CONTROL ALGORITHM MODELING
GUIDELINES USING MATLAB®,
Simulink®, and Stateflow®
Version 2.0

MathWorks Automotive Advisory Board
(MAAB)
July 27th, 2007
CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB®, SIMULINK®, AND STATEFLOW®

1. HISTORY

2. INTRODUCTION

2.1. MOTIVATION

2.1.1. Guideline template

2.1.2. Guideline ID:

2.1.3. Guideline Title:

2.1.4. Priority:

2.1.5. Scope:

2.1.6. MATLAB® Versions

2.1.7. Prerequisites:

2.1.8. Description:

2.1.9. Rationale:

2.1.10. Last change:

2.2. DOCUMENT USAGE

2.2.1. Guideline Interaction Semantics

3. NAMING CONVENTIONS

3.1. GENERAL GUIDELINES

3.1.1. ar_0001: Filenames

3.1.2. ar_0002: Directory names

3.2. MODEL CONTENT GUIDELINES

3.2.1. jc_0201: Usable characters for Subsystem name

3.2.2. jc_0211: Usable characters for Input block and Output block

3.2.3. jc_0221: Usable characters for signal line name

3.2.4. jc_0231: Usable characters for block names

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

4. MODEL ARCHITECTURE

4.1. SIMULINK® AND STATEFLOW® PARTITIONING

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

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

4.2. SUBSYSTEM HIERARCHIES

4.2.1. db_0143: Similar block types on the model levels

4.2.2. db_0144: Use of Subsystems

4.2.3. db_0040: Model hierarchy

4.3. J-MAAB MODEL ARCHITECTURE DECOMPOSITION

4.3.1. jc_0301: Controller model

4.3.2. jc_0311: Top layer / root level

4.3.3. jc_0321: Trigger layer

4.3.4. jc_0331: Structure layer

4.3.5. jc_0341: Data flow layer

5. MODEL CONFIGURATION OPTIONS

5.1. jc_0011: Optimization parameters for Boolean data types

5.1.2. jc_0021: Model diagnostic settings

6. SIMULINK

6.1. DIAGRAM APPEARANCE

6.1.1. na_0004: Simulink model appearance

6.1.2. db_0043: Simulink font and font size

6.1.3. db_0042: Port block in Simulink models
7. STATEFLOW

7.1. CHART APPEARANCE

7.1.1. db_0123: Stateflow port names
7.1.2. db_0129: Stateflow transition appearance
7.1.3. db_0137: States in state machines
7.1.4. db_0133: Use of patterns for Flowcharts
7.1.5. db_0132: Transitions in Flowcharts
7.1.6. jc_0501: Format of entries in a State block
7.1.7. jc_0511: Setting the return value from a graphical function
7.1.8. jc_0531: Placement of the default transition
7.1.9. jc_0521: Use of the return value from graphical functions

7.2. STATEFLOW DATA AND OPERATIONS

7.2.1. na_0001: Bitwise Stateflow operators
7.2.2. jc_0451: Use of unary minus on unsigned integers in Stateflow
7.2.3. na_0013: Comparison operation in Stateflow
1. History

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.04.2001</td>
<td>Initial document Release, Version 1.00</td>
</tr>
<tr>
<td>04.27.2007</td>
<td>Version 2.00 Update release</td>
</tr>
</tbody>
</table>
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. Respecting 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)

2.1.1. Guideline template

Guidelines are described with the following template. Companies who wish to create additional guidelines are encouraged to use the template.

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

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.

2.1.2. 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.

2.1.3. Guideline Title:
The title should be a short, but unique description of the guidelines area of application (e.g., length of names). The title is used for the "prerequisites"-field and for custom checker-tools. There should be a hyperlink with the title-text. It is used for links to the guideline. Note: The title should not be a redundant short description of the guidelines content, because while the latter may change over time, the title should remain stable.

2.1.4. Priority:
Each guideline must be rated with one of these priorities "mandatory", "strongly recommended" or "recommended." The priority not only describes the importance of the guideline but also determines the consequences of violations.

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Strongly Recommended</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEFINITION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• guidelines that all companies agree to that are absolutely essential</td>
<td>• guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming 100% to this guideline</td>
<td>• guidelines that are recommended to improve the appearance, but are not critical to running the model</td>
</tr>
<tr>
<td>• guidelines that all companies conform to 100%</td>
<td>• models should conform to these guidelines to the greatest extent possible, however 100% compliance is not required</td>
<td>• guidelines where conformance is preferred but not required</td>
</tr>
</tbody>
</table>

| **CONSEQUENCES** | | |
| If the guideline is violated | | |
| • essential things are missing | • the quality and the appearance will deteriorate | • the appearance will not conform with other projects |
| • the model may not work properly | • there may be an adverse effect on maintainability, portability, and reusability | |

| **WAIVER POLICY** | | |
| If the guideline is intentionally ignored | | |
| • the reasons must be documented | | |
2.1.5. 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, which work closely together with The MathWorks. MAAB includes the sub-groups J-MAAB, and NA-MAAB.

"J-MAAB" is a sub group of MAAB, and is a group of automotive manufacturers and suppliers in JAPAN, which works closely with The MathWorks. Rules with J-MAAB scope are local to Japan.

"NA-MAAB" is a sub group of MAAB and is a group of automotive manufacturers and suppliers in USA and Europe, which works closely with The MathWorks. That rule is local rule in USA and Europe. Coverage is USA and Europe.

2.1.6. MATLAB® Versions
The guidelines were written to support all versions of MATLAB and Simulink. If the rule applies to a specific version, or versions, it is denoted in this field. The versions information will be one of the following three formats.

• All : all versions of MATLAB
• RX, RY, RZ : a specific version of MATLAB
• RX and earlier : all versions of MATLAB until version RX
• RX and later: all versions of MATLAB from version RX to the current version
• RX through RY: all versions of MATLAB between RX and RY

2.1.7. Prerequisites:
This field is for links to other guidelines, which are prerequisite to this guideline (logical conjunction).
The guideline ID (for consistency) and the title (for readability) have to be used for the links.
The "Prerequisites" field should contain no other text.

2.1.8. Description:
The "Description" 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.

2.1.9. 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/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
  • Requirements Traceability
  • Testing
  • Problem-free system integration
  • Clean interfaces
• Code generation: Efficient/effective embedded code generation
  • Fast software changes
  • Robustness of generated code

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

2.2. Document Usage
The following paragraphs give some directions on how to use this document for reference and for compiling a project-specific guideline document. Information on automated checking of the guidelines can be found in Appendix A.

2.2.1. Guideline Interaction Semantics
The initial sections of the document, naming conventions and model architecture are basic guidelines that apply to all types of models. The later sections, Simulink and Stateflow deal with specific rules for those environments. Some guidelines are dependent on other guidelines, these; are explicitly listed throughout the template.
3. Naming Conventions

3.1. General Guidelines

3.1.1. ar_0001: Filenames

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>ar_0001: Filenames</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**

A filename conforms to the following constraints:

**NAME**
- **Form**: filename = name.extension
- **name**: no leading digits, no blanks
- **extension**: no blanks

**Uniqueness**: all filenames within the parent project directory

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

**Underscores**
- **name**: can use underscores to separate parts
- **extension**: should not use underscores

**Rationale**
- ✓ Readability
- ✓ Workflow
- □ Verification and Validation
- □ Code Generation
- □ Simulation

Last Change: V1.00

3.1.2. ar_0002: Directory names

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>ar_0002: Directory names</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**

A directory name conforms to the following constraints:
3.2. Model Content Guidelines

3.2.1. jc_0201: Usable characters for Subsystem name

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0201: Usable characters for Subsystem names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>strongly recommended</td>
</tr>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
<tr>
<td>MATLAB</td>
<td>All</td>
</tr>
<tr>
<td>Version</td>
<td>All</td>
</tr>
</tbody>
</table>

Prerequisites

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

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

**UNIQUENESS**
- all directory names within the parent project directory

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

**UNDERSCORES**
- 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
- ☑ Verification and Validation
- ☑ Simulation
- ☑ Code Generation

Last Change V1.00

Last Change V2.0
### 3.2.2. jc_0211: Usable characters for Inport block and Outport block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0211: Usable characters for Inport block and Outport block</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**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

**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
- 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
- name: 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
- Code Generation

**Last Change** V2.0

---

### 3.2.3. jc_0221: Usable characters for signal line name

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0221: Usable characters for signal line names</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

All named signals should conform to the following constraints:

**FORM**

- name:
  - should not start with a number
  - should not have blank spaces
  - should not have any control characters

**ALLOWED CHARACTERS**

- name: a b c d e f g h i j k l m n o p q r s t u v w x y z
- 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
- name: 0 1 2 3 4 5 6 7 8 9 _

**UNDERSCORES**

- name:
  - underscores can be used to separate parts
### 3.2.4. jc_0231: Usable characters for block names

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0231: Usable characters for block names</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

All named blocks should conform to the following constraints:

- **FORM**
  - should not start with a number
  - should not start with a blank space
  - may not use double byte characters
  - carriage returns are allowed

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

Note: this rule does not apply to Subsystem blocks.

**Rationale**

- Readability
- Workflow
- Code Generation

**Last Change** V2.0

### 3.2.5. na_0014: Use of local language in Simulink and Stateflow

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0014: Use of local language in Simulink and Stateflow</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Last Change** V2.0
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

![Simulink Block Properties](image)

- Text annotation directly entered in the model

![Simulink Model](image)

**Stateflow Example**
- The Description field of the chart or state Properties

![Stateflow State](image)
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.

<table>
<thead>
<tr>
<th>Rationale</th>
<th>Readability</th>
<th>Workflow</th>
<th>Code Generation</th>
<th>Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last Change V2.0
4. Model Architecture

Basic Blocks
This document uses the term “Basic Blocks” to refer to blocks from the base Simulink library; examples of basic blocks are shown below.

![Basic Blocks Diagram]

4.1. Simulink® and Stateflow® Partitioning

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

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0006: Guidelines for mixed use of Simulink and Stateflow</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
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, Stateflow should be used.
  - 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, Simulink should be used.

Specifics:
- If the primary nature of the function is logical, but some simple numerical calculations are done to support the logic, it is preferable to implement the simple numerical functions using the Stateflow action language.
- If the primary nature of the function is numerical, but some simple logical operations are done to support the arithmetic, it is preferable to implement the simple logical functions within Simulink.
If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, a Simulink subsystem should be used to implement the numerical calculations. Stateflow should invoke the execution of this subsystem using a function-call.
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 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 – that would be better modeled in Stateflow.)
Simulink should be used to implement numerical expressions containing continuously-valued states, e.g., difference equations, integrals, derivatives, and filters.

### Incorrect

![Incorrect Simulink model](Image)

### Correct

![Correct Simulink model](Image)

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

#### Last Change
V2.0
4.1.2. na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

Within Stateflow, the choice of whether to utilize 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, then state charts should be used. Some examples are a diagnostic model with pass, fail, abort, and conflict states, or a model that calculates different modes of operation for a control algorithm.
- If the primary nature of the function segment involves if-then-else statements, then flowcharts or truth tables should be used.

**Specifics:**

- If the primary nature of the function segment is to calculate modes or states, but if-then-else statements are required, it is recommended that a flow chart be added to a state within the state chart. (refer to 7.5 Flowchart Patterns)

**Rationale**

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

**Last Change** V2.0

4.2. Subsystem Hierarchies

4.2.1. db_0143: Similar block types on the model levels

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0143: Similar block types on the model levels</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

Every level of a model must be designed with building blocks of the same type. (i.e. only subsystems or only basic blocks).

**Blocks which can be placed on every model level:**
### 4.2.2. db_0144: Use of Subsystems

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0144: Use of Subsystems</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

#### Prerequisites

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

Grouping blocks into subsystems primarily for the purpose of saving space in the diagram should be avoided. Each subsystem in the diagram should represent a unit of functionality required to accomplish the purpose of the model or sub model.

#### Rationale

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

#### Last Change

V2.0
4.2.3. db_0040: Model hierarchy

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0040: Model hierarchy</th>
</tr>
</thead>
<tbody>
<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 model hierarchy should correspond to the functional structure of the control system.</td>
</tr>
<tr>
<td>Rationale</td>
<td>☑ Readability</td>
</tr>
<tr>
<td></td>
<td>☑ Verification and Validation</td>
</tr>
<tr>
<td></td>
<td>☑ Workflow</td>
</tr>
<tr>
<td></td>
<td>☑ Code Generation</td>
</tr>
<tr>
<td></td>
<td>☐ Simulation</td>
</tr>
<tr>
<td>Last Change</td>
<td>V2.0</td>
</tr>
</tbody>
</table>

4.3. J-MAAB Model Architecture Decomposition

4.3.1. jc_0301: Controller model

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0301: Controller model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</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>Control models are organized using the following hierarchical structure. Details on each layer are provided in the latter rules.</td>
</tr>
<tr>
<td></td>
<td>- Top layer / root level</td>
</tr>
<tr>
<td></td>
<td>- Trigger layer</td>
</tr>
<tr>
<td></td>
<td>- Structure layer</td>
</tr>
<tr>
<td></td>
<td>- Data flow layer</td>
</tr>
<tr>
<td></td>
<td>Use of the Trigger level is optional. In the diagram below &quot;Type A&quot; shows the use of a trigger level while &quot;Type B&quot; shows a model without a trigger level.</td>
</tr>
</tbody>
</table>
### 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

![Top Layer Example Diagram]

---

### Rationale

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

---

### Last Change

V2.0
### 4.3.3. jc_0321: Trigger layer

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0321: Trigger layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</td>
</tr>
<tr>
<td>Scope</td>
<td>J-MAAB</td>
</tr>
<tr>
<td>MATLAB Version</td>
<td>All</td>
</tr>
</tbody>
</table>

**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](image)

**Rationale**

- Readability
- Workflow
- Code Generation

**Last Change** V2.0

### 4.3.4. jc_0331: Structure layer

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0331: Structure layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</td>
</tr>
<tr>
<td>Scope</td>
<td>J-MAAB</td>
</tr>
<tr>
<td>MATLAB Version</td>
<td>All</td>
</tr>
</tbody>
</table>

**Prerequisites**

**Description**

1. Describe a structure layer like the following description example.

   - In case of Type B, specify sample time at a 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

2. Subsystem of a structure layer should be Atomic Subsystem.
4.3.5. jc_0341: Data flow layer

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0341: Data flow layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</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>Describe a data flow layer as in the following example.</td>
</tr>
<tr>
<td></td>
<td>- 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</td>
</tr>
</tbody>
</table>
Data Flow Layer Example

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☐ Readability</th>
<th>☐ Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☑ Workflow</td>
<td>☑ Code Generation</td>
</tr>
<tr>
<td></td>
<td>☐ Simulation</td>
<td></td>
</tr>
</tbody>
</table>

Last Change V2.0
5. Model Configuration Options

5.1.1. jc_0011: Optimization parameters for Boolean data types

<table>
<thead>
<tr>
<th>ID:Title</th>
<th>jc_0011: Optimization parameters for Boolean data types</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

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

<table>
<thead>
<tr>
<th>MATLAB version</th>
<th>Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R13SP2 and earlier</td>
<td>Boolean Logic signals</td>
</tr>
<tr>
<td></td>
<td>Boolean logic signals</td>
</tr>
<tr>
<td>R14 and later</td>
<td>Use logic signals as Boolean data (vs. double)</td>
</tr>
<tr>
<td></td>
<td>✅ Implement logic signals as boolean data (vs. double)</td>
</tr>
</tbody>
</table>

**Rationale**

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

**Last Change**

V2.0

5.1.2. jc_0021: Model diagnostic settings

<table>
<thead>
<tr>
<th>ID:Title</th>
<th>jc_0021: Model diagnostic settings</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Rationale</th>
<th>□ Readability</th>
<th>☑ Workflow</th>
<th>□ Verification and Validation</th>
<th>☑ Code Generation</th>
<th>□ Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Change</td>
<td>V2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 6. Simulink

#### 6.1. Diagram Appearance

##### 6.1.1. na_0004: Simulink model appearance

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0004 Simulink model appearance</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>View Options</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Browser</td>
<td>unchecked</td>
</tr>
<tr>
<td>Screen color</td>
<td>white</td>
</tr>
<tr>
<td>Status Bar</td>
<td>checked</td>
</tr>
<tr>
<td>Toolbar</td>
<td>checked</td>
</tr>
<tr>
<td>Zoom factor</td>
<td>Normal (100%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Display Options</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Color</td>
<td>white</td>
</tr>
<tr>
<td>Foreground Color</td>
<td>black</td>
</tr>
<tr>
<td>Execution Context Indicator</td>
<td>unchecked</td>
</tr>
<tr>
<td>Library Link Display</td>
<td>none</td>
</tr>
<tr>
<td>Linearization Indicators</td>
<td>checked</td>
</tr>
<tr>
<td>Model/Block I/O Mismatch</td>
<td>unchecked</td>
</tr>
<tr>
<td>Model Block Version</td>
<td>unchecked</td>
</tr>
<tr>
<td>Sample Time Colors</td>
<td>unchecked</td>
</tr>
<tr>
<td>Sorted Order</td>
<td>unchecked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal Display Options</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port Data Types</td>
<td>unchecked</td>
</tr>
<tr>
<td>Signal Dimensions</td>
<td>unchecked</td>
</tr>
<tr>
<td>Storage Class</td>
<td>unchecked</td>
</tr>
<tr>
<td>Test point Indicators</td>
<td>checked</td>
</tr>
<tr>
<td>Viewer Indicators</td>
<td>checked</td>
</tr>
<tr>
<td>Wide Non-scalar Lines</td>
<td>checked</td>
</tr>
</tbody>
</table>

**Rationale**

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

**Last Change**

| V2.0 |
### 6.1.2. db_0043: Simulink font and font size

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0043: Simulink font and font size</th>
</tr>
</thead>
<tbody>
<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>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).</td>
</tr>
<tr>
<td>Rationale</td>
<td>☑ Readability ☑ Workflow ☐ Code Generation ☐ Simulation</td>
</tr>
<tr>
<td>Last Change</td>
<td>V2.0</td>
</tr>
</tbody>
</table>
6.1.3. db_0042: Port block in Simulink models

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0042: Port block in Simulink models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>strongly recommended</td>
</tr>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
<tr>
<td>MATLAB</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>All</td>
</tr>
<tr>
<td>Prerequisites</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

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.
  - Duplicate Inports cannot be used at the root level.

**Correct**

![Correct Diagram]

**Incorrect**

![Incorrect Diagram]

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

<table>
<thead>
<tr>
<th>Rationale</th>
<th>Readability</th>
<th>Verification and Validation</th>
<th>Workflow</th>
<th>Code Generation</th>
<th>Simulation</th>
<th>Simulation</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Last Change</th>
<th>V2.0</th>
</tr>
</thead>
</table>
6.1.4. na_0005: Port block name visibility in Simulink models

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0005: Port block name visibility in Simulink models</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

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

**Organizational-Scoped Alternatives (follow one practice):**

1. The name of an Inport or Outport is not hidden. ("Format / Hide Name" is not allowed.)

   ![Diagram 1]

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.*

   ![Diagram 2]

**Rationale**

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

**Last Change** V2.0

6.1.5. jc_0081: Icon display for Port block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0081: Icon display for Port block</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**
The ‘Icon display’ setting should be set to ‘Port number’ for Inport and Outport.
6.1.6. jm_0002: Block resizing

**ID: Title** jm_0002: Block resizing

**Priority** mandatory

**Scope** MAAB

**MATLAB Version** All

**Prerequisites**

All blocks in a model must be sized such that their icon is completely visible and recognizable. In particular, any text displayed (e.g. tunable parameters, filenames, 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, the user may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. In this approach, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block.

**Correct**

![Correct diagram]

**Incorrect**

![Incorrect diagram]
### 6.1.7. db_0142: Position of block names

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0142: Position of block names</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**
If shown the name of each block should be placed below the block.

**Description**

#### Correct

![Correct Diagram](image)

#### Incorrect

![Incorrect Diagram](image)

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

**Last Change** V2.0

### 6.1.8. jc_0061: Display of block names

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0061: Display of block names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>recommended</td>
</tr>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
<tr>
<td>MATLAB</td>
<td>All</td>
</tr>
</tbody>
</table>

**Prerequisites**

**Description**

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

**Last Change** V2.0
### Prerequisites

- The block name should be displayed when it provides descriptive information.

- The block name should not be displayed if the block function is known from its appearance.

### Description

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☑ Readability</th>
<th>☐ Verification and Validation</th>
<th>☐ Workflow</th>
<th>☐ Code Generation</th>
<th>☐ Simulation</th>
</tr>
</thead>
</table>

### 6.1.9. db_0146: Triggered, enabled, conditional Subsystems

**ID: Title** db_0146: Triggered, enabled, conditional Subsystems

**Priority** strongly recommended

**Scope** MAAB

**MATLAB Version** All

**Prerequisites**

- 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 are:
  - Function call
  - Enabled
  - Triggered
  - If / Else Action

**Correct**
### 6.1.10. db_0140: Display of basic block parameters

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>db_0140: Display of basic block parameters</td>
</tr>
</tbody>
</table>

- **Priority**: recommended
- **Scope**: MAAB
- **MATLAB Version**: All

**Prerequisites**: Important parameters with values other than the block’s default values should be displayed. Note: The attribute string is one method to support this. The block annotation tab allows the users to add the desired attribute information.

**Correct**
### 6.1.11. jm_0013: Annotations

**ID: Title**  jm_0013: Annotations

**Priority**  strongly recommended

**Scope**  MAAB

**MATLAB Version**  R12.1

**Prerequisites**

**Description**

Annotations should not have a drop shadow on them. ("Format / Show Drop Shadow" is not allowed.)

- **This is a correct annotation**
- **This is an incorrect annotation**

**Rationale**

- Readability
- Verification and Validation

**Last Change**  V2.0

### 6.1.12. db_0032: Simulink signal appearance

**ID: Title**  db_0032: Simulink signal appearance

**Priority**  strongly recommended

**Scope**  MAAB

**MATLAB Version**  All

**Last Change**  V2.0
Prerequisites

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.

Description

Correct
- Signal flow should be drawn from left to right

Incorrect
- Signal flow should be drawn from left to right

Rationale
- Readability
- Verification and Validation

Last Change V2.0

6.1.13. 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
- The signal flow in a model is from left to right.
  - Exception: Feedback loops
- Sequential blocks or subsystems are arranged from left to right.
  - Exception: Feedback loops
- Parallel blocks or subsystems are arranged from top to bottom.
6.1.14. jc_0171: Maintaining signal flow when using Goto and From blocks

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0171: Maintaining signal flow when using Goto and From blocks</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Prerequisites

- Visual depiction of signal flow must be maintained between subsystems.
- Use of Goto and From blocks is allowed provided that
  - At least one signal line is used between connected subsystems.
  - If the subsystems are connected both in a feedforward and feedback loop then at least one signal line for each direction must be connected.

Correct

Incorrect

Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Change V2.0
### 6.1.15. jm_0010: Port block names in Simulink models

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jm_0010: Port block names in Simulink models</th>
</tr>
</thead>
<tbody>
<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_0042: Ports in Simulink models</td>
</tr>
<tr>
<td></td>
<td>na_0005: Port block name visibility in Simulink models</td>
</tr>
</tbody>
</table>

#### Description

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

1. The names of Inport blocks and Outport blocks must match the corresponding signal or bus names.
   - **Exceptions:**
     - When any combination of an Inport block, an Outport block, and any other block have the same block name, 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, the user should apply a consistent naming practice for these blocks. Suggested practices include leaving the names as their default names (e.g., 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.

<table>
<thead>
<tr>
<th>Rationale</th>
<th>Readability</th>
<th>Workflow</th>
<th>Verification and Validation</th>
<th>Simulation</th>
<th>Code Generation</th>
<th>Simulation</th>
</tr>
</thead>
</table>

### 6.1.16. jc_0281: Naming of Trigger Port block and Enable Port block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0281: Naming of Trigger Port block and Enable Port block</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

#### Description

For Trigger port blocks and Enable port blocks
- The block name should match the name of the signal triggering the subsystem.
6.2. Signals

Signal labels are used to make model functionality more understandable from the Simulink diagram. They can also be used to control the variable names used in simulation and code generation. Signal labels should be entered only once (at the point of signal origination). Often it is desirable 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.

Signals may be scalars, vectors, or busses. They may carry data or control flows. Unless explicitly stated otherwise, the following naming rules apply to all types of signals.

6.2.1. na_0008: Display of labels on signals

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0008: Display of labels on signals</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

A label must be displayed on any signal originating from the following blocks:

- Inport block
- From block (block icon exception applies – see Note below)
- Data Store Read block (block icon exception applies)
- Subsystem block or Stateflow chart block (block icon exception applies)
- Constant block (block icon exception applies)
- Bus Selector block (the tool forces this to happen)
- Demux block
- Selector block

A label must be displayed on any signal connected to the following destination blocks (directly or via a basic block that performs a non transformative 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 need not also have the label displayed unless the signal label is needed elsewhere due to a destination-based rule.

- In addition, a label may be displayed on any other signal of interest to the user or the user's customers.

![Diagram](image)

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☑ Readability</th>
<th>☑ Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☑ Workflow</td>
<td>☑ Code Generation</td>
</tr>
<tr>
<td></td>
<td>☐ Simulation</td>
<td></td>
</tr>
</tbody>
</table>

Last Change | V2.0

6.2.2. na_0009: Entry versus propagation of signal labels

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>Na_0009: Entry versus propagation of signal labels</th>
</tr>
</thead>
<tbody>
<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_0008: Display of labels on signals</td>
</tr>
</tbody>
</table>
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.

![Diagram showing signal flow and label placement]

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☑️ Readability</th>
<th>☑️ Verification and Validation</th>
<th>☑️ Workflow</th>
<th>☑️ Code Generation</th>
<th>☐ Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Change</td>
<td>V2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.2.3. db_0097: Position of labels for signals and busses

<table>
<thead>
<tr>
<th><strong>ID</strong>: Title</th>
<th><strong>db_0097: Position of labels for signals and busses</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>strongly recommended</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>MAAB</td>
</tr>
<tr>
<td><strong>MATLAB</strong></td>
<td>All</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>All</td>
</tr>
<tr>
<td><strong>Prerequisites</strong></td>
<td></td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Rationale</th>
<th>✓ Readability</th>
<th>☐ Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ Workflow</td>
<td>☐ Code Generation</td>
</tr>
<tr>
<td></td>
<td>☐ Simulation</td>
<td></td>
</tr>
</tbody>
</table>

Last Change  V2.0

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

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0081: Unconnected signals and block inputs / outputs</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Prerequisites

Description

A system must not have any:
- Unconnected subsystem or basic block inputs.
- Unconnected subsystem or basic block outputs
- Unconnected signal lines
- An otherwise unconnected input should be connected to a ground block
- An otherwise unconnected output should be connected to a terminator block

Correct

Incorrect

<table>
<thead>
<tr>
<th>Rationale</th>
<th>✓ Readability</th>
<th>☐ Verification and Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ Workflow</td>
<td>☐ Code Generation</td>
</tr>
<tr>
<td></td>
<td>☐ Simulation</td>
<td></td>
</tr>
</tbody>
</table>

Last Change  V2.0

6.3. Block Usage

6.3.1. na_0003: Simple logical expressions in If Condition block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0003: Simple logical expressions in If Condition block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</td>
</tr>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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 below for 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 include:                                    | - u1  
- 5  
- K  
- (u1 > 0)  
- (u1 <= G)  
- (u1 > U2)  
- (~u1) |
| Examples of acceptable logical expressions include:                         | - u1 | u2  
- (u1 > 0) & (u1 < 20)  
- (u1 > 0) & (u2 < u3)  
- (u1 > 0) & (~u2) |
| Examples of unacceptable logical expressions include:                       | - u1 & u2 | u3 | (too many primary expressions)  
- 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) |
6.3.2. 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**

- 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.

**Description**

- 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.
### 6.3.3. jm_0001: Prohibited Simulink standard blocks inside controllers

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jm_0001: Prohibited Simulink standard blocks inside controllers</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

Controller models must be designed from discrete blocks.

Sources are not allowed:

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

Continuous blocks are not allowed:
Integrator  Derivative  Transport Delay  Variable Transport Delay  State-Space  Transfer Fcn  Zero-Pole

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

<table>
<thead>
<tr>
<th>Rationale</th>
<th>□ Readability</th>
<th>□ Workflow</th>
<th>□ Verification and Validation</th>
<th>□ Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Change</td>
<td>V2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.3.4. hd_0001: Prohibited Simulink sinks

**ID:** Title  | hd_0001: Prohibited Simulink sinks
---|---
Priority  | strongly recommended
Scope  | MAAB
MATLAB Version  | All
Prerequisites

**Description**

Controller models must be designed from discrete blocks.

**Sinks are not allowed:**

- Scope
- XY Graph
- Display
- To File
- To Workspace
- Stop Simulation
- Floating Scope

**Rationale**

- Readability
- Workflow
- Verification and Validation
- Code Generation

**Last Change** V2.0

6.3.5. 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.

![Sink Block Parameters: Goto](image.png)
6.3.6. 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**

The switch condition, input 2, must be a Boolean value. The block parameter “Criteria for passing first input” should be set to \( u2 \neq 0 \).

The block parameter “Criteria for passing first input” must not be set to \( u2 > \text{Threshold} \) for R13 versions of MATLAB.

**Correct**

![Correct Switch Block Diagram]

**Incorrect**

![Incorrect Switch Block Diagram]
6.3.7. jc_0121: Use of the Sum block

**ID: Title** Jc_0121: Use of the Sum block

<table>
<thead>
<tr>
<th>Priority</th>
<th>recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
<tr>
<td>MATLAB Version</td>
<td>All</td>
</tr>
</tbody>
</table>

**Prerequisites**

**Description**

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

- The round shape can be used 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.
6.3.8. jc_0131: Use of Relational Operator block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0131: Use of Relational Operator block</th>
</tr>
</thead>
<tbody>
<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>When the relational operator is used to compare a signal to a constant value the constant input should be the second (lower) input.</td>
</tr>
</tbody>
</table>
6.3.9. 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 Read  
- Data Store Write  
- Data Store Memory

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

**Rationale**
- Readability  
- Workflow  
- Code Generation

**Last Change** V2.0

---

6.4. Block Parameters

**6.4.1. db_0112: Indexing**

**ID: Title** db_0112: Indexing  
**Priority** strongly recommended  
**Scope** MAAB  
**MATLAB Version** All  
**Prerequisites**

**Description**
- One based indexing [1, 2, 3, …] is used for  
  - MATLAB  
  - Workspace variables and structures  
  - Local variables of m-functions
6.4.2. na_0010: Grouping data flows into signals

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0010: Grouping data flows into signals</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

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

Busses
Signals that do not meet the vectorization criteria described above must only be grouped into bus signals. 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:
Row vector \[1 \times n\]  
Column vector \[n \times 1\]  
Wheel speed vector \[1 \times \text{Number of wheels}\]  
Cylinder vector \[1 \times \text{Number of cylinders}\]  
Position vector based on 2-D coordinates \[1 \times 2\]  
Position vector based on 3-D coordinates \[1 \times 3\]  

Some examples of bus signals include:

<table>
<thead>
<tr>
<th>Bus Type</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Bus</td>
<td>Force Vector ([F_x, F_y, F_z])</td>
</tr>
<tr>
<td></td>
<td>Position</td>
</tr>
<tr>
<td></td>
<td>Wheel Speed Vector ([\theta_{lf}, \theta_{rf}, \theta_{lr}, \theta_{rr}])</td>
</tr>
<tr>
<td></td>
<td>Acceleration</td>
</tr>
<tr>
<td></td>
<td>Pressure</td>
</tr>
<tr>
<td>Controller Bus</td>
<td>Sensor Bus</td>
</tr>
<tr>
<td></td>
<td>Actuator Bus</td>
</tr>
<tr>
<td>Serial Data Bus</td>
<td>Coolant Temperature</td>
</tr>
</tbody>
</table>

- **Rationale**: Readability; Workflow
- **Last Change**: V2.0

### 6.4.3. db_0110: Tunable parameters in basic blocks

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0110: Tunable parameters in basic blocks</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

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

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

**Correct**

- `tunable_parameter_value`
- `tunable_parameter_vector`
- `tunable_parameter_array`

**Incorrect**

- `tunable_parameter_value*2`
- `tunable_parameter_vector*3`
- `tunable_parameter_array*3`
### 6.5. 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.

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

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>na_0012: Use of Switch vs. If-Then-Else Action Subsystem</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**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.

```
 IF_Value double

 IF_Condition

 Else_Value double
```

**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, can require fairly complicated computations.)

```
 DynamicSlipFlag

 If

 ((u1)

 else

 if

 [ ] Out

 TireSlipConst

 WheelerSpeed EngSpeed

 out:

 Merge TireSlip

 CalculateTireSlip

```
- 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 will determine the appropriate construct.

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

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☑ Readability</th>
<th>☑ Workflow</th>
<th>☑ Verification and Validation</th>
<th>☑ Code Generation</th>
<th>☑ Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Change</td>
<td>V2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0114: Simulink patterns for If-then-else-if constructs</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Description

The following patterns should be used for If-then-else-if constructs within Simulink:

<table>
<thead>
<tr>
<th>Equivalent Functionality</th>
<th>Simulink pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF THEN ELSE IF with blocks</td>
<td>![Simulink Diagram]</td>
</tr>
</tbody>
</table>

```java
if (If_Condition) {
    output_signal = If_Value;
} else if (Else_If_Condition) {
    output_signal = Else_If_Value;
} else {
    output_signal = Else_Value;
}
```
IF THEN ELSE IF
with if/then/else subsystems:
if(Fault_1_Active & Fault_2_Active)
{
    ErrMsg = SafetyCrit;
}
else if (Fault_1_Active | Fault_2_Active)
{
    ErrMsg = DriveWarn;
}
else
{
    ErrMsg = NoFaults;
}

Rationale
- Readability
- Workflow
- Code Generation

Last Change V2.0

6.5.3. db_0115: Simulink patterns for case constructs

| ID: Title | db_0115: Simulink patterns for case constructs |
| Priority | strongly recommended |
| Scope | MAAB |
| MATLAB Version | All |

Prerequisites

The following patterns are used for case constructs within Simulink:

**Equivalent Functionality**

```matlab
switch (PRNDL_Enum)
    case 1
        TqEstimate = ParkV;
        break;
    case 2
        TqEstimate = RevV;
        break;
    default
        TqEstimate = NeutralV;
        break;
end
```

**Simulink Pattern**
CASE with subsystems:

\[
\text{output\_version1} = \text{function\_version1}(\text{input\_signal}); \\
\text{output\_version2} = \text{function\_version2}(\text{input\_signal}); \\
\text{output\_version3} = \text{function\_version3}(\text{input\_signal}); \\
\text{output\_version4} = \text{function\_version4}(\text{input\_signal});
\]

switch (selection) {
    case 1:
        \text{output\_signal} = \text{output\_version1};
        break;
    case 2:
        \text{output\_signal} = \text{output\_version2};
        break;
    case 3:
        \text{output\_signal} = \text{output\_version3};
        break;
    case 4:
        \text{output\_signal} = \text{output\_version4};
    }

CASE with enabled subsystems:

switch (selection) {
    case 1:
        \text{output\_version1} = \text{function\_version1}(\text{input\_signal});
        \text{output\_signal} = \text{output\_version1};
        break;
    case 2:
        \text{output\_version2} = \text{function\_version2}(\text{input\_signal});
        \text{output\_signal} = \text{output\_version2};
        break;
    case 3:
        \text{output\_version3} = \text{function\_version3}(\text{input\_signal});
        \text{output\_signal} = \text{output\_version3};
        break;
    default:
        \text{output\_version4} = \text{function\_version4}(\text{input\_signal});
        \text{output\_signal} = \text{output\_version4};
    }

Rationale:
- ☑ Readability
- ☑ Workflow
- ☑ Verification and Validation
- ☐ Code Generation

Last Change: V2.0
6.5.4. db_0116: Simulink patterns for logical constructs with logical blocks

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0116: Simulink patterns for logical constructs with logical blocks</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

The following patterns are used for logical combinations within Simulink:

<table>
<thead>
<tr>
<th>Equivalent Functionality</th>
<th>Simulink pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combination of logical signals: conjunctive</td>
<td>![Diagram of conjunctive combination]</td>
</tr>
<tr>
<td>Combination of logical signals: disjunctive</td>
<td>![Diagram of disjunctive combination]</td>
</tr>
</tbody>
</table>

**Rationale**

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

Last Change V1.00

6.5.5. db_0117: Simulink patterns for vector signals

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0117: Simulink patterns for vector signals</th>
</tr>
</thead>
</table>

62
The following patterns are used for vector signals within Simulink:

<table>
<thead>
<tr>
<th>Equivalent Functionality</th>
<th>Simulink Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_value; }</td>
<td><img src="image1" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) * tunable_parameter_vector(i); }</td>
<td><img src="image2" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>output_signal = 1;</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_signal = output_signal * input_vector(i); }</td>
<td><img src="image3" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>output_signal = 1;</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_signal = output_signal / input_vector(i); }</td>
<td><img src="image4" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_value; }</td>
<td><img src="image5" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_vector(i) = input_vector(i) + tunable_parameter_vector(i); }</td>
<td><img src="image6" alt="Vector loop in Simulink" /></td>
</tr>
<tr>
<td>Vector loop:</td>
<td></td>
</tr>
<tr>
<td>output_signal = 0;</td>
<td></td>
</tr>
<tr>
<td>for (i=0; i&lt;input_vector_size; i++) { output_signal = output_signal + input_vector(i); }</td>
<td><img src="image7" alt="Vector loop in Simulink" /></td>
</tr>
</tbody>
</table>
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 ☑ Verification and Validation
☑ Workflow ☑ Code Generation
☐ Simulation

Last Change V1.00

6.5.6. jc_0351: Methods of initialization

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0351: Methods of initialization</th>
</tr>
</thead>
<tbody>
<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_0140: Display of block parameters</td>
</tr>
</tbody>
</table>

**Description**

**Simple initialization:**
- Blocks such as the Unit Delay, that 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:
For complex initializations the following rules hold.
- The initialization should be performed in a separate subsystem.
- The initialization subsystem should have a name that indicates that initialization is performed by the subsystem.

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

Example A

Example B

6.5.7. jc_0111: Direction of Subsystem

<table>
<thead>
<tr>
<th>Rationale</th>
<th>☐ Readability</th>
<th>☑ Workflow</th>
<th>☐ Verification and Validation</th>
<th>☐ Code Generation</th>
</tr>
</thead>
</table>

Last Change V2.0

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0111: Direction of Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>strongly recommended</td>
</tr>
<tr>
<td>Scope</td>
<td>J-MAAB</td>
</tr>
<tr>
<td>MATLAB</td>
<td>All</td>
</tr>
</tbody>
</table>
Subsystem must not be reversed.

**Correct**

**Incorrect**

**Rationale**
- [x] Readability
- [ ] Verification and Validation
- [ ] Workflow
- [ ] Code Generation
- [ ] Simulation

**Last Change** V2.0
# 7. Stateflow

## 7.1. Chart Appearance

### 7.1.1. db_0123: Stateflow port names

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0123: Stateflow port names</th>
</tr>
</thead>
<tbody>
<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 name of a Stateflow input/output should be the same as the corresponding signal. Exception: Reusable Stateflow blocks may have different port names.</td>
</tr>
</tbody>
</table>
| Rationale               | ✔ Readability  ☐ Verification and Validation  
  ✔ Workflow  ☐ Code Generation  
  ☐ Simulation                                                        |
| Last Change             | V1.00                                                                                          |

### 7.1.2. db_0129: Stateflow transition appearance

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0129: Stateflow transition appearance</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
| Description             | Transitions in Stateflow:  
  • Do not cross each other, if possible.  
  • Are not drawn one upon the other.  
  • Do not cross any states, junctions or text fields.  
  • Are allowed if transitioning to an internal state.  
  Transition labels can be visually associated to the corresponding transition. Correct |
| Correct                 |                                                                                               |
7.1.3. db_0137: States in state machines

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0137: States in state machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</td>
</tr>
<tr>
<td>Scope</td>
<td>MAAB</td>
</tr>
</tbody>
</table>

Incorrect Rationale:
- Readability
- Workflow
- Simulation
In state machines:

- There are at least two exclusive states.
- A state cannot have only one substate.
- The initial state of a hierarchical level with exclusive states is clearly defined by a default transition.

A transition in a Flowchart has a condition, a condition action or an empty transition.

- Conditions are drawn on the horizontal.
- Actions are drawn on the vertical.
- Loop constructs are intentional exceptions to this rule.
Transition with condition action:

```
[condition]
```

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:

```
/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:

```
/* comment */
```

Rationale | ☑️ Readability | ☑️ Verification and Validation
7.1.6. jc_0501: Format of entries in a State block

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_501: Format of entries in a State block</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

A new line should be:

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

**Correct**

```matlab
State
en:
    entry_value=1;
    during_value=0;
    du:
    entry_value=0;
    during_value=1;
ex:
    exit_value=1;
```
7.1.7. jc_0511: Setting the return value from a graphical function

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0511: Setting the return value from a graphical function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>mandatory</td>
</tr>
<tr>
<td>Scope</td>
<td>J-MAAB</td>
</tr>
<tr>
<td>MATLAB Version</td>
<td>All</td>
</tr>
</tbody>
</table>

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

Correct
Return value A is set in one place

Incorrect
Return value A is set in multiple places.
7.1.8. jc_0531: Placement of the default transition

**ID: Title**  jc_0531: Placement of the default transition

**Priority**  recommended

**Scope**  J-MAAB

**MATLAB Version**  All

**Prerequisites**

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

### Correct

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

### Incorrect

- Default transition is connected at the side of the state (State 1).
- The destination state of the default transition is lower than the other states in the same hierarchy (SubSt_off).
### 7.1.9. jc_0521: Use of the return value from graphical functions

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0521: Use of the return value from graphical functions</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**

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".

```
function
    temp_k = temp_test()
```

```
{ a = temp_test()

[a == 1]
```

The data type of the variable in the comparison operation is clear.

**Incorrect**

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

```
function
    temp_k = temp_test()

[temp_test() == 1 ]
```
7.2. Stateflow data and operations

7.2.1. 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 bitwise operations are desired.

If bitwise operations are desired, the “Enable C-bit Operations” needs to be enabled.

1. From the File Menu \ Chart Properties.
2. Select Enable C-bit operations.

Correct
Use “&&” and “II” for Boolean operation.

Use “&” and “I” for bit operation.
7.2.2. 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**

```c
si16_var1 = -si16_var2;
```

**Incorrect**

```c
ui16_var1 = -ui16_var2;
```

**Rationale**

- Readability
- Workflow
- Code Generation

**Last Change** V 2.0

---

7.2.3. na_0013: Comparison operation in Stateflow

**ID: Title** na_0013: Comparison operation in Stateflow

**Priority** recommended

**Scope** MAAB

**MATLAB Version** All
### Prerequisites

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

<table>
<thead>
<tr>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same data type in &quot;i&quot; and &quot;n&quot;</td>
<td>Different data type in &quot;i&quot; and &quot;d&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different data type in &quot;i&quot; and &quot;d&quot;</td>
</tr>
</tbody>
</table>

• Do not make comparisons between unsigned integers and negative numbers.

### Description

<table>
<thead>
<tr>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different data type in &quot;i&quot; and &quot;d&quot;</td>
</tr>
</tbody>
</table>

### Rationale

- [ ] Readability
- [x] Workflow
- [ ] Code Generation
- [ ] Simulation
- [ ] Verification and Validation

### Last Change

V2.0

7.2.4. db_0122: Stateflow and Simulink interface signals and parameters

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0122: Stateflow and Simulink interface signals and parameters</th>
</tr>
</thead>
<tbody>
<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>A Chart uses strong data typing with Simulink (The option &quot;Use Strong Data Typing with Simulink I/O&quot; must be selected).</td>
</tr>
</tbody>
</table>
7.2.5. 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

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.
- There must be no local variables on the machine level (i.e. there is no interaction between local data in different charts).
- Parameters and constants are allowed at the machine level.

### Description

**Correct**

**Incorrect**
7.2.6. jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**

- 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 can be done between two integer data types.

**Correct**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>double</td>
</tr>
<tr>
<td>d2</td>
<td>double</td>
</tr>
</tbody>
</table>

```matlab
diff = (d1 - d2);

(-LIMIT <= diff) & (diff <= LIMIT)
```

**Incorrect**

```matlab
d1 == d2
```

**Rationale**

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

**Last Change**: V2.0
### 7.2.7. jc_0491: Reuse of variables within a single Stateflow scope

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0491: Reuse of variables within a single Stateflow scope</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Description**

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

**Correct**

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

**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

<table>
<thead>
<tr>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ i=1; [i&lt;10] }</td>
<td>{ i=1; [i&lt;10] }</td>
</tr>
<tr>
<td>{ i++; }</td>
<td>{ i++; }</td>
</tr>
<tr>
<td>a=b+c;</td>
<td>i=a+b;</td>
</tr>
</tbody>
</table>
### Rationale
- ✔ Readability
- ✔ Workflow
- ✔ Code Generation
- □ Verification and Validation
- □ Simulation

### Last Change
V2.0

#### 7.2.8. jc_0541: Use of tunable parameters in Stateflow

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jc_0541: Use of tunable parameters in Stateflow</th>
</tr>
</thead>
<tbody>
<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>Tunable parameters should be included in a Chart as inputs from the Simulink model.</td>
</tr>
</tbody>
</table>

#### Correct

![Diagram of Correct Usage](image)

```
Correct

param1
Constant

Chart

<table>
<thead>
<tr>
<th>Name</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>param1</td>
<td>Input</td>
</tr>
</tbody>
</table>
```
The following rules apply to logic in Stateflow:

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

**Incorrect**

```
xyTrac

xForce = WheelTqTot * ml.cos(wheelAng);
yForce = WheelTqTot * ml.sin(wheelAng);
```

**Rationale**

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

**Last Change** V2.0
### 7.2.10. jm_0011: Pointers in Stateflow

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jm_0011: Pointers in Stateflow</th>
</tr>
</thead>
<tbody>
<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 Stateflow diagram, pointers to custom code variables are not allowed.</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>
</tbody>
</table>

### 7.3. Events

#### 7.3.1. db_0126: Scope of events

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0126: Scope of events</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
| Description        | The following rules apply to events in Stateflow:  
|                    |  • All events of a Chart must be defined on the chart level or lower.  
|                    |  • There is no event on the machine level (i.e. there is no interaction with local events between different charts). |
| Rationale          | Readability: ✔️ Verification and Validation: ✔️ Workflow: ✔️ Code Generation: ☐ Simulation: ☐ |
| Last Change        | V2.0                      |

#### 7.3.2. jm_0012: Event broadcasts

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>jm_0012: Event broadcasts</th>
</tr>
</thead>
<tbody>
<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_0126: Scope of events</td>
</tr>
</tbody>
</table>
| 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  
|
7.4. Statechart Patterns

7.4.1. db_0150: State machine patterns for conditions

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0150: State machine patterns for conditions</th>
</tr>
</thead>
<tbody>
<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:</td>
</tr>
</tbody>
</table>

Example using the send syntax:

Example using qualified event names:
### Equivalent Functionality State Machine Pattern

<table>
<thead>
<tr>
<th>ONE CONDITION:</th>
<th>State Machine Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(condition)</em></td>
<td><img src="image1.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UP TO THREE CONDITIONS, SHORT FORM:</th>
<th>State Machine Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(The use of different logical operators in this form is not allowed, use sub conditions instead)</em></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
<tr>
<td><em>(condition1 &amp;&amp; condition2)</em></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
<tr>
<td>*(condition1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TWO OR MORE CONDITIONS, MULTILINE FORM:</th>
<th>State Machine Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(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)</em></td>
<td><img src="image5.png" alt="Diagram" /></td>
</tr>
<tr>
<td><em>(condition1 ... &amp;&amp; condition2 ... &amp;&amp; condition3)</em></td>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
<tr>
<td>*(condition1 ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Rationale</strong></th>
<th><strong>State Machine Pattern</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ Readability</td>
<td>☑ Verification and Validation</td>
</tr>
<tr>
<td>☑ Workflow</td>
<td>☐ Code Generation</td>
</tr>
<tr>
<td>☐ Simulation</td>
<td></td>
</tr>
</tbody>
</table>

| Last Change | V2.0 |

#### 7.4.2. db_0151: State machine patterns for transition actions

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0151: State machine patterns for transition actions</th>
</tr>
</thead>
</table>

**Description:**

The following patterns are used for transition actions within Stateflow state machines:
### 7.5. 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.

#### 7.5.1. db_0148: Flowchart patterns for conditions

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0148: Flowchart patterns for conditions</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Prerequisites**

The following patterns are used for conditions within Stateflow Flowcharts:

<table>
<thead>
<tr>
<th>Equivalent Functionality</th>
<th>Flowchart Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE CONDITION:</td>
<td><img src="image" alt="Flowchart Pattern" /></td>
</tr>
<tr>
<td>[condition]</td>
<td><img src="image" alt="Flowchart Pattern" /></td>
</tr>
<tr>
<td>action;</td>
<td><img src="image" alt="Flowchart Pattern" /></td>
</tr>
<tr>
<td>action1; action2; action3;</td>
<td><img src="image" alt="Flowchart Pattern" /></td>
</tr>
</tbody>
</table>

**Rationale**

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

**Last Change** V1.00
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)

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)

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))
The following patterns are used for condition actions within Stateflow Flowcharts:

### Equivalent Functionality

**ONE CONDITION ACTION:**

```plaintext
action;
```

**TWO OR MORE CONDITION ACTIONS, MULTILINE FORM:**

(Two or more condition actions in one line are not allowed.)

```plaintext
action1; ...
action2; ...
action3; ...
```

### Flowchart Pattern

```
{ action; }
```

```
/* comment */
{ action; }
```

```
{ action1; action2; action3; }
```
7.5.3. db_0134: Flowchart patterns for If constructs

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0134: Flowchart patterns for If constructs</th>
</tr>
</thead>
<tbody>
<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</td>
</tr>
<tr>
<td></td>
<td>db_0149: Flowchart patterns for condition actions</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Equivalent Functionality</th>
<th>Flowchart Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF THEN</td>
<td></td>
</tr>
</tbody>
</table>
| if (condition)
  { action; }           |                   |

CONDITION ACTIONS, WHICH ARE VISUALLY SEPARATED:
(This form can be mixed up with the patterns listed above.)
action1a;
action1b;
action2;
action3;

Rationale
- ☑ Readability
- ☑ Workflow
- ☑ Verification and Validation
- ☐ Code Generation
- ☐ Simulation

Last Change V1.00
IF THEN ELSE
if (condition) {
    action1;
} else {
    action2;
}

IF THEN ELSE IF
if (condition1) {
    action1;
} else if (condition2) {
    action2;
} else if (condition3) {
    action3;
} else {
    action4;
}

Cascade of IF THEN
if (condition1) {
    action1;
    if (condition2) {
        action2;
        if (condition3) {
            action3;
        }
    }
}

Rationale

☑ Readability
☑ Workflow
☐ Simulation

☑ Verification and Validation
☐ Code Generation

Last Change V1.00
7.5.4. db_0159: Flowchart patterns for case constructs

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0159: Flowchart patterns for case constructs</th>
</tr>
</thead>
<tbody>
<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</td>
</tr>
<tr>
<td></td>
<td>db_0149: Flowchart patterns for condition actions</td>
</tr>
</tbody>
</table>

Description

The following patterns must be used for case constructs within Stateflow Flowcharts:

Equivalent Functionality

```java
CASE with exclusive selection
selection = ...
switch (selection) {
  case 1:
    action1;
    break;
  case 2:
    action2;
    break;
  case 3:
    action3;
    break;
  default:
    action4;
}
```

Flowchart Pattern

```
{ selection = ...; }
```

```
{ selection == 1 } { action1; }
```

```
{ selection == 2 } { action2; }
```

```
{ selection == 3 } { action3; }
```

```
{ action4 }
```

```
```
CASE with exclusive conditions

c1 = condition1;
c2 = condition2;
c3 = condition3;
if (c1 && !c2 && !c3) {
    action1;
} elseif (c1 && c2 && !c3) {
    action2;
} elseif (c1 && !c2 && c3) {
    action3;
} else {
    action4;
}

Rationale
- Readability
- Workflow
- Verification and Validation
- Simulation

Last Change V1.00

7.5.5. db_0135: Flowchart patterns for loop constructs

<table>
<thead>
<tr>
<th>ID: Title</th>
<th>db_0135: Flowchart patterns for loop constructs</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Description

The following patterns must be used to create Loops within Stateflow Flowcharts:

### Equivalent Functionality

FOR LOOP

(index=0;index<number_of_loops;index++)
{
    action;
}
WHILE LOOP
while (condition) {
    action;
}

DO WHILE LOOP
do {
    action;
} while (condition);
Appendix A: Recommendations for Automation Tools

These recommendations are intended for any company that develops tools that automate checking of the Style Guidelines. These guidelines were developed by the MathWorks Automotive Advisory Board (MAAB), and it is expected that tool vendors will create tools that check models developed by MathWorks tools 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 first.

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.
Appendix B: Guideline Writing

The most important things to address when writing a new guideline are that each guideline should be:

- understandable and unambiguous
- easy to find
- minimal

Guidelines with these characteristics are easier to understand and use.

By "understandable and unambiguous" we mean that a guideline's description should be precise, clearly worded, concise and should define an evaluate able property 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" we mean that a 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.)

Lastly, by "minimal" we mean that a 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.
10. Appendix C: Flowchart Reference

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

<table>
<thead>
<tr>
<th>Straight Line Flow Chart Pattern</th>
<th>Curved Line Flow Chart Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF THEN</strong></td>
<td>![Straight Line Pattern]</td>
</tr>
<tr>
<td>![Condition]</td>
<td>![Curved Line Pattern]</td>
</tr>
<tr>
<td>{ action; }</td>
<td>{ condition }</td>
</tr>
<tr>
<td>{ action1; }</td>
<td>{ action2; }</td>
</tr>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

| **IF THEN ELSE**                 | ![Straight Line Pattern]       |
| ![Condition]                     | ![Curved Line Pattern]         |
| { action2; }                     | { condition }                  |
| { action1; }                     | { action1; }                   |
| ![Diagram](image3.png)          | ![Diagram](image4.png)        |

| **IF THEN ELSE IF**              | ![Straight Line Pattern]       |
| ![Condition]                     | ![Curved Line Pattern]         |
| { action2; }                     | { condition }                  |
| { action1; }                     | { action1; }                   |
| ![Diagram](image5.png)          | ![Diagram](image6.png)        |
The following patterns are used for case constructs within Stateflow Flowcharts:

<table>
<thead>
<tr>
<th>Straight Line Flow Chart Pattern</th>
<th>Curved Line Flow Chart Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE with exclusive selection</td>
<td></td>
</tr>
</tbody>
</table>
CASE with exclusive conditions
The following patterns are used for For Loops within Stateflow Flowcharts:

<table>
<thead>
<tr>
<th>Straight Line Flow Chart Pattern</th>
<th>Curved Line Flow Chart Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR LOOP</td>
<td></td>
</tr>
</tbody>
</table>

For Loop Diagrams:
The following patterns are alternately used for If-then-else-if constructs within Stateflow Flowcharts:
### 11. Glossary

**Actions**

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

\[
\text{Condition action} \quad \text{Transition action}
\]

![Diagram](image)

States can have entry, during, exit, and, on *event_name* actions. For example,

```
Power_on/  
entry: action1();  
during: action2();  
ext: action3();  
on switch_off: action4();
```

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**

You sometimes 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.

```plaintext
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
  }
}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Button Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connective junction</td>
<td>![icon]</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Button Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default transition</td>
<td>![Icon]</td>
<td>Use a Default transition to indicate, when entering this level in the hierarchy, which state becomes active by default.</td>
</tr>
</tbody>
</table>

**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 flow 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.
**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 non-graphical 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Button Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td><img src="image" alt="State Icon" /></td>
<td>Use a state to depict a mode of the system.</td>
</tr>
</tbody>
</table>
**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.
<table>
<thead>
<tr>
<th>Block Name</th>
<th>Condition Under Which Block Will Be Virtual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Selector</td>
<td>Virtual if input bus is virtual</td>
</tr>
<tr>
<td>Demux</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Enable</td>
<td>Virtual unless connected directly to an Outport block</td>
</tr>
<tr>
<td>From</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Goto</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Goto Tag Visibility</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Ground</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Inport</td>
<td>Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.</td>
</tr>
<tr>
<td>Mux</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Outport</td>
<td>Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.</td>
</tr>
<tr>
<td>Selector</td>
<td>Virtual except in matrix mode</td>
</tr>
<tr>
<td>Signal Specification</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Subsystem</td>
<td>Virtual unless the block is conditionally executed and/or the block's Treat as Atomic Unit option is selected</td>
</tr>
<tr>
<td>Terminator</td>
<td>Always virtual</td>
</tr>
<tr>
<td>Trigger</td>
<td>Virtual if the Outport port is not present</td>
</tr>
</tbody>
</table>

**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.