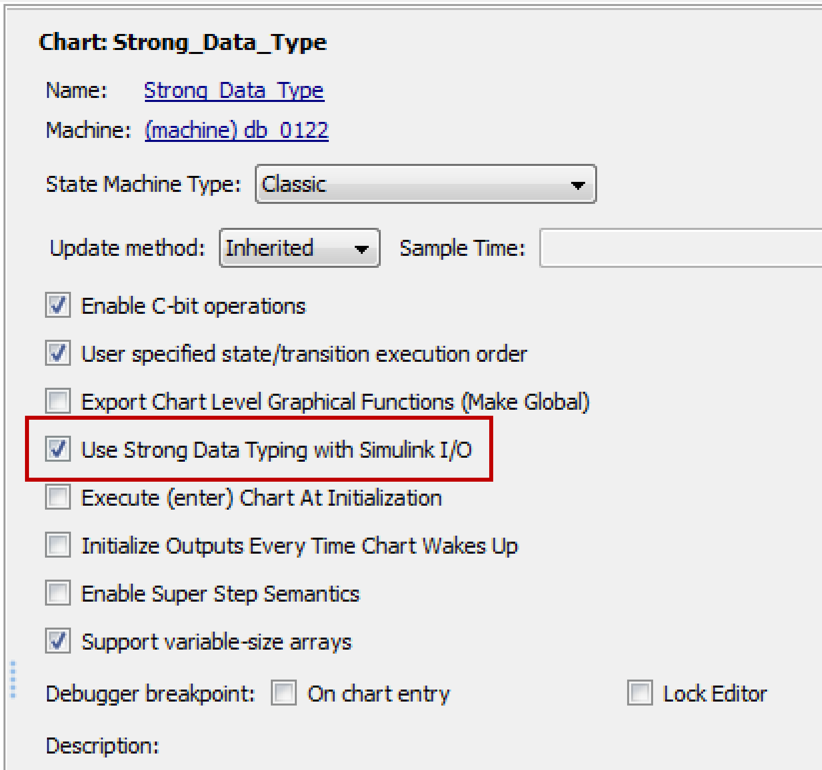
8.2.3. na_0013: Comparison operation in Stateflow

ID: Title	**na_0013: Comparison operation in Stateflow**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
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” `[i<n]` Name: i — Data Type: uint8 Name: n — Data Type: uint8 **Incorrect** Different data type in “i” and “d” `[i<d]` Name: i — Data Type: uint8 Name: d — Data Type: int16 **Correct**

	 • Do not make comparisons between unsigned integers and negative numbers. **Incorrect** 
Rationale	☐ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☑ Simulation
Last Change	V2.10

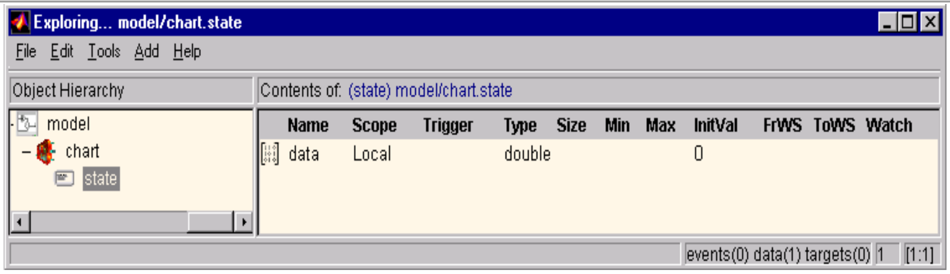
8.2.4. db_0122: Stateflow and Simulink interface signals and parameters

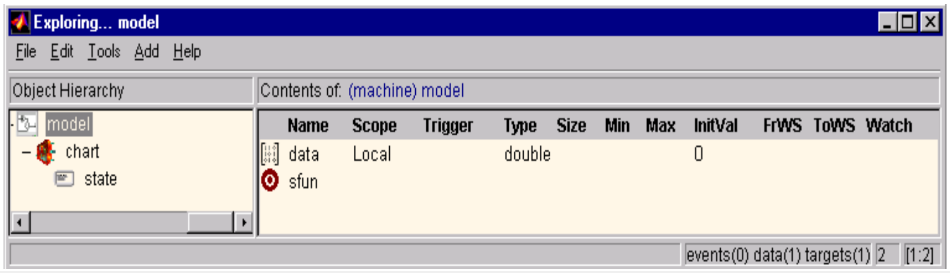
ID: Title	**db_0122: Stateflow and Simulink interface signals and parameters**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	All charts should use strong data typing with Simulink (The option "Use Strong Data Typing with Simulink I/O" must be selected).

	
Rationale	☐ Readability ☐ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.00

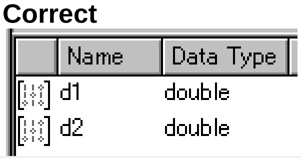
8.2.5. db_0125: Scope of internal signals and local auxiliary variables

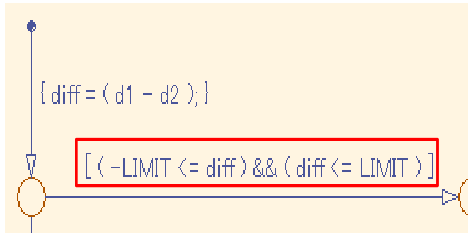
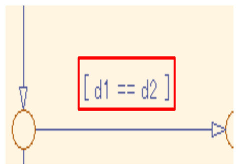
ID: Title	**db_0125: Scope of internal signals and local auxiliary variables**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Internal signals and local auxiliary variables are "Local data" in Stateflow: • All local data of a Stateflow block must be defined on the chart level or below the Object Hierarchy. • No local variables exist on the machine level (that is, there is no interaction between local data in different charts). • Parameters and constants are allowed at the machine level. **Correct**

<table>
<tr><td></td><td>

Incorrect

</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.00</td></tr>
</table>

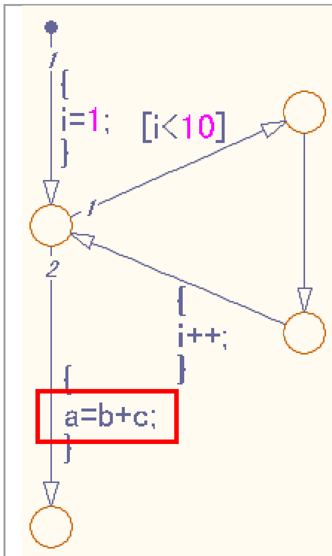
8.2.6. jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow

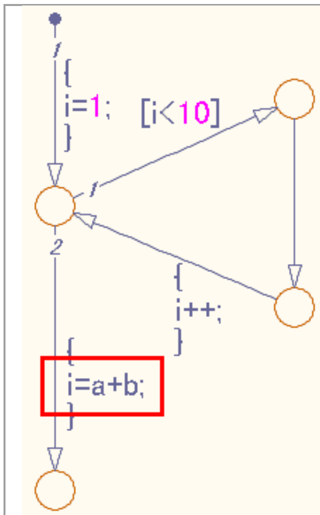
<table>
<tr><td>ID: Title</td><td>jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow</td></tr>
<tr><td>Priority</td><td>recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>• Do not use hard equality comparisons (Var1 == Var2) with two floating point numbers.
• If a hard comparison is required, a margin of error should be defined and used in the comparison (LIMIT in the example).
• Hard equality comparisons may be done between two integer data types.
Correct

</td></tr>
</table>

	 **Incorrect** 
Rationale	☐ Readability ☐ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.00

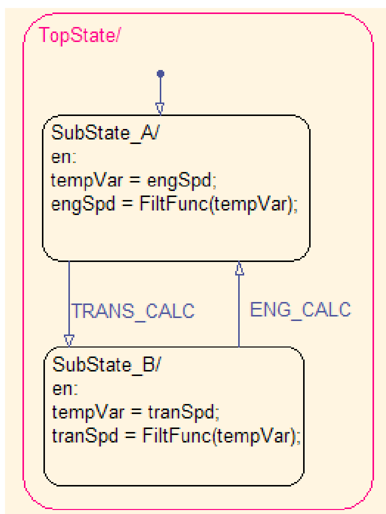
8.2.7. jc_0491: Reuse of variables within a single Stateflow scope

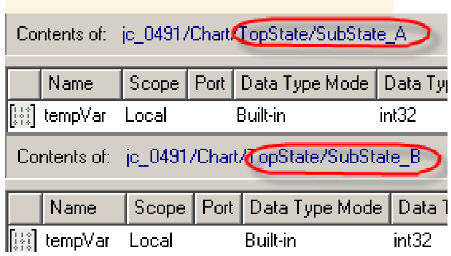
ID: Title	**jc_0491: Reuse of variables within a single Stateflow scope**
Priority	recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The same variable should not have multiple meanings (usages) within a single Stateflow state. **Correct** Variable of loop counter must not be used other than loop counter. **Incorrect** The meaning of the variable “i” changes from the index of the loop counter to the sum of a+b





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




Contents of: jc_0491/Chart/TopState/SubState_A

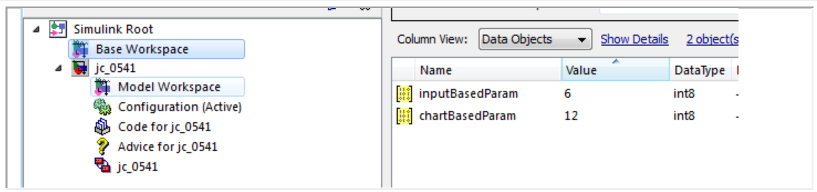
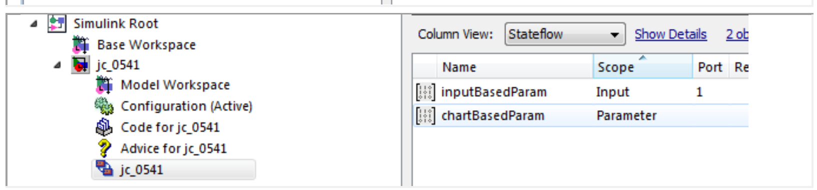
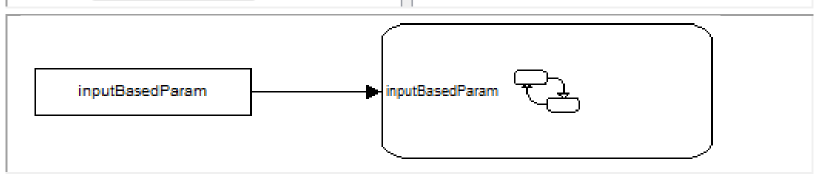
	Name	Scope	Port	Data Type Mode	Data Ty
	tempVar	Local		Built-in	int32

Contents of: jc_0491/Chart/TopState/SubState_B

	Name	Scope	Port	Data Type Mode	Data T
	tempVar	Local		Built-in	int32

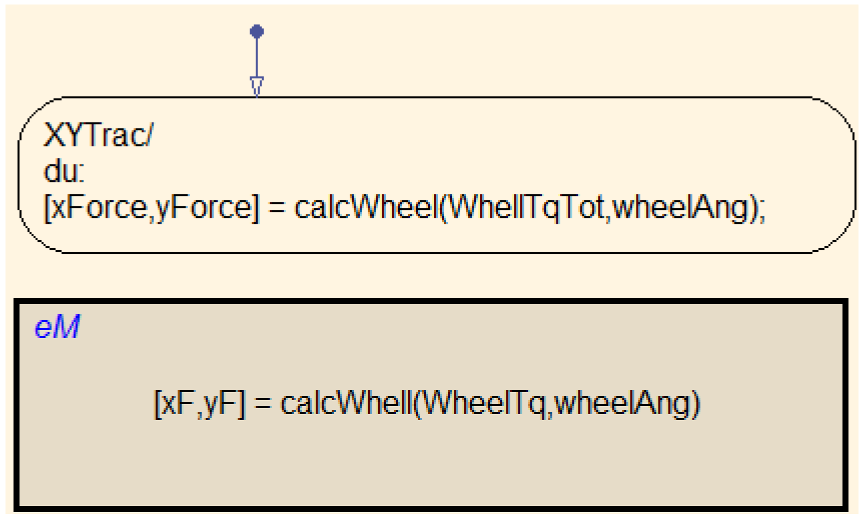
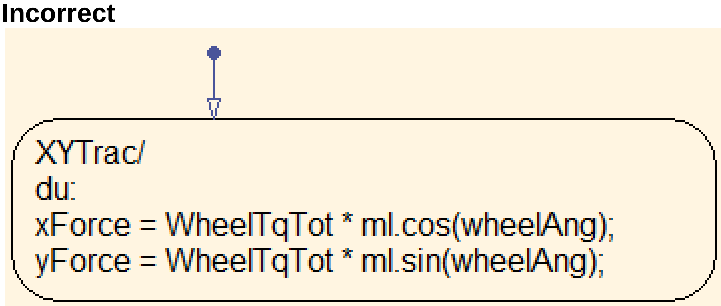
Rationale	☑ Readability ☐ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

8.2.8. jc_0541: Use of tunable parameters in Stateflow

<table>
<tr><th>ID: Title</th><th>jc_0541: Use of tunable parameters in Stateflow</th></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>Create tunable parameters in Stateflow charts in one of the following ways:
1.) Define the parameters in the Stateflow chart and corresponding parameters in the base workspace
2.) Include the tunable parameters as an input into the Stateflow chart. The parameters must be defined in the base workspace.
<table>
<tr><td>Base workspace definitions</td><td>
</td></tr>
<tr><td>Stateflow chart definitions</td><td>
</td></tr>
<tr><td>Stateflow chart</td><td>
</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☐ Readability
☐ Workflow
☐ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

8.2.9. db_0127: MATLAB commands in Stateflow

ID: Title	db_0127: MATLAB commands in Stateflow
Priority	mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	In Stateflow charts: • Do not use the .ml syntax

	Individual companies should decide on the use of MATLAB functions. If they are permitted, then MATLAB functions should only be accessed through the MATLAB function block. **Correct**  **Incorrect** 
Rationale	☐ Readability ☐ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Note	Code generation supports a limited subset of the MATLAB functions. For a complete list of the supported function, see the MathWorks documentation.
Last Change	V2.20

8.2.10. jm_0011: Pointers in Stateflow

ID: Title	**jm_0011: Pointers in Stateflow**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	In a Stateflow diagram, pointers to custom code variables are not allowed.

Rationale	☑ Readability ☐ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V1.00

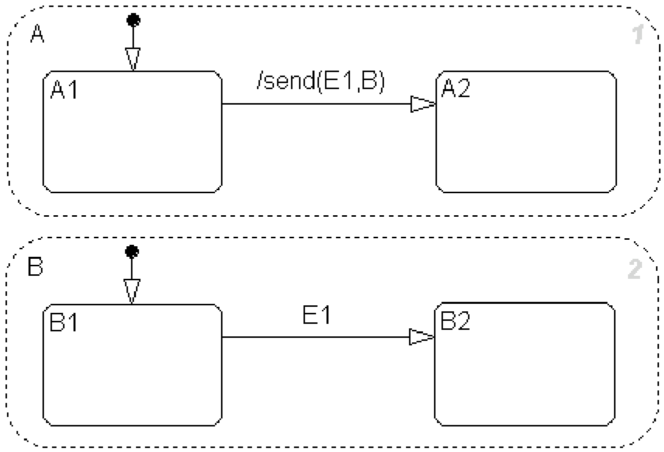
8.3. Events

8.3.1. db_0126: Scope of events

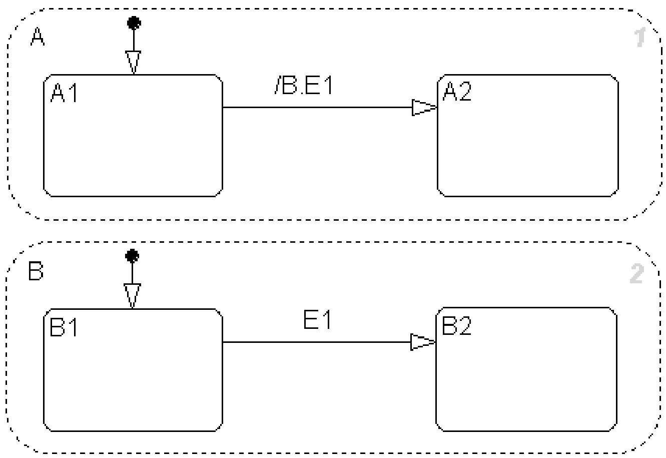
ID: Title	**db_0126: Scope of events**
Priority	Mandatory
Scope	MAAB
MATLAB Version	Pre R2009b
Prerequisites	
Description	The following rules apply to events in Stateflow: • All events of a Chart must be defined on the chart level or lower. • There is no event on the machine level (that is, there is no interaction with local events between different charts).
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.20

8.3.2. jm_0012: Event broadcasts

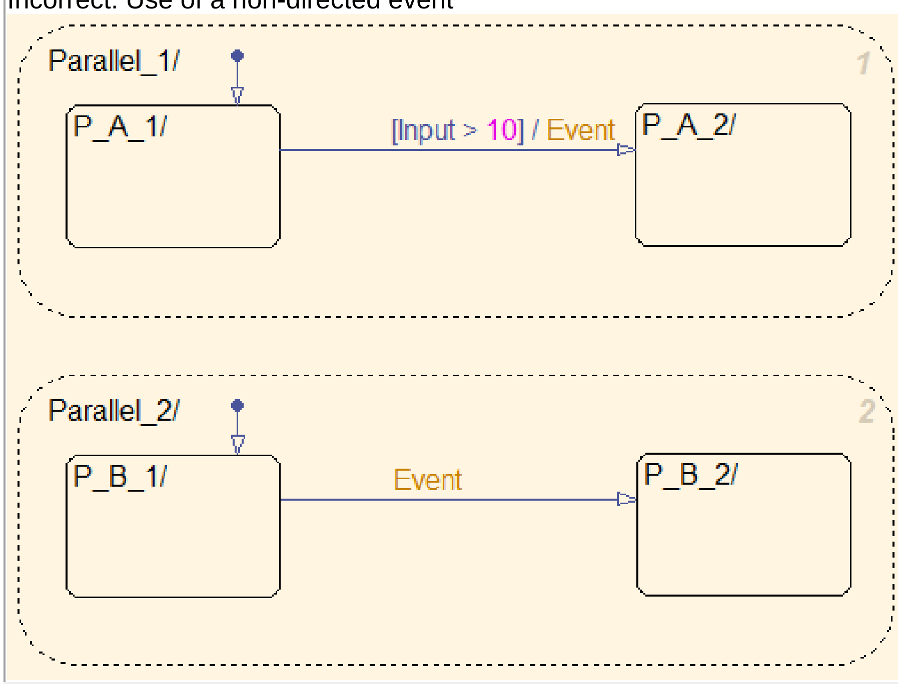
ID: Title	**jm_0012: Event broadcasts**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	db_0126: Scope of events
Description	The following rules apply to event broadcasts in Stateflow: • Directed event broadcasts are the only type of event broadcasts allowed. • The send syntax 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 the send syntax:



Correct: Example using qualified event names:



Incorrect: Use of a non-directed event

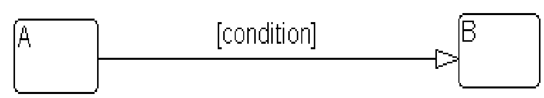


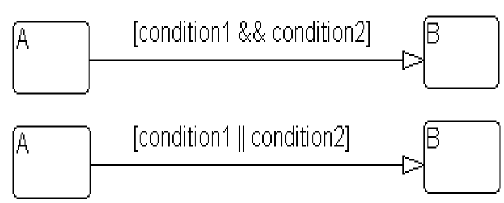
<table>
<tr><td>Rationale</td><td>☑ Readability
☑ Workflow
☑ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

8.4. Statechart Patterns

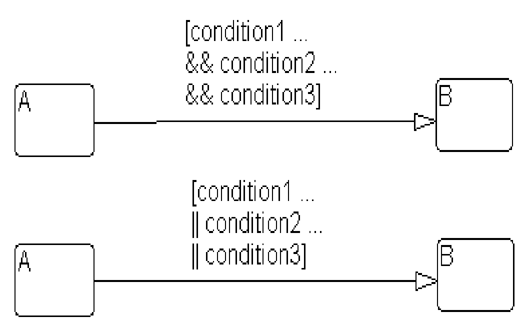
8.4.1. db_0150: State machine patterns for conditions

<table>
<tr><td>ID: Title</td><td>db_0150: State machine patterns for conditions</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for conditions within Stateflow state machines:
<table>
<tr><th>Equivalent Functionality</th><th>State Machine Pattern</th></tr>
<tr><td>ONE CONDITION:

(condition)</td><td>
</td></tr>
<tr><td>UP TO THREE CONDITIONS, SHORT FORM:
(The use of different logical operators in this form is not allowed, use sub conditions instead)

(condition1 && condition2)
(condition1 || condition2)</td><td>
</td></tr>
<tr><td>TWO OR MORE CONDITIONS, MULTILINE FORM:
A sub condition is a set of logical operations, all of the same type, enclosed in parentheses.
(The use of different operators in this form is not allowed, use sub conditions instead.)

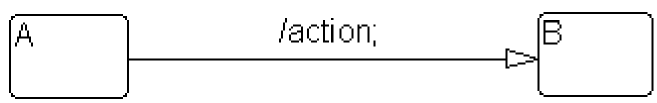
(condition1 ...
&& condition2 ...
&& condition3)

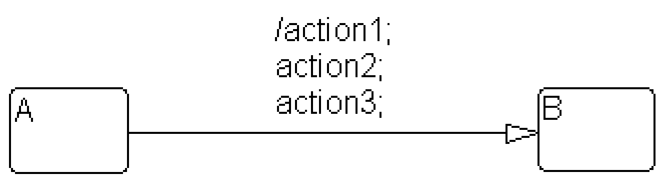
(condition1 ...
|| condition2 ...
|| condition3)</td><td>
</td></tr>
</table></td></tr>
</table>

Rationale	☑ Readability ☐ Workflow ☐ Simulation ☐ Verification and Validation ☐ Code Generation
Last Change	V2.20

8.4.2. db_0151: State machine patterns for transition actions

<table>
<tr><td>ID: Title</td><td>db_0151: State machine patterns for transition actions</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for transition actions within Stateflow state machines:
<table>
<tr><th>Equivalent Functionality</th><th>State Machine Pattern</th></tr>
<tr><td>ONE TRANSITION ACTION:

action;</td><td>
</td></tr>
<tr><td>TWO OR MORE TRANSITION ACTIONS, MULTILINE FORM: (Two or more transition actions in one line are not allowed.)

action1;
action2;
action3;</td><td>
</td></tr>
</table></td></tr>
<tr><td>Rationale</td><td>☑ Readability
☑ Workflow
☑ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

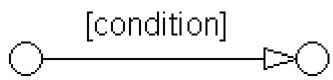
8.5. Flowchart Patterns

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

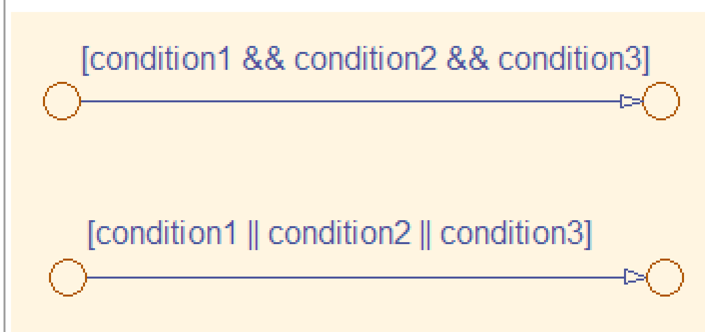
8.5.1. db_0148: Flowchart patterns for conditions

ID: Title	**db_0148: Flowchart patterns for conditions**
Priority	strongly recommended

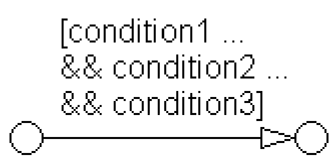
<table>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for conditions within Stateflow Flowcharts:
<table>
<tr><th>Equivalent Functionality</th><th>Flowchart Pattern</th></tr>
<tr><td>ONE CONDITION:

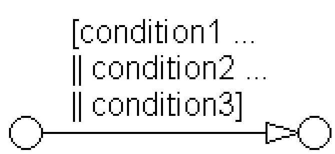
[condition]</td><td>


</td></tr>
<tr><td>UP TO THREE CONDITIONS, SHORT FORM:
(The use of different logical operators in this form is not allowed. Use sub conditions instead.)

[condition1 && condition2 && condition3]
[condition1 || condition2 || condition3]</td><td>


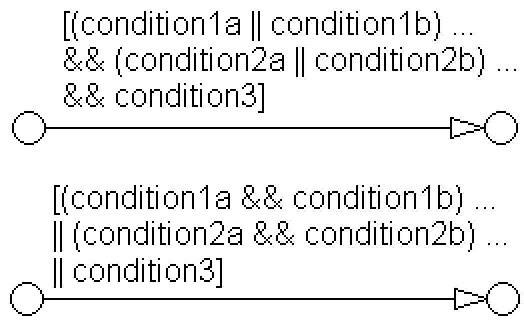
</td></tr>
<tr><td>TWO OR MORE CONDITIONS, MULTILINE FORM:
(The use of different logical operators in this form is not allowed. Use sub conditions instead.)

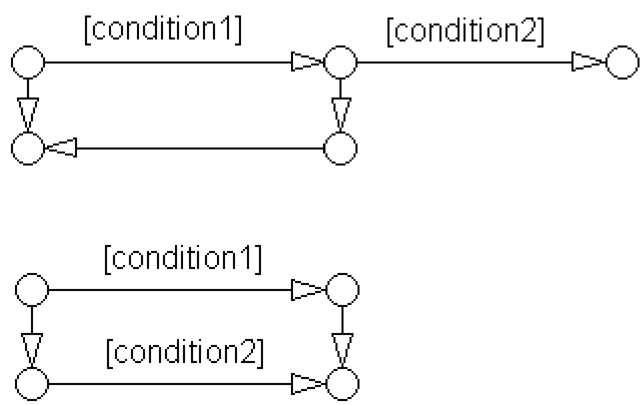
[condition1 ...
&& condition2 ...
&& condition3]
[condition1 ...
|| condition2 ...
|| condition3]</td><td>




</td></tr>
</table>
</td></tr>
</table>

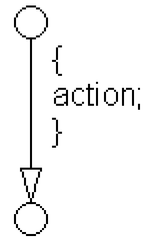
<table>
<tr><td></td><td>
<table>
<tr><td>CONDITIONS WITH SUBCONDITIONS:
(The use of different logical operators to connect sub conditions is not allowed. The use of brackets is mandatory.)

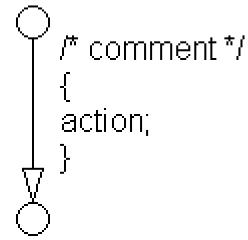
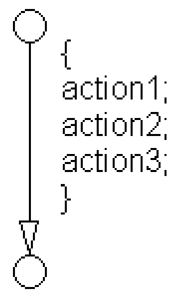
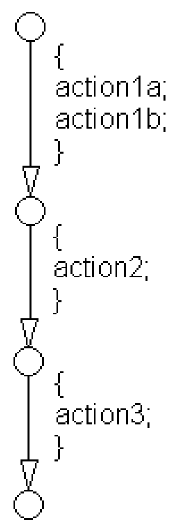
[(condition1a || condition1b) ...
&& (condition2a || condition2b) ...
&& (condition3)]
[(condition1a && condition1b) ...
|| (condition2a && condition2b) ...
|| (condition3)]</td><td>
</td></tr>
<tr><td>CONDITIONS THAT ARE VISUALLY SEPARATED:
(This form can be combined with the preceding patterns.)

[condition1 && condition2]
[condition1 || condition2]</td><td>
</td></tr>
</table>
</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

8.5.2. db_0149: Flowchart patterns for condition actions

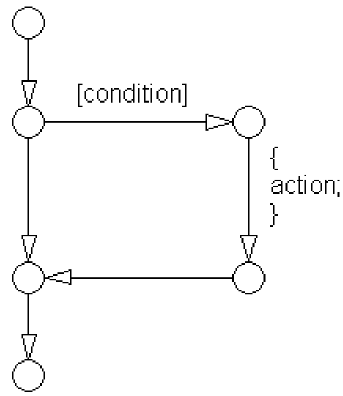
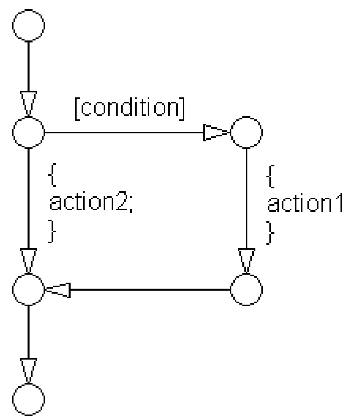
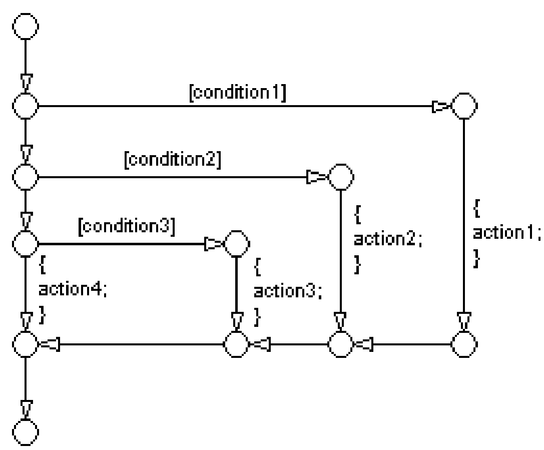
<table>
<tr><td>ID: Title</td><td>db_0149: Flowchart patterns for condition actions</td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for condition actions within Stateflow Flowcharts:
<table><tr><th>Equivalent Functionality</th><th>Flowchart Pattern</th></tr></table></td></tr>
</table>

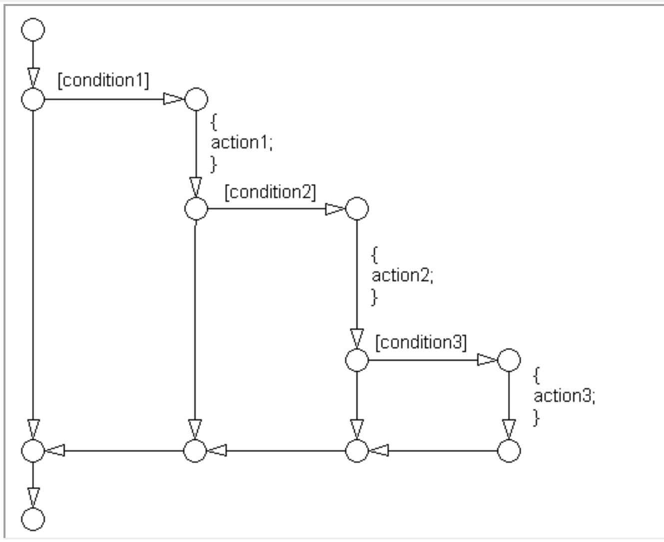
<table>
<tr><td></td><td>
<table>
<tr><td>ONE CONDITION ACTION:
action;</td><td>


</td></tr>
<tr><td>TWO OR MORE CONDITION ACTIONS, MULTILINE FORM:
(Two or more condition actions in one line are not allowed.)
action1; ...
action2; ...
action3; ...</td><td>
</td></tr>
<tr><td>CONDITION ACTIONS, WHICH ARE VISUALLY SEPARATED:
(This form can be combined with the preceding patterns.)
action1a;
action1b;
action2;
action3;</td><td>
</td></tr>
</table>
</td></tr>
<tr><td>Rationale</td><td>☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td>V2.20</td></tr>
</table>

8.5.3. db_0134: Flowchart patterns for If constructs

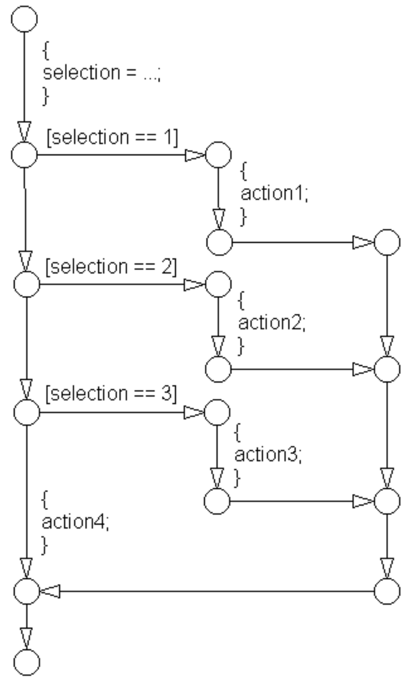
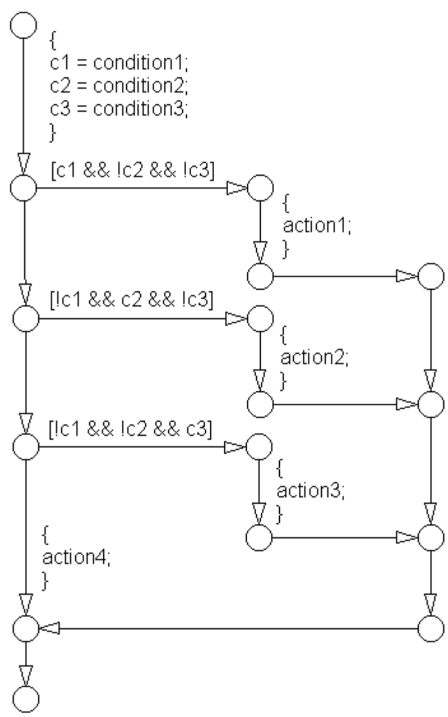
ID: Title	**db_0134: Flowchart patterns for If constructs**
Priority	strongly recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	db_0148: Flowchart patterns for conditions

<table>
<tr><td></td><td>db_0149: Flowchart patterns for condition actions</td></tr>
<tr><td>Description</td><td>The following patterns are used for If constructs within Stateflow Flowcharts:
<table>
<tr><th>Equivalent Functionality</th><th>Flowchart Pattern</th></tr>
<tr><td>IF THEN
if (condition){
    action;
}</td><td>
</td></tr>
<tr><td>IF THEN ELSE
if (condition) {
    action1;
}
else {
    action2;
}</td><td>
</td></tr>
<tr><td>IF THEN ELSE IF
if (condition1) {
    action1;
}
else if (condition2) {
    action2;
}
else if (condition3) {
    action3;
}
else {
    action4;
}</td><td>
</td></tr>
</table></td></tr>
</table>

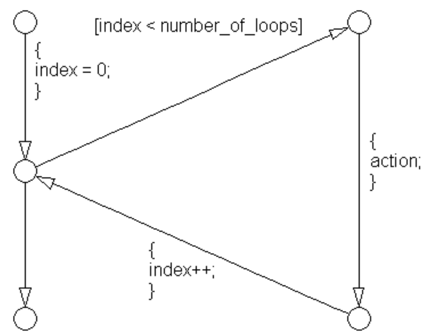
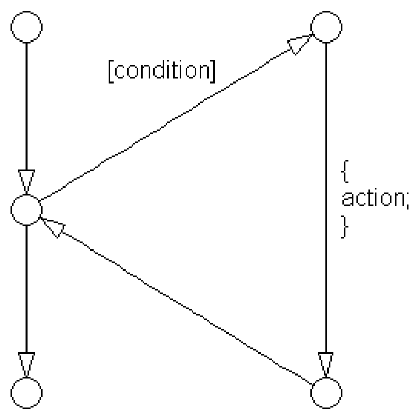
<table>
<tr><td></td><td><table><tr><td>Cascade of IF THEN
if (condition1) {
  action1;
  if (condition2) {
    action2;
    if (condition3) {
      action3;
    }
  }
}</td><td>
</td></tr></table></td></tr>
<tr><td>Rationale</td><td>☑ Readability
☑ Workflow
☑ Simulation
☑ Verification and Validation
☑ Code Generation</td></tr>
<tr><td>Last Change</td><td>V1.00</td></tr>
</table>

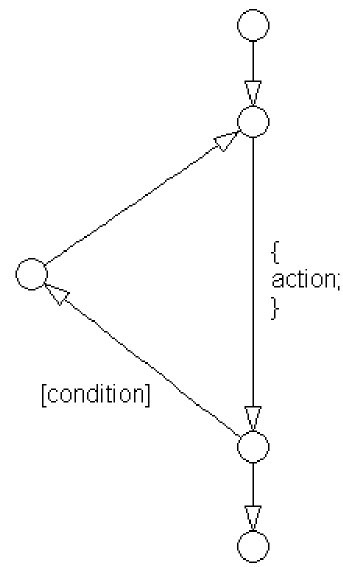
8.5.4. db_0159: Flowchart patterns for case constructs

<table>
<tr><th>ID: Title</th><th>db_0159: Flowchart patterns for case constructs</th></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>db_0148: Flowchart patterns for conditions
db_0149: Flowchart patterns for condition actions</td></tr>
<tr><td>Description</td><td>The following patterns must be used for case constructs within Stateflow Flowcharts:<table><tr><th>Equivalent Functionality</th><th>Flowchart Pattern</th></tr></table></td></tr>
</table>

<table>
<tr><td></td><td>CASE with exclusive selection
selection = ...;
switch (selection) {
    case 1:
        action1;
    break;
    case 2:
        action2;
    break;
    case 3:
        action3;
    break;
    default:
        action4;
}</td><td>
</td></tr>
<tr><td></td><td>CASE with exclusive conditions
c1 = condition1;
c2 = condition2;
c3 = condition3;
if (c1 && !c2 && !c3) {
    action1;
}
else if (!c1 && c2 && !c3) {
    action2;
}
else if (!c1 && !c2 && c3) {
    action3;
}
else {
    action4;
}</td><td>
</td></tr>
<tr><td>Rationale</td><td colspan="2">☑ Readability
☐ Workflow
☐ Simulation
☐ Verification and Validation
☐ Code Generation</td></tr>
<tr><td>Last Change</td><td colspan="2">V1.00</td></tr>
</table>

8.5.5. db_0135: Flowchart patterns for loop constructs

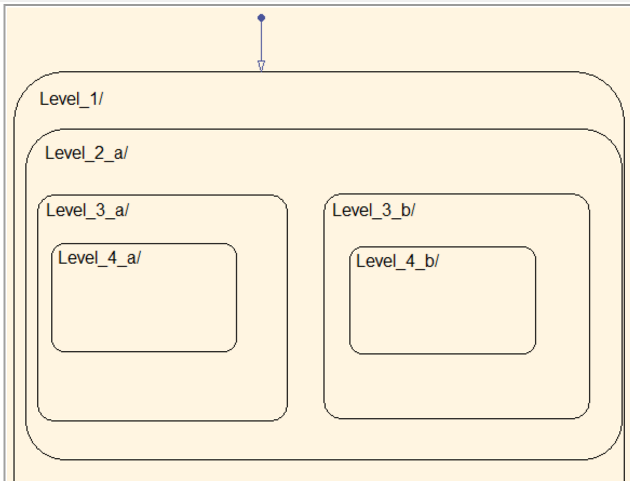
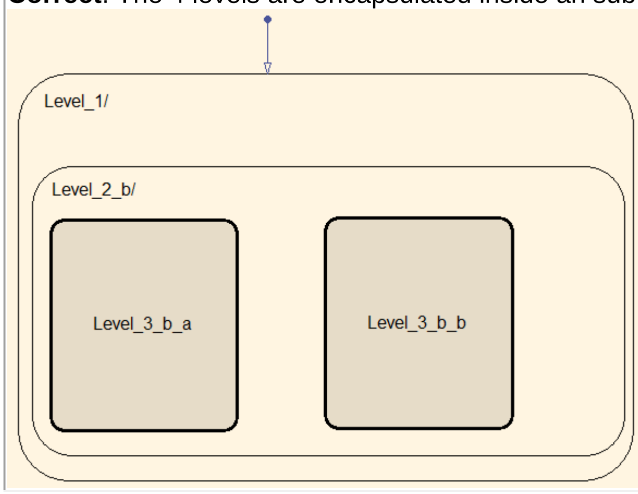
<table>
<tr><td>ID: Title</td><td>db_0135: Flowchart patterns for loop constructs</td></tr>
<tr><td>Priority</td><td>recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>db_0148: Flowchart patterns for conditions
db_0149: Flowchart patterns for condition actions</td></tr>
<tr><td>Description</td><td>The following patterns must be used to create Loops within Stateflow Flowcharts:
<table>
<tr><th>Equivalent Functionality</th><th>Flowchart Pattern</th></tr>
<tr><td>FOR LOOP
for
(index=0;index<number_of_loops;index++)
{
action;
}</td><td>
</td></tr>
<tr><td>WHILE LOOP
while (condition) {
action;
}</td><td>
</td></tr>
</table></td></tr>
</table>

	DO WHILE LOOP do { action; } while (condition);	
Rationale	☑ Readability ☐ Workflow ☐ Simulation	☐ Verification and Validation ☐ Code Generation
Last Change	V1.00	

8.6. State chart architecture

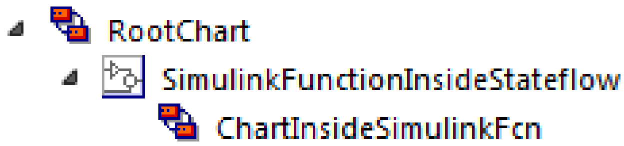
8.6.1. na_0038: Levels in Stateflow charts

ID: Title	**na_0038: Levels in Stateflow charts**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisite	
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 then 3 deep.

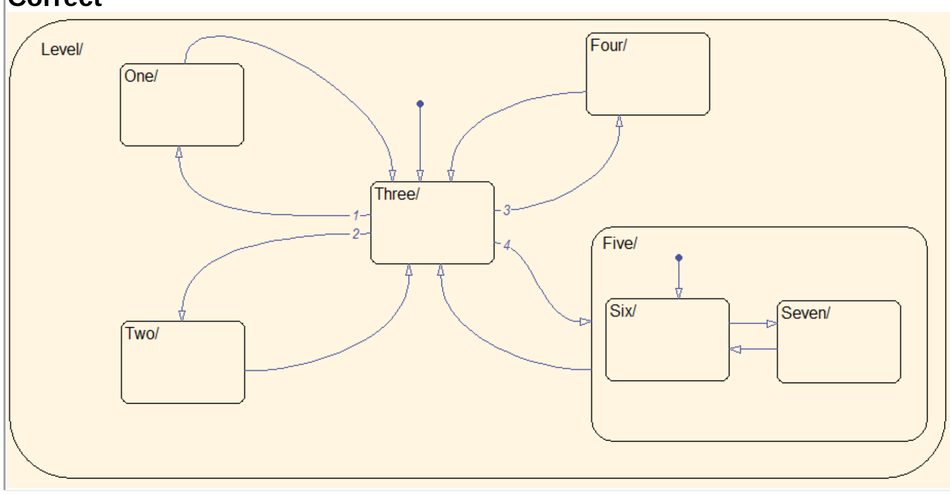
	 **Correct**: The 4 levels are encapsulated inside an sub chart 
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V3.00

8.6.2. na_0039: Use of Simulink in Stateflow charts

ID: Title	**na_0039: Use of Simulink in Stateflow charts**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	2010B and Later
Prerequisite	
Description	Do not nest Stateflow charts inside Simulink functions included in Stateflow charts. **Incorrect**

	
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

8.6.3. na_0040: Number of states per container

ID: Title	**na_0040: Number of states per container**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisite	
Description	The number of viewable States per container should be limited, typically to 6 to 10 states per container. The number is based on the visible states in the diagram. **Correct** 
Note	A container is either a State, Box or root level chart.
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

8.6.4. na_0041: Selection of function type

ID: Title	**na_0041: Selection of function type**

Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisite	
Description	Stateflow supports three types of functions: Graphical, MATLAB and Simulink. The appropriate function depends on the type of operations required: • Simulink • Transfer functions • Integrators • Table look-ups • MATLAB • Complex equations • If / then /else logic • Graphical functions • If / then / else logic
Rationale	☐ Readability ☐ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

8.6.5. na_0042: Location of Simulink functions

ID: Title	**na_0042: Location of Simulink functions**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisite	na_0039: Use of Simulink in Stateflow charts
Description	When deciding whether to embed Simulink functions inside a Stateflow chart, the following conditions make embedding the preferred option. If the Simulink functions: • Use only local Chart data or • Use a mixture of local Chart data and inputs from Simulink or • Are called from multiple locations within the chart or • Are not called every time step
Rationale	☑ Readability ☐ Verification and Validation ☑ Workflow

	☐ Simulation ☐ Code Generation
Last Change	V3.00

9.Enumerated Data

9.1.1. na_0033: Enumerated Types Usage

ID: Title	**na_0033: Enumerated Types Usage**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	R2010b and later
Prerequisites	na_0002: Appropriate implementation of fundamental logical and numerical operations
Description	An enumerated data type should be used when a signal or parameter can take on a finite set of integer values, and those values are associated with a set of named items. The names, called *literals*, have meaning in the context of the algorithm or the domain in which it operates. Typically, these literals represent an operating mode, signal status, build variation, or some other discrete property that the quantity represented by the variable can take on. A typical automotive example of this is the modes of a transmission: Park, Reverse Neutral, Drive, Low Within a project, there must be provisions in the code build process to ensure that the same literal is not defined by multiple enumerated data types.
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☑ Simulation
See also	dm_0002: Enumerated type usage
Last Change	V3.00

9.1.2. na_0031: Definition of default enumerated value

ID: Title	**na_0031: Definition of default enumerated value**
Priority	Recommended
Scope	NA-MAAB
MATLAB Version	R2010b and later
Prerequisites	
Description	The default value of the enumeration should always be explicitly defined for the enumerated type.
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

10.MATLAB Functions

10.1. MATLAB Function Appearance

10.1.1. 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 Version	All
Prerequisites	
Description	The number of levels of nested if /else and case statements should be limited, typically to 3 levels.
See also	jr_0002: Number of nested if/else and case statement blocks
Rationale	☑ Readability ☐ Verification and Validation ☐ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

10.1.2. na_0019: Restricted Variable Names

ID: Title	**na_0019: Restricted Variable Names**
Priority	Mandatory
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	To improve the readability of the MATLAB code, avoid using reserved C variable names. For example, avoid using const, TRUE, FALSE, infinity, nil, double, single, or enum in MATLAB Function code. These names may conflict with the compiler after C code is generated from the MATLAB code. Avoid using variable names that conflict with MATLAB Functions, for example "conv".
Note	Reserved key words are defined in Simulink Coder > User's Guide > Code Generation> Configuration > Code Appearance.
See also	Derived from jh_0021: Restricted Variable Names
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V3.00

10.1.3. na_0025: MATLAB Function Header

ID: Title	**na_0025: MATLAB Function Header**
Priority	Strongly Recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	MATLAB Functions must have a descriptive header. Header content may include, but is not limited to, the following types of information: • Function name • Description of function • Assumptions and Limitations • Description of changes from previous versions • Lists of inputs and outputs **Example:** `%% Function Name: NA_0025_Example_Header` `%` `% Description: An example of a header file` `%` `% Assumptions: None` `%` `% Inputs:` `%   List of input arguments` `%` `% Outputs:` `%   List of output arguments` `%` `% $Revision: 3.0$` `% $Author: MAAB$` `% $Date: July 24,2012$` `%` `%------------------------------------------------`
See also	jh_0073: eML Header version
Rationale	☑ Readability ☑ Workflow ☐ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V3.00

10.2. MATLAB Function Data and Operations

10.2.1. 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 Version	All
Prerequisites	
Description	All inputs and outputs to MATLAB Function blocks should have the data type explicitly defined, either in the Model Explorer or at the start of the function. This provides a more rigorous data type check for MATLAB Function blocks and prevents the need for using assert statements.
See also	jh_0063: eML block input / output settings
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

10.2.2. na_0024: Global Variables

ID: Title	**na_0024: Global Variables**
Priority	Strongly recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	The preferred method for accessing common data is with 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`

	<pre>function WheelFaultEvaluation(WheelData) %#codegen global ErrorFlag_DataStore if (WheelData.SlipHigh) ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,WHEELSLIP); end if (WheelData.SlipHigh) ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,LOWRPMFAULT); end end</pre>
See also	ek_0003: Global Variables
Rationale	☑ Readability ☐ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V3.00

10.3. MATLAB Function Patterns

10.3.1. na_0022: Recommended patterns for Switch / Case statements

ID: Title	**na_0022: Recommended patterns for Switch / Case statements**
Priority	Mandatory
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	Switch / Case statements must use constant values for the “Case” arguments. Input variables cannot be used in the “Case” arguments **Correct** <pre>function outVar = NA_0022_Pass(SwitchVar) %#codegen switch SwitchVar case Case_1_Parameter % Parameter outVar = 0; case NA_0022.Case_2 % Enumerated Data type outVar = 1; case 3 % Hard Code Value outVar = 2; otherwise outVar = 10; end end</pre> **Incorrect**

	`` function outVar = NA_0022_Fail(Case_1,Case_2,Case_3,SwitchVar) %#codegen switch SwitchVar case Case_1 outVar = 1; case Case_2 outVar = 2; case Case_3 outVar = 3; otherwise outVar = 10; end end ```
See also	jh_0026: Switch / Case statement
Rationale	☐ Readability ☑ Verification and Validation ☐ Workflow ☑ Code Generation ☑ Simulation
Last Change	V3.00

10.4. MATLAB Function Usage

10.4.1. na_0016: Source lines of MATLAB Functions

ID: Title	na_0016: Source lines of MATLAB Functions
Priority	Mandatory
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	The length of MATLAB functions should be limited, with a recommended limit of 60 lines of code. This restriction applies to MATLAB Functions that reside in the Simulink block diagram and external MATLAB files with a .m extension. If sub-functions are used, they may use additional lines of code. Also limit the length of sub-functions to 60 lines of code.
See also	IM_0008: Source lines of eML
Rationale	☑ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

10.4.2. na_0017: Number of called function levels

ID: Title	na_0017: Number of called function levels

Priority	Mandatory
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	The number of levels of sub-functions should be limited, typically to 3 levels. MATLAB function blocks that resides at the Simulink block diagram level counts as the first level, unless it is simply a wrapper for an external MATLAB file with a .m extension. This includes functions that are defined within the MATLAB block and those in separate .m files.
Note	Standard utility functions, such as built in functions like sqrt or log, are not included in the number of levels. Likewise, commonly used custom utility functions can be excluded from the number of levels.
See also	im_0009: Number of called function levels
Rationale	☑ Readability ☑ Verification and Validation ☐ Workflow ☐ Code Generation ☐ Simulation
Last Change	V3.00

10.4.3. na_0021: Strings

ID: Title	**na_0021: Strings**
Priority	Strongly recommended
Scope	NA-MAAB
MATLAB Version	All
Prerequisites	
Description	The use of strings is not recommended. MATLAB Functions store strings as character arrays. The arrays cannot be resized to accommodate a string value of different length, due to lack of dynamic memory allocation. Stings are not a supported data type in Simulink, so MATLAB Function blocks cannot pass the string data outside the block. For example, the following code will produce an error: *name = 'rate_error'; %this creates a 1 x 10 character array* *name = 'x_rate_error'; %this causes an error because the array size is now 1 x 12, not 1 x 10*
Note	If the string is being used for switch / case behavior, consider using enumerated data types.
See also	jh_0024: Strings
Rationale	☐ Readability ☑ Verification and Validation ☑ Workflow ☑ Code Generation ☐ Simulation
Last Change	V3.00

11.Appendix A: Recommendations for Automation Tools

These recommendations are for companies 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. In order to provide the maximum information to potential users of the tools, the MAAB strongly recommends that tool vendors provide a compliance matrix that is easily accessible when 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.
    - The MAAB Style Guidelines Title and Version document number will be included
- ☐ 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 semi-automatically corrects the non-compliance.
- ☐ Check – The tool checks and flags non-compliances. 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 guideline is not checked by the tool. It is highly recommended that the vendor provide a recommendation of how to manually check any guideline not checked by the tool.

12.Appendix B: Guideline Writing

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

- ☐ Understandable and unambiguous
- ☐ Easy to find
- ☐ Minimal

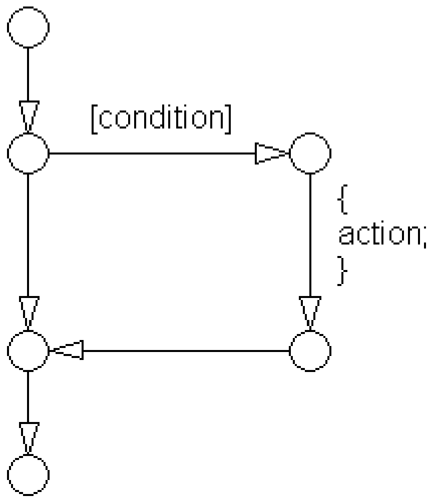
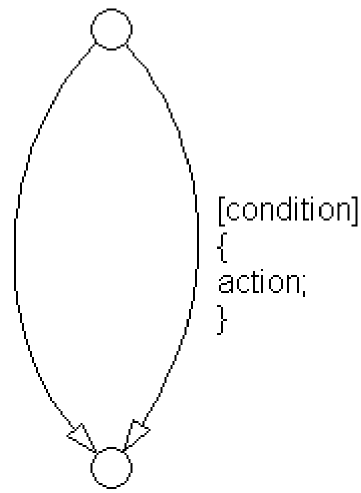
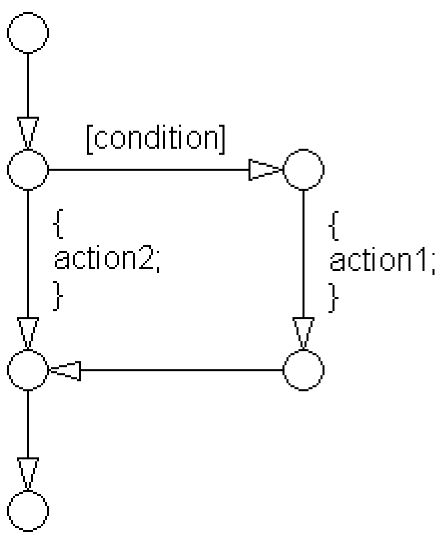
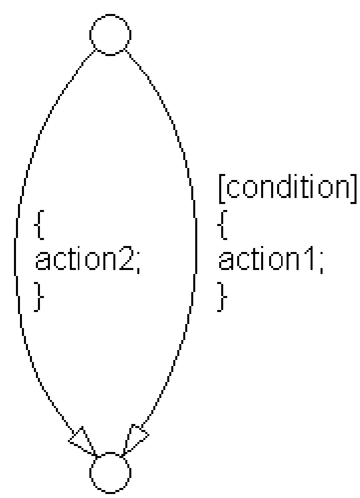
Guidelines with these characteristics are easier to understand and use.

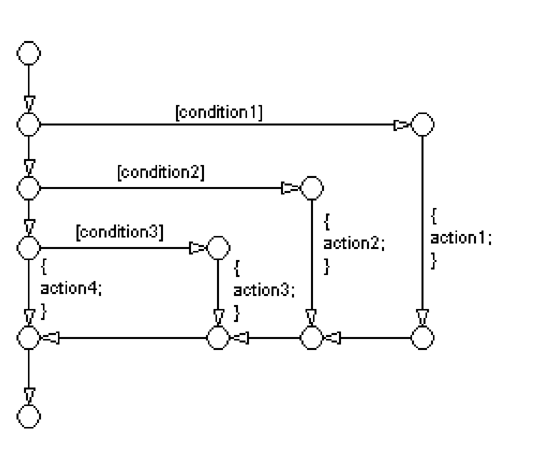
"**Understandable and unambiguous** Guideline description should be precise, clearly worded, concise and should define property characteristic of a model (or part of a model). Use the words "must," "shall," "should," and "may" carefully; they have distinct meanings that are important for model developers and model checkers (human and automated). It is helpful to the reader if the guideline author describes how the conformant state can be reached (e.g. by selecting particular options or clicking a certain button). Examples, counterexamples, pictures, diagrams, and screenshots are also helpful and therefore encouraged.
Minimize the allowable exceptions to a guideline; they blur the 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" below for some solutions.

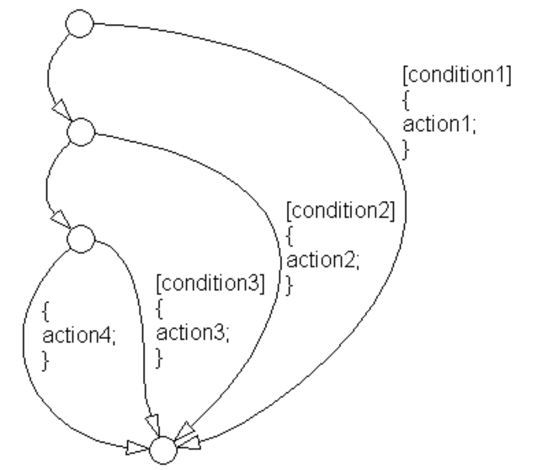
By "**Easy to find** Guideline should have a clear, stable title and be properly located among all the other guidelines. A guideline's title should describe the topic covered but not the specific evaluation criteria. This makes the title less likely to change over time and therefore easier to find. Specific evaluation criteria should be included in the guideline's description. For example, if a guideline addresses the characters allowed in names, the guideline's title should be something like "Allowed characters in names," and the guideline's description should indicate specifically what characters are or are not to be used. If a guideline has prerequisites, they should appear above or before the dependent guideline. (This may not always be possible if the prerequisite is in a different section.)

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

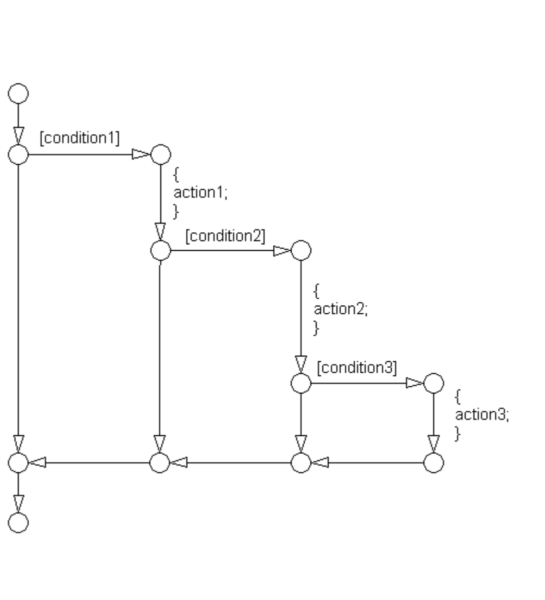
13.Appendix C: Flowchart Reference

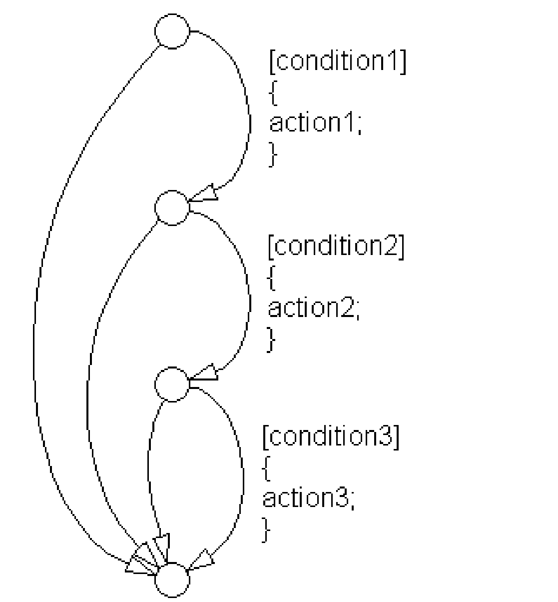
<table>
<tr><td colspan="2">The following patterns are used for If-then-else-if constructs within Stateflow Flowcharts:</td></tr>
<tr><th>Straight Line Flow Chart Pattern</th><th>Curved Line Flow Chart Pattern</th></tr>
<tr><td colspan="2">IF THEN</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">IF THEN ELSE</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">IF THEN ELSE IF</td></tr>
</table>





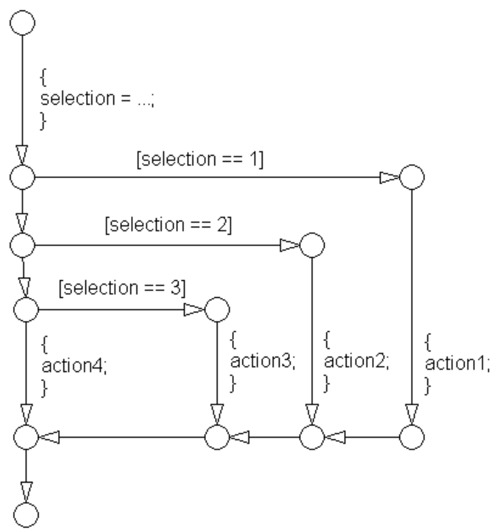
Cascade of IF THEN

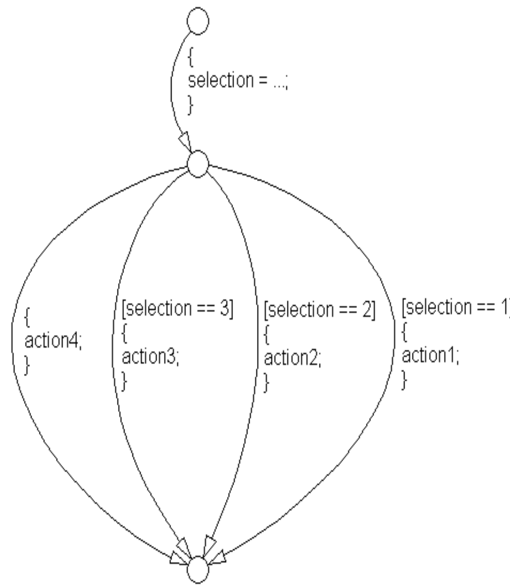




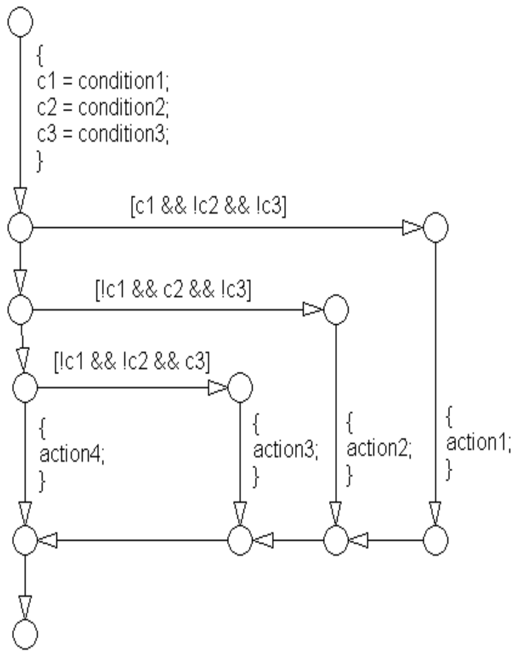
The following patterns are used the following patterns for case constructs within Stateflow Flowcharts:

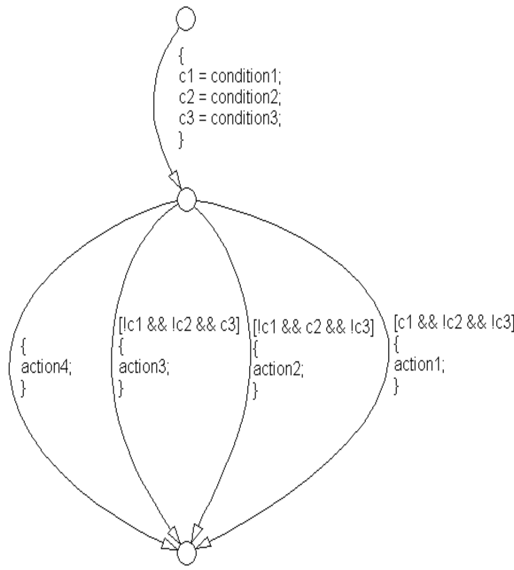
Straight Line Flow Chart Pattern	**Curved Line Flow Chart Pattern**
CASE with exclusive selection	





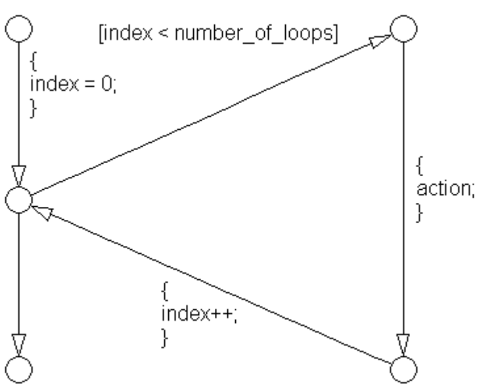
CASE with exclusive conditions

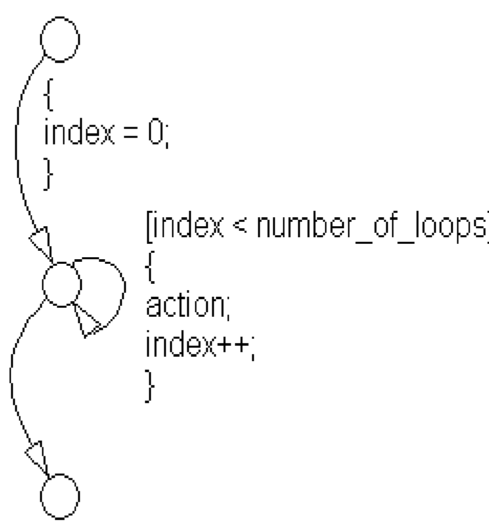




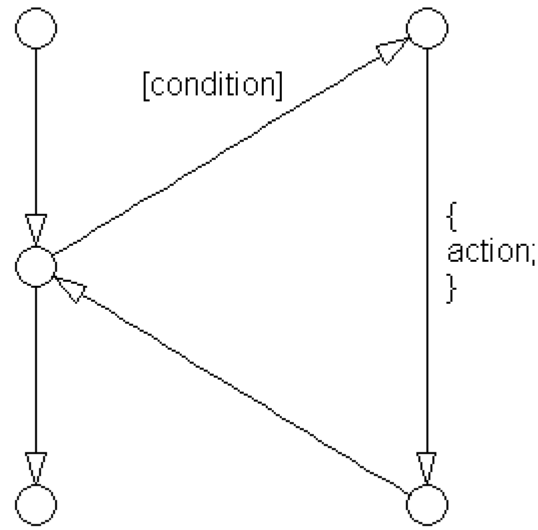
Use the following patterns for For Loops within Stateflow Flowcharts:

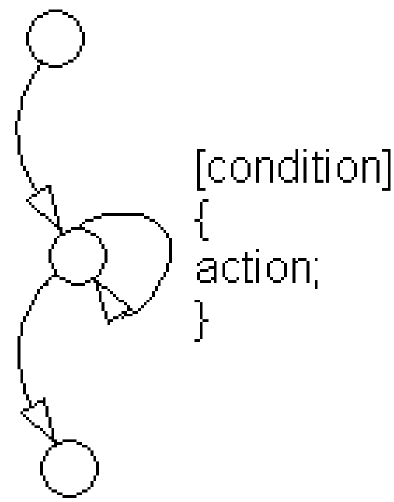
Straight Line Flow Chart Pattern	**Curved Line Flow Chart Pattern**
FOR LOOP	



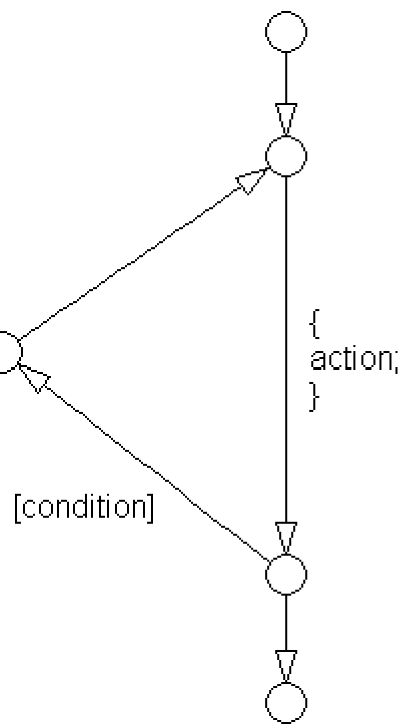


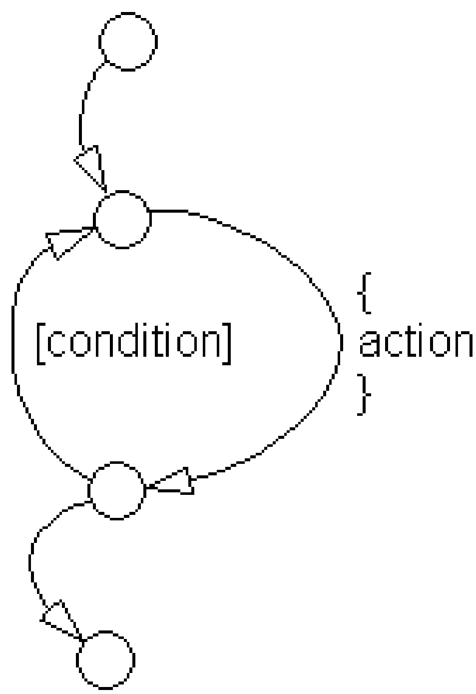
WHILE LOOP





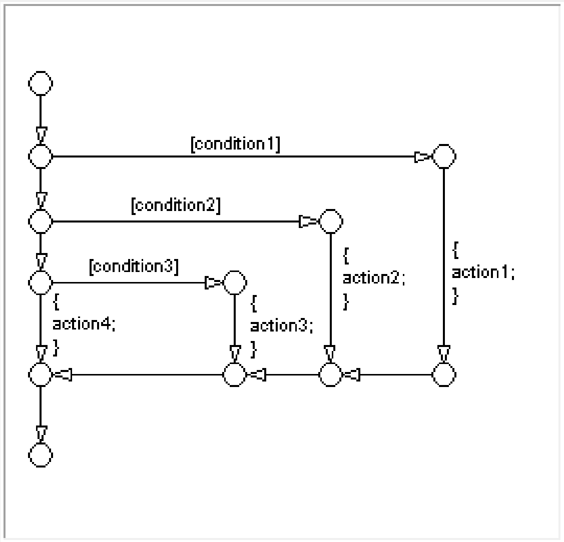
DO WHILE LOOP

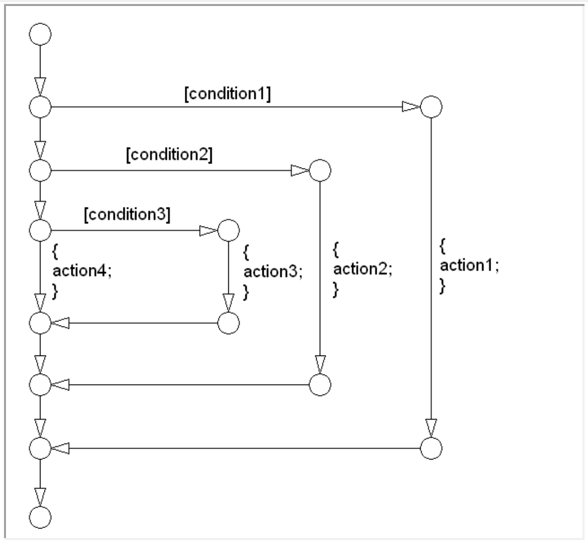




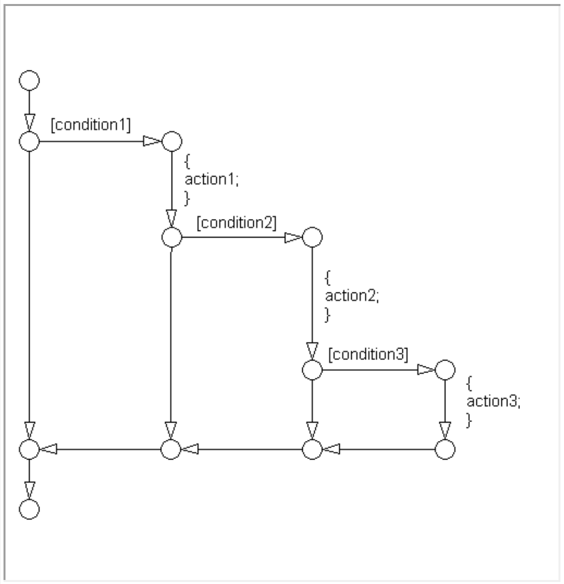
Alternately, use the following patterns for If-then-else-if constructs within Stateflow Flowcharts:

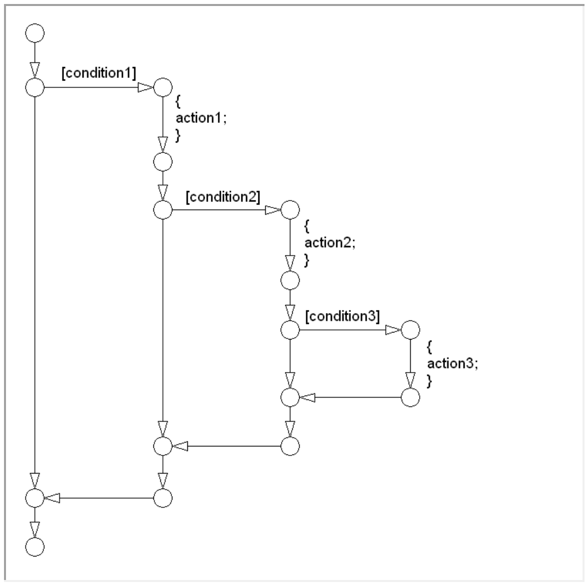
Straight Line Flow Chart Pattern	**Alternate Straight Line Flow Chart Pattern**
IF THEN ELSE IF	





Cascade of IF THEN





14.Obsolete rules

14.1. Removed in version 2.2

JM_0013 : Annotations : The rule was original written due to a printing bug in R13. The bug was fixed in R14 SP1.

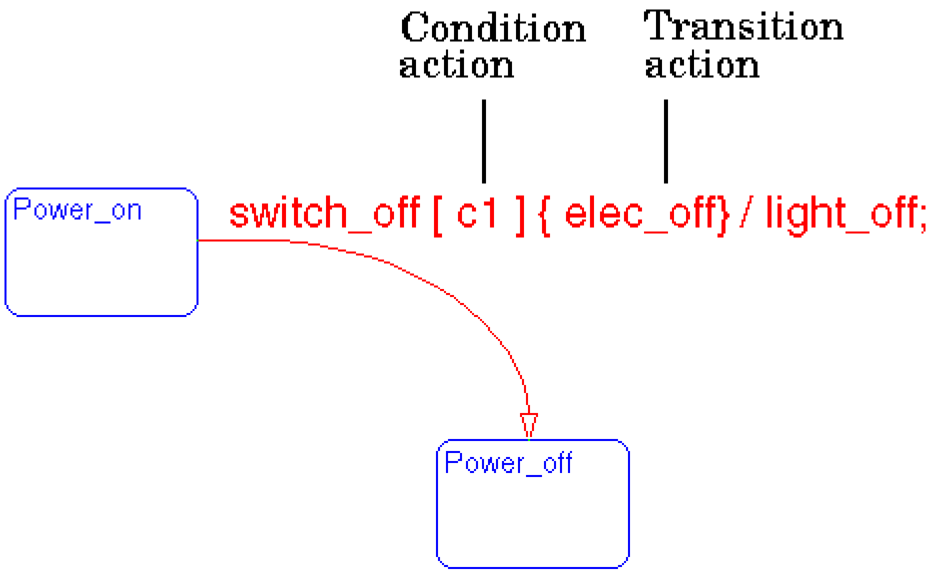
14.2. Removed in version 3.0

No guidelines were removed in version 3.0

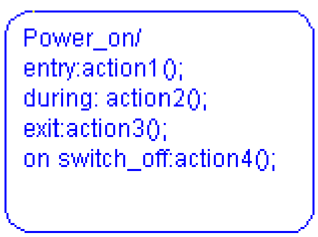
15.Glossary

Actions

Actions are part of Stateflow diagram execution. The action can be executed as part of a transition from one state to another, or depend 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,



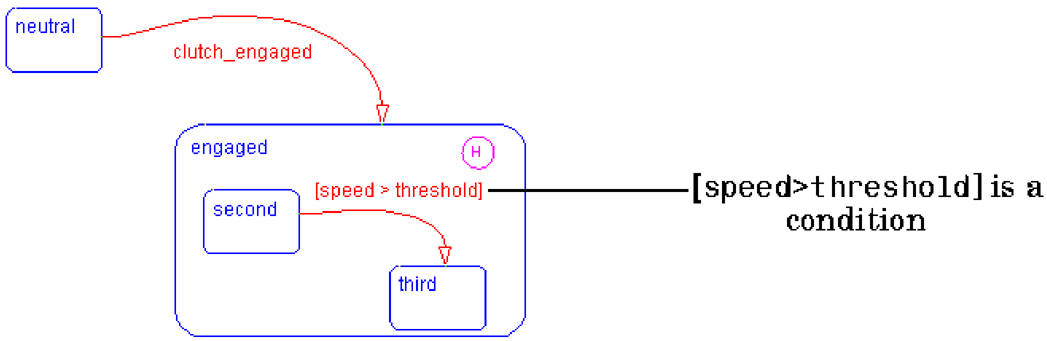
If you enter the name and backslash followed directly by an action or actions (without the entry keyword), the action(s) are interpreted as entry action(s). This shorthand is useful if you are only specifying entry actions.
The *action language* defines the categories of actions you can specify and their associated notations. An action can be a function call, an event to be broadcast, a variable to be assigned a value, etc.

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.

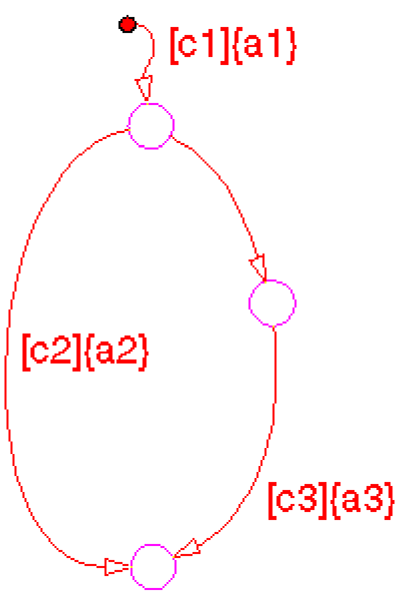
Chart Instance
A *chart instance* is a link from a Stateflow model to a chart stored in a Simulink library. A chart in a library can have many chart instances. Updating the chart in the library automatically updates all the instances of that chart.

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



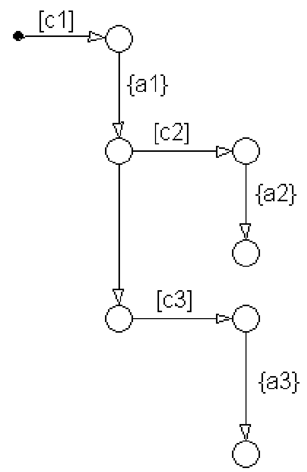
The action language defines the notation to define conditions associated with transitions.

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 desired system behavior.
This example shows how connective junctions (displayed as small circles) are used to represent the flow of an if code structure.

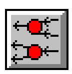



```
if [c1]{
      a1
      if [c2]{
          a2
      }else if [c3]{
        a3
      }
}
```

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. All events that affect the Stateflow diagram must be defined. 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 f*low subgraph* is the set of Flowcharts that start on the same transition segment.

Hierarchy

Hierarchy enables you to 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* provides the means to specify 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. Thus, a Stateflow model that includes links to library charts has multiple state machines. When Stateflow simulates a model that includes charts from a library model, it includes all charts from the library model even if there are links to only some of its models. However, when Stateflow generates a stand-alone or Real-Time Workshop® target, it includes only those charts for which there are links. A model that includes links to a library model can be simulated only if all charts in the library model are free of parse and compile errors.

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

Nonvirtual Block
Blocks that perform a calculation; such as a Gain block.

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

Stateflow notation consists of:

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

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

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

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

Real-Time Workshop Target
An executable built from code generated by Real-Time Workshop

S-Function
A customized Simulink block written in C or M-Code. C-code S-Functions can be inlined in Real-Time Workshop. 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.

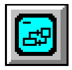
Signal propagation
Process used by Simulink to determine attributes of signals and blocks, such as data types, labels, sample time, dimensionality, and so on, that are determined by connectivity

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 multi-rate, i.e., have different parts that are sampled or updated at different rates.
It 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.

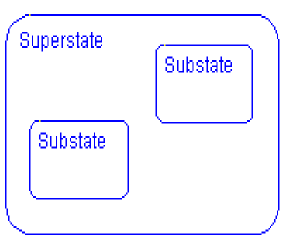
Stateflow Debugger
Use the *Stateflow Debugger* to debug and animate your Stateflow diagrams. Each state in the Stateflow diagram simulation is evaluated for overall code coverage. This coverage analysis is done automatically when the target is compiled and built with the debug options. The Debugger 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

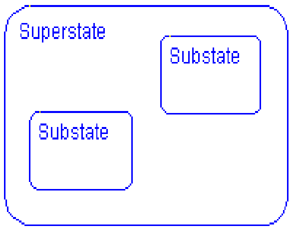
Stateflow Explorer
Use the *Stateflow Explorer* to add, remove, and modify data, event, and target objects.

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

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



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



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

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 parameters* is a parameter that can be adjusted both 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 since the signal source may be a simple routing block such as a demux block.

Virtual Block
When creating models, you need to 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 simply 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 the virtual and conditionally virtual blocks in Simulink.

Virtual Blocks

Block Name	Condition Under Which Block Will Be Virtual
Bus Selector	Virtual if input bus is virtual
Demux	Always virtual
Enable	Virtual unless connected directly to an Outport 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.
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

A *virtual scrollbar* enables you to 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.