A Simulink Application Style Guide

The attached document **does not contain** ITAR-controlled data within the definition of the International Traffic in Arms Regulations and has been reviewed and approved for release to foreign persons. The document has not been approved for unlimited public release.

3.2.15.18.20.1 Style Guide

1 INTRODUCTION

This Simulink™ Application Style Guide is an important basis for program success and teamwork - not only in-house, but also when cooperating with subcontractors. Respecting the Style Guide's guidelines is one key prerequisite to achieving

- system integration without problems.
- clean interfaces.
- uniform appearance of models, code and documentation.
- reusable models.
- readable models.
- hassle-free exchange of models
- avoidance of legacies.
- a clean process
- professional documentation.
- understandable presentations.
- fast software changes.
- cooperation with subcontractors.
- easy transitioning of technology to programs

The style guide is an involving document. The intent is to automate the guideline conformity eventually. A command could be run and it would list the aspects of the model that do not conform to the guidelines. This would make standardization easier.

This document is based on the Mathworks Automotive Industry Style Guide, available from the Mathworks web site. Matlab and Simulink are Trademarks of The Mathworks, Inc.

1.1 Guideline template

All guidelines are described using the following template:

ID: Title	**xxxx: Title of the guideline (unique, short)**
Priority	One of mandatory / strongly recommended / recommended
Scope	Individual program or multiple programs or all programs
Automation	One of check / correction / possible / none
Prerequisites	Links to guidelines, which are prerequisite to this guideline (ID+title)
Description	Description of the guideline (text, images)
Benefit	Benefit of respecting the guideline
Penalty	Penalty of breaking the guideline
Author	Name of the first author or name of the source document
Last Change	Date of last change, Author of last change

Note: The elements of this template are considered to be the minimum required items that must be present for proper understanding and exchange of guidelines.

1.1.1 Guideline ID:

The guideline ID is a four digit number

Each guideline author starts with ID's counting up from zero. There should be no gap in ID's numbers.

Once a new guideline has an ID, it is fixed to this guideline.

The ID is used for references to guidelines and checker tools

1.1.2 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 hot link 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 fairly stable.

1.1.3 Priority

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

mandatory	strongly recommended	recommended
DEFINITION		
guidelines that all programs agree to that are absolutely essential guidelines that all programs conform to 100%	guidelines that are agreed upon to be a good practice, but legacy models preclude a program from conforming 100% to this guideline	guidelines that are recommended to improve the appearance, but are not critical to running the model these guidelines will traditionally appear in project specific guidelines
CONSEQUENCES If the guideline is violated ...		
essential things are missing the model may not work properly	the quality and the appearance will deteriorate there is a greater risk of consequences in maintainability, portability, and reusability	the appearance will not conform with other projects
WAIVER POLICY If you decide to violate this guideline...		
You must document your reasons, etc., etc This is only an example of the waiver policy.		

1.1.4 Scope:

The programs affected by the particular guideline should be listed.

1.1.5 Automation:

One of "**correction**", "**check**", "**possible**" or "**none**".

Correction: check and (semi-) automatic correction is possible and is already available

Check: check is possible and there is a checker for this guideline available. The checker-name is the ID of the guideline.

Possible: check is possible but has not been implemented yet.

None: It's not possible to automatically check compliance to the guideline.

1.1.6 Prerequisites:

This is a field 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.

1.1.7 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, 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.

1.1.8 Benefit:

The "Benefit"-field contains a description of the advantages of respecting the guideline.

If needed, images and tables can be added.

Note: This field is an important basis for deciding whether or not to adopt or follow the guideline. The content should be as specific as possible (not: "quality is improved")

1.1.9 Penalty:

The "Penalty"-field contains a description of the disadvantages of breaking the guideline.

If needed, images and tables can be added.

Note: This field is an important basis for deciding whether or not to adopt or follow the guideline. The content should be as specific as possible (not: "quality is reduced")

1.1.10 Author:

The "Author"-field contains the original author or the source document of the guideline.

The format is "firstname lastname".

Note: This field serves as contact information for questions and change requests regarding the guidelines and also documents the ownership or responsibility for maintaining the guideline

1.1.11 Last change:

The "Last change"-field contains the date and the author of the last change.

The format is dd.mm.yyyy, firstname lastname.

1.2 Guideline writing

When writing a new guideline, there are several things to keep in mind. This section describes the most important ones of them and gives more detailed instructions on how to fill in the template. Above all, every guideline should be

understandable and unambiguous

easy to find

motivating

minimal

With "understandable and unambiguous", we mean that the description of the guideline should describe precisely the state of guideline conformance (something that is in a certain way, has a certain property or something that must be done). So words like "should", "could", "might", ... must be avoided. Note: Since most tips in this section are not hard and fast rules, we will still use them here. This is not only important for understanding by developers but also in terms of specifications for automated checks. If it is not obvious, the description should also describe how the conformant state can be reached (e.g. by selecting some option, clicking a certain button, ...). Counterexamples and/or screenshots/illustrations may be provided as necessary. The description itself should be short and succinct, but additional notes may be provided in an extra paragraph. An important thing to keep in mind is the avoidance of exceptions, since they blur the guideline and make it harder to apply. One exception to a rule might be ok, but if there are many, you might be trying to cover too many things in one place (see "minimal" below). In this case, it is better to break the guideline down into smaller ones. Also, some exceptions can easily be expressed as separate guidelines that restrict the original one (see Guideline_Interaction_Semantics below) and may also have a smaller scope.

The "easy to find" requirement affects the title of the guideline and its placement in the table of contents. Just like a heading in a document describes the subject and not the content of the following chapter, the (permanent) title of a guideline should not be a summary of the description of the guideline (which may change) but indicate its area of application. For example, a guideline may concerns the area "allowed characters in names"(=title) and then specify, that "no special characters may be used"(=description). Such a title also gives a hint about where (in which chapter) to place (or look for) the guideline. If the guideline has prerequisites, they should be listed above/before the dependent guideline (this may not always be possible if the prerequisite is in a different section). Remember: the guideline-ID has no semantics (e.g. order of guidelines) and prerequisites must be of same or wider scope. Of course, circular prerequisites must be avoided.

By "motivating" we mean that it should be "a pleasure to follow the guideline". The benefits and penalties play an important part here. ***Many developers see guidelines as a burden that causes extra work and limits their creative freedom. It is the purpose of the benefits and penalties to convince them otherwise. Therefore, those two fields should be filled in with care. Generic statements like "improves quality" or "prevents errors" must be avoided. Instead, describe the aspect of quality affected (e.g. readability, maintainability, reliability,...) or the exact error (custom tool malfunction, runtime error, type mismatch, ...).***

Lastly, with "minimal" we would like to point out, that a guideline should address only one thing at a time. In other words, guidelines should be atomic. So instead of writing a super-guideline that tries to prevent errors and make things look good at the same time, make it two and maybe connect them with a prerequisite (we might care about looks only after everything works - but then again if nobody understands it, it might never work...). As another example, don't specify fonts, colors and layout in one guideline. It is very hard to get many people (at a wide scope) to agree on many things at the same time. In fact, there is a danger that the compromise reached in the end is so soft and abstract; it is not good for anything. Either way, progress comes easier in small steps. Also, small guidelines are much more resilient to change than a big guideline which eventually will end up containing many exceptions which blur its original intent and make it hard to understand (see above).

1.3 Document usage

The following paragraphs give some directions on how to use this document for reference and for compiling a project-specific guideline document

1.3.1 Guideline Organization

The document is structured as follows: The top level consists of a few introductory and explanatory chapters (like this one) a global chapter for custom tool or domain independent guidelines (e.g. Operating System / Files, MATLAB®, Simulink®, Stateflow®, Handcode...). and one chapter for every relevant custom tool or domain. The hierarchy of sections within each such chapter follows the hierarchy suggested by the custom tool or domain, e.g. directories and files for the operating system, functions and libraries for programming languages / environments, models, submodels, blocks and libraries for modeling tools, etc.

1.3.2 Guideline Interaction Semantics

The guideline document contains many guidelines that are related, similar or even (seemingly) contradicting each other. Following is a description of how the interaction of multiple guidelines works:

Guidelines in the Global chapter are valid for all the tools and domains in the other chapters.

Guidelines in a specific chapter specialize (replace, supersede, overwrite, overload) guidelines from the Global chapter.

Guidelines from a specific subsection in any chapter specialize (replace, supersede, overwrite, overload) guidelines from a higher-level section.

Prerequisite guidelines reinforce the guideline by which they are referenced. Both the prerequisite **and** the referring guideline must be complied with (logical conjunction).

Note on Specialization: the notion of guideline specialization is similar to the concept of overloading in object-oriented programming. While a specific guideline does replace the higher-level guideline (i.e. the higher level guideline is no longer valid), it typically does so by means of specialization, i.e. it will comply with it but impose additional constraints. This may be expressly documented by describing the constraints only and including the higher-level guideline as a prerequisite (again, this is similar to OO where the overloading function often first calls its overloaded counterpart). In rare cases however, a specific guideline may also contradict its higher-level counterpart by allowing specific exceptions. Such exceptions should not be included in the higher-level guideline because doing so will move the description away from its specific point of application!
Note on Prerequisites: Since a prerequisite is used to reinforce a guideline, it may not be of lesser scope than the referring guideline. From a different perspective, the referring guideline extends the prerequisites (see specialization above). A specific guideline may extend a generic one but the other direction does not make sense.

1.3.3 Guideline Lookup

When looking for a guideline, one should start at the most specific subsection of the affected tool or domain and then ascend the hierarchy up until an appropriate guideline is found. If no appropriate guideline can be found in the chapter on the specific tool / domain, check the Global chapter - again starting at the most specific subsection. When a guideline has been found, follow the links in the template to find all the prerequisites guidelines that must also be followed (see interaction semantics above).

1.3.4 Document Scope

This document covers the details of a Matlab / Simulink / Stateflow model. Higher level procedures on how to design a model, scope a model, generate code from a model, target RTCF components, etc. are covered in the in other documentation

2 REFERENCES

2.1 Government Documents

TBD

2.2 Non-Government Documents

2.2.1 Documents

TBD

2.2.2 Mathworks Documents

TBD

2.2.3 Other Documents

TBD

2.3 Precedence

TBD

3 GLOBAL

3.1 General

3.1.1 0001: Indexing

ID: Title	**0001: Indexing**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Indexing [1, 2, 3,...] is used for MATLAB • Workspace variables and structures • Local variables of m-functions • Global variables Simulink • Signal vectors and matrices • Tunable parameter vectors and matrices • m-coded s-function input and output signal vectors and matrices • m-coded s-function tunable parameter vectors and matrices • m-coded s-function local variables Indexing [0, 1, 2, ...] is used for Simulink • c-coded s-function input and output signal vectors and matrices • c-coded s-function input tunable parameters • c-coded s-function tunable parameter vectors and matrices • c-coded s-function local variables Stateflow

	All array indexing in Stateflow is done using 0-based indexing and C syntax; see “Array Notation” in the Stateflow documentation for more details C-Code Local variables and structures Global variables
Benefit	Respecting the guideline ensures ... • cooperation with subcontractors. • uniform appearance of models, code and documentation.
Penalty	Breaking the guideline ... may cause errors in functionality, e.g. auto-c-code generation. may cause misunderstandings with names. may cause confusing presentations.
Author	
Last Change	9-19-2008

3.2 Naming

3.2.1 0002: Project filenames

<table>
<tr><th>ID: Title</th><th>0002: Project filenames</th></tr>
<tr><td>Priority</td><td>mandatory</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>Project files are all files within the project directory structure.
A project filename conforms to the following constraints:
<table>
<tr><td>FORM</td><td>filename = name.extension
<b>name</b>: no leading digits, no blanks
<b>extension</b>: no blanks
<b>case:</b> Mixed case is OK, but using names that differ only in case is prohibited – some host OSes are case preserving but not case sensitive. Each project should select a more specific naming convention that adheres to this requirement; for example, CamelCase.ext or camel_case.ext.</td></tr>
<tr><td>UNIQUENESS</td><td>all filenames within the parent project directory</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td>ALLOWED CHARACTERS</td><td><b>name</b>:
lowercase a-z, uppercase A-Z, digits 0-9, underscore, and dash
<b>extension</b>:
lowercase a-z, uppercase A-Z</td></tr>
<tr><td>UNDERSCORES</td><td><b>name</b>:
underscores to separate parts, no sequence of more than one underscore, no leading or trailing underscores
<b>extension</b>:
no underscores</td></tr>
<tr><td>LENGTH</td><td><b>name</b>: 3 - 30
<b>extension</b>: 1 - 6</td></tr>
</table></td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...

• readability.
• platform independency.
• accessible project files.
• simple and fast editing.
• prevention of typing errors.
• prevention of ambiguities.
• custom tool integration.
• parsing of names.
• conformance to the C naming convention.</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...

• may cause misunderstandings with names.
• may cause problems when using different platforms (file systems).
• may cause problems with non-accessible files.
• may result in hard to find name mismatches.
• may result in subtle malfunctions.
• may cause custom tool errors, e.g. with code generators and compilers.
• may cause problems transferring data or files between custom tools.
• may cause errors in functionality.
• may cause data loss.</td></tr>
<tr><td>Author</td><td></td></tr>
<tr><td>Last Change</td><td>19.09.2008,</td></tr>
</table>

3.2.2 0003: Project directory names

ID: Title	**0003: Project directory names**
Priority	mandatory

<table>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>Project directories are all directories within the project directory structure.
A project directory name conforms to the following constraints:
<table>
<tr><td>FORM</td><td>directory name = name
name: no leading digits, no blanks
case: Mixed case is OK, but using names that differ only in case is prohibited – some host OSes are case preserving but not case sensitive. Each project should select a more specific naming convention that adheres to this requirement; for example, CamelCase or camel_case.</td></tr>
<tr><td>UNIQUENESS</td><td>all directory names within the parent project directory</td></tr>
<tr><td>ALLOWED CHARACTERS</td><td>**name**:
lowercase a-z, uppercase A-Z, digits 0-9, underscore, and dash</td></tr>
<tr><td>UNDERSCORES</td><td>**name**: underscores to separate parts, no sequence of more than one underscore, no leading or trailing underscores</td></tr>
<tr><td>LENGTH</td><td>**name**: 3 - 30</td></tr>
</table></td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...
readability.
platform independency.
accessible project files.
simple and fast editing.
prevention of typing errors.
prevention of ambiguities.
custom tool integration.
parsing of names.
conformance to the C naming convention.</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...
may cause misunderstandings with names.</td></tr>
</table>

	may cause problems when using different platforms (file systems). may cause problems with non-accessible files. may result in hard to find name mismatches. may result in subtle malfunctions. may cause custom tool errors, e.g. with code generators and compilers. may cause problems transferring data or files between custom tools. may cause errors in functionality. may cause data loss.
Author	
Last Change	19.09.2008,

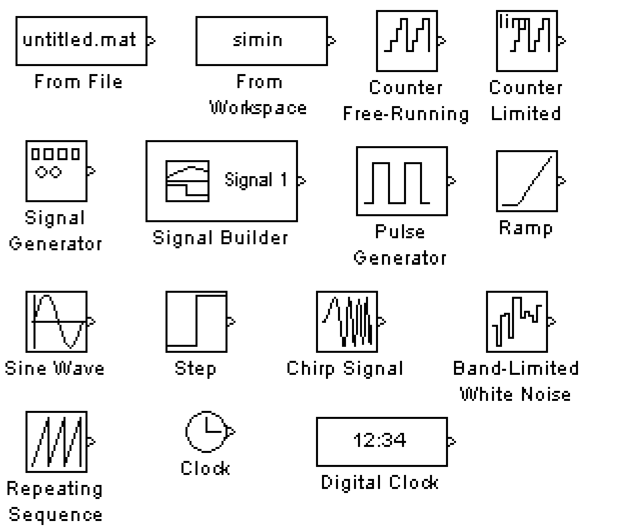
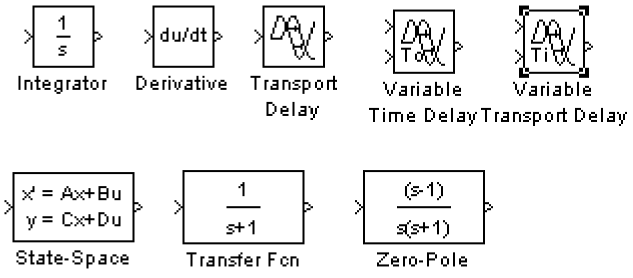
4 SIMULINK

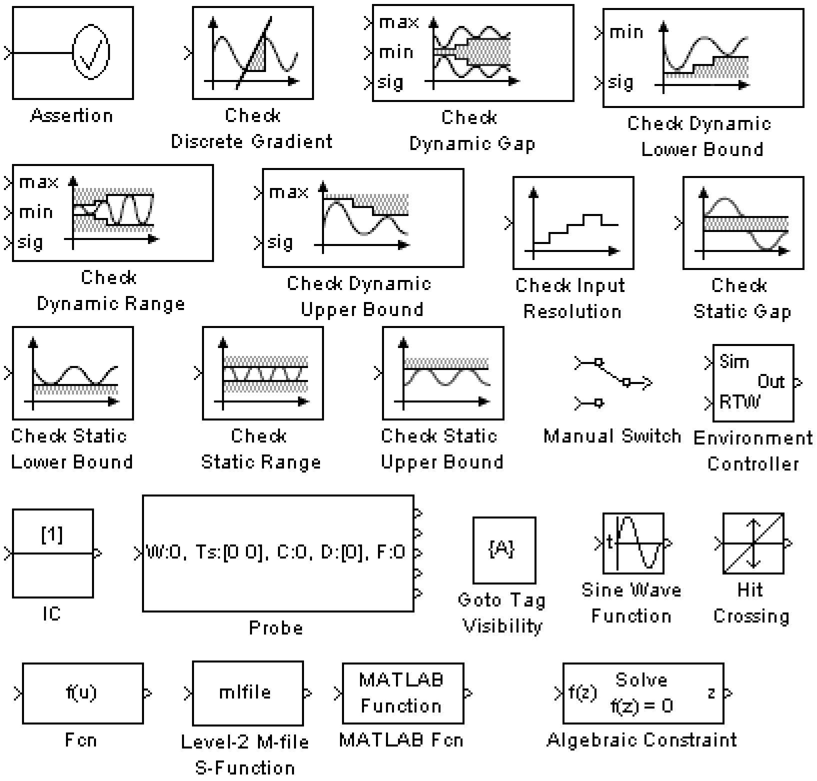
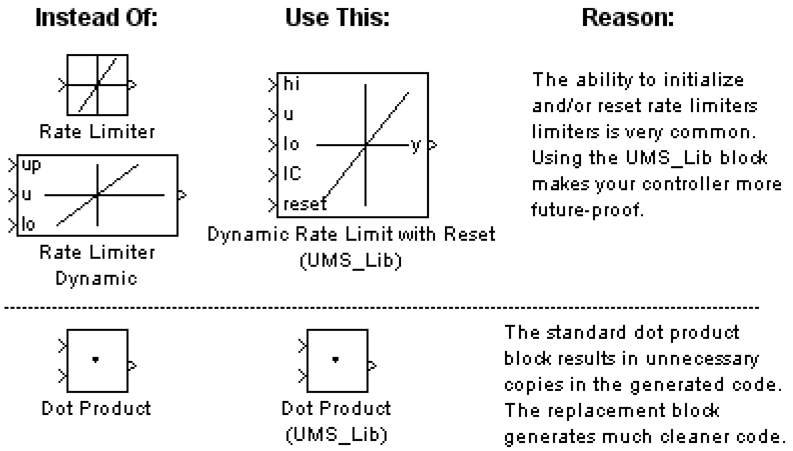
4.1 General

4.1.1 0004: Use of Simulink

ID: Title	**0004: Use of Simulink**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Simulink is used to ... • model controller difference equations. • model controller test environments. • serve as the framework for holding Stateflow diagrams • simulate systems or subsystems. • generate auto-c-code of systems or subsystems for embedded (production) code and/or real- and non-real-time simulation outside of the Matlab / Simulink environment. Simulink is not used to ... • model significant control flow and logic.
Benefit	Respecting the guideline ensures ... efficient models and auto-c-code. readability. eliminates complex logic within Simulink.
Penalty	Breaking the guideline ... • may cause inefficient models or auto-c-code.
Author	
Last Change	19.09.2008,

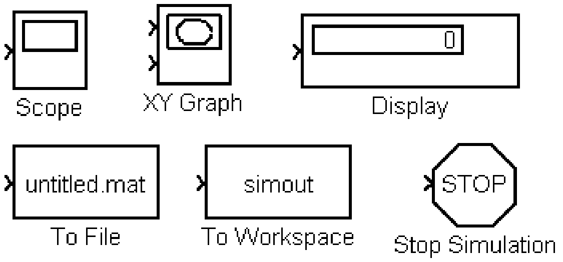
4.1.2 0005: Prohibited Simulink standard blocks inside controllers and/or “production code”

<table>
<tr><td><b>ID: Title</b></td><td><b>0005: Prohibited Simulink standard blocks inside controllers and/or production code</b></td></tr>
<tr><td>Priority</td><td>Mandatory</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>Possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td><b>Controller / production code models must only be designed from discrete blocks.</b>
<table>
<tr><td colspan="2"><b>Sources that are not allowed:</b></td></tr>
<tr><td>Signal Generator
Step
Ramp
Sine Wave
Repeating Sequence
Pulse Generator
Chirp Signal
Clock
Digital Clock
From File
From Workspace
Band-Limited White Noise
Counter Free-Running
Counter Limited
Signal Builder</td><td>
</td></tr>
<tr><td colspan="2"><b>Continuous blocks are not allowed:</b></td></tr>
<tr><td>Integrator
Derivative
Transport Delay
Variable Time Delay
Variable Transport Delay
State-Space
Transfer Fcn
Zero-Pole</td><td>
</td></tr>
<tr><td colspan="2"><b>Other blocks, which are not allowed:</b></td></tr>
</table></td></tr>
</table>

	Algebraic Constraint Manual Switch Hit Crossing Polynomial MATLAB Fcn Goto Tag Visibility Probe Assertion Check Discrete Grad. Check Dynamic Gap Check Dynamic Lower Bound Check Dynamic Rng Check Dynamic Upper Bound Check Input Resolution Check Static Gap Check Static Lower Bound Check Static Range Check Static Upper Bound Environment Controller IC Sine Wave Function Fcn Level-2 M-file S-fcn	 Assertion Check Discrete Gradient max min sig Check Dynamic Gap min sig Check Dynamic Lower Bound max min sig Check Dynamic Range max sig Check Dynamic Upper Bound Check Input Resolution Check Static Gap Check Static Lower Bound Check Static Range Check Static Upper Bound Manual Switch Sim Out RTW Environment Controller [1] IC W:0, Ts:[0 0], C:0, D:[0], F:0 Probe {A} Goto Tag Visibility t Sine Wave Function Hit Crossing f(u) Fcn mlfile Level-2 M-file S-Function MATLAB Function MATLAB Fcn f(z) Solve f(z) = 0 z Algebraic Constraint
Description	**Prohibited blocks for which approved replacements exist in PROJECT-developed libraries:**	
	These blocks in the standard library shall not be used; the replacement blocks shown are to be used instead.	 Instead Of: Use This: Reason: Rate Limiter up u lo Rate Limiter Dynamic hi u lo IC reset y Dynamic Rate Limit with Reset (UMS_Lib) Dot Product Dot Product (UMS_Lib) The ability to initialize and/or reset rate limiters limiters is very common. Using the UMS_Lib block makes your controller more future-proof. The standard dot product block results in unnecessary copies in the generated code. The replacement block generates much cleaner code.

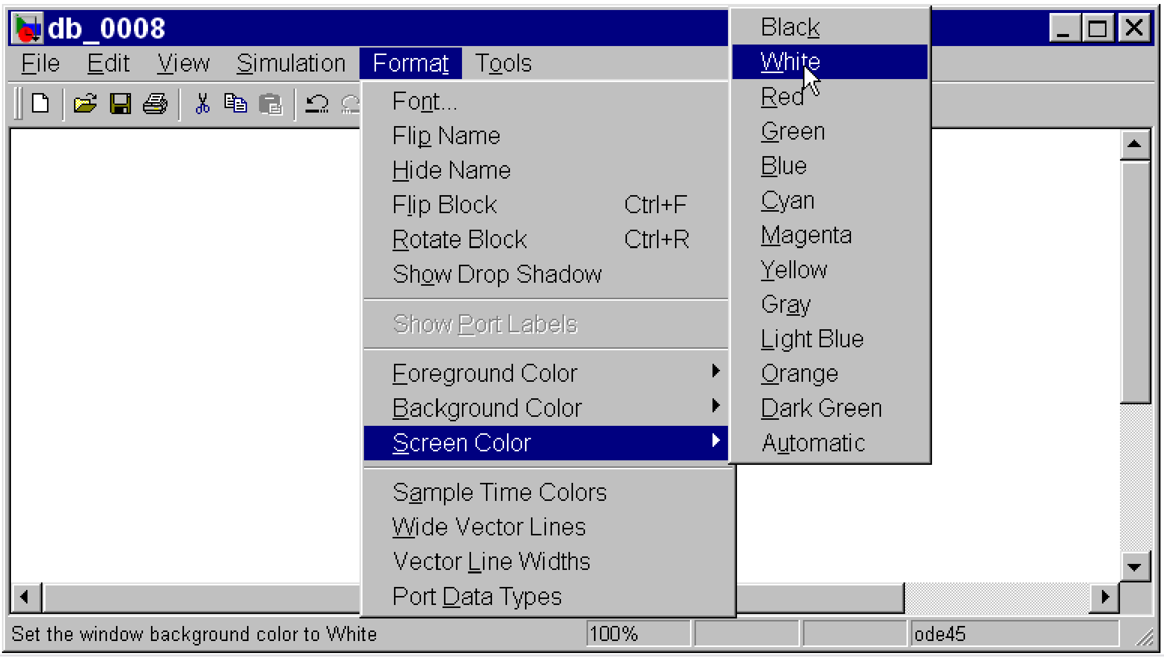
Benefit	Respecting the guideline ensures ... A controller that more closely represents the way it will be implemented. A controller that is not dependent on absolute time and does not reference non-finite numbers. A controller that results in efficient generated code.
Penalty	Breaking the guideline ... Code may not generate correctly or efficiently. Code may not be usable under hard-real-time operating systems or from RTCF. Code may not be testable.
Author	
Last Change	19.09.2008,

4.1.3 0006: Prohibited Simulink Sink

ID: Title	**0006: Prohibited Simulink Sink**
Priority	Mandatory
Scope	PROJECT
Automation	Possible
Prerequisites	
Description	**Controller models must only be designed from discrete blocks.** **Sinks are not allowed:** Scope XY Graph Display To File To Workspace Stop Simulation  Scope XY Graph 0 Display untitled.mat To File simout To Workspace STOP Stop Simulation It is very useful to inspect running simulations, save data from running sims, etc.; however, these blocks shall only be used outside the actual production controller. Signal logging, floating scopes, etc. make this straightforward. An added benefit is that signal logging is compatible with reference models.
Benefit	Respecting the guideline ensures ... A controller that more closely represents the way it will be implemented. A controller that allows diagnostic data output in the Simulink environment even when used as a model referencing
Penalty	Breaking the guideline … will not be able to use triggered or enabled subsystems
Author	
Last Change	19.09.2008,

4.2 Colors

4.2.1 0007: Simulink window screen color

ID: Title	**0007: Simulink window screen color**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	- All Simulink windows should have a white screen color.  db_0008 File Edit View Simulation Format Tools Font... Flip Name Hide Name Flip Block Ctrl+F Rotate Block Ctrl+R Show Drop Shadow Show Port Labels Foreground Color Background Color Screen Color Sample Time Colors Wide Vector Lines Vector Line Widths Port Data Types Black White Red Green Blue Cyan Magenta Yellow Gray Light Blue Orange Dark Green Automatic 100% ode45 Set the window background color to White
Benefit	Respecting the guidelines ensures ... - uniform appearance of models.
Penalty	Breaking the guideline ... - cause an ugly and inconsistent appearance of models.
Author	
Last Change	19.09.2008,

4.2.2 0008: Block Color Coding

ID: Title	**0008: Block Color Coding**

Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Until there is a widely available and effective block diagram difference tool available to PROJECT model developers, the block background color should be used to keep track of changes in the code. When modifications are made to the block diagrams, the altered blocks should be changed to a unique color. This must be followed through to the uppermost level. To prevent blocks from becoming unreadable when the "Sample Time Colors" option is selected, do not use black, red, green, or yellow. When checking a revised model into the source control system, the check-in comment must indicate the color used for modifications. Colors should be chosen so that the black foreground annotations are easily readable.
Benefit	Respecting the guideline ensures ... • clear communication between authors. • More understandable model revision history.
Penalty	Breaking the guideline ... • may cause unnecessary confusion about the model revision history. •
Author	
Last Change	19.09.2008,

4.3 Project models and libraries

4.3.1 General

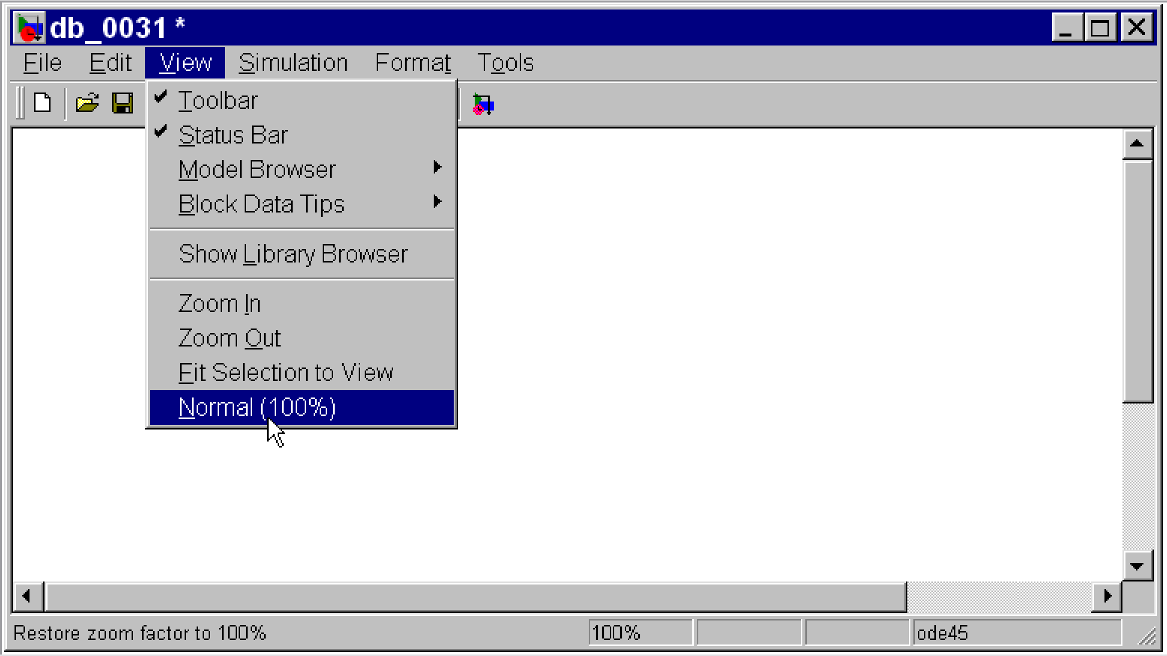
4.3.1.1 0009: Simulink font and font size

ID: Title	**0009: Simulink font and font size**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	

Description	All elements (blocks, signal labels,...) except text annotations within a model must have the same common font style and font size. Note: The selected font should be directly portable (e.g. Simulink/Stateflow default font) or convertible between platforms (e.g. Arial/Helvetica 12pt).
Benefit	Respecting the guideline ensures ... readable models. uniform appearance of models (screen and printer). reusable models. understandable presentations. ensures platform independency.
Penalty	Breaking the guideline ... • may cause unreadable models. • may cause a lot of redesign work. • may cause problems when switching to different platforms.
Author	
Last Change	18.08.2003

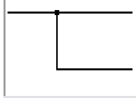
4.3.1.2 0010: Simulink window appearance

ID: Title	**0010: Simulink window appearance**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	All Simulink windows apply to the following rules: The zoom factor is 100 % (normal).

	 db_0031 * File Edit View Simulation Format Tools ✓ Toolbar ✓ Status Bar Model Browser Block Data Tips Show Library Browser Zoom In Zoom Out Fit Selection to View Normal (100%) 100% ode45Restore zoom factor to 100%
Benefit	Respecting the guideline ensures ... readable models. uniform appearance of models. reusable models. understandable presentations.
Penalty	Breaking the guideline ... may cause unreadable models. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

4.3.1.3 0011: Simulink signal appearance

ID: Title	**0011: Simulink signal appearance**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Signal lines ...

	should not cross each other, if possible. are drawn with right angles. are not drawn one upon the other. do not cross any basic-blocks, subsystems. If signal lines are branched the mark point of the branching point is situated on the branching point. 
Benefit	Respecting the guideline ensures ... readable models. uniform appearance of models. reusable models. understandable presentations. traceable models.
Penalty	Breaking the guideline ... • may cause unreadable models. • may cause a lot of redesign work. • may make it more difficult for others to understand the diagram.
Author	
Last Change	18.08.2003

4.3.1.4 0012: Ports in Simulink models

ID: Title	**0012: Ports in Simulink models**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	In a Simulink model, the ports comply with the following rules:

	• Inports should be placed on the left side, but they can be moved to prevent signal crossings. Also if inport signal is used in many places in a subsystem, either attach the import to a GoTo block, and then From blocks where the signal is needed, or use the “duplicate inport” feature. • Outports should be placed on the right side, but they can be moved to prevent signal crossings. The name of an inport or outport is not hidden. ("Format / Hide Name" is not allowed.)
Benefit	Respecting the guideline ensures ... • clear interfaces. • readable models. • uniform appearance of models. • may eliminate signal crossings.
Penalty	Breaking the guideline ... may cause unreadable models.
Author	
Last Change	19.09.2008,

4.3.1.5 0013: Port Names in Simulink models

ID: Title	**0013: Port Names in Simulink models**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	0010
Description	The name of an inport and outport block must match the appropriate signal or bus name. Exception: • When any combination of an inport block, an outport block, and a basic block have the same block name, the inport name should have the “_in” suffix and the outport name should have the "_out" suffix. Note: There is an option to cause the inport and outport to automatically take on the name of the signal connected to the port. Be sure the correct name is stated using this guideline. Typically, one port should be used for each signal (bus, scalar, vector, or matrix). Use a bus for clarity when the number of ports is very large.
Benefit	Respecting the guideline ensures ...

	information is available for report generation
Penalty	Breaking the guideline ... •will cause an error because a basic block can not have the same name as an inport block.
Author	
Last Change	01.12.2003

4.3.1.6 0014: Data flow in Simulink models

ID: Title	**0014: Data flow in Simulink models**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The data flow in a model is oriented from left to right.Exception: Feedback loops Sequential blocks are oriented from left to right. Exception: Feedback loops Parallel blocks are oriented from top to bottom.
Benefit	Respecting the guideline ensures ... • a clear model structure. • readable models.
Penalty	Breaking the guideline ... may cause problems with integration of imported subsystems. may cause a lot of redesign work.
Author	
Last Change	19.09.2008,

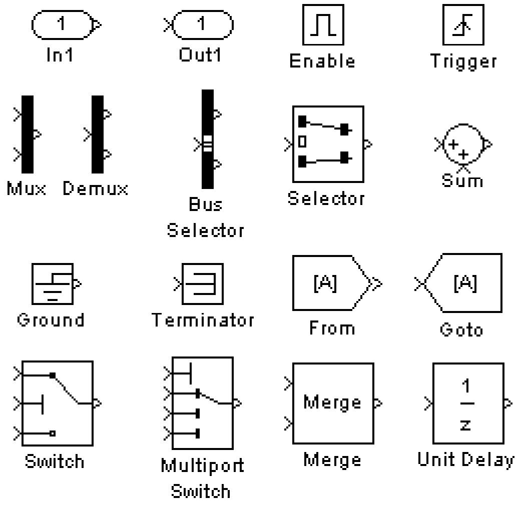
4.3.1.7 0015: Position of block names

ID: Title	**0015: Position of block names**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The name of all ...

	systems, subsystems, Stateflow-blocks s-function-blocks basic blocks are placed below the blocks.
Benefit	Respecting the guideline ensures ... readable models. working with other models.
Penalty	Breaking the guideline ... may cause a lot of redesign work.
Author	
Last Change	18.08.2003

4.3.1.8 0016: Similar block types on the model levels

ID: Title	**0016: Similar block types on the model levels**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Every level of a model must be designed with building blocks of the same type. (i.e. only subsystems or basic blocks). **Examples of Blocks which can be placed on every model level:**

<table>
<tr><td></td><td><table><tr><td>Inport
Outport
Enable (not on highest model level)
Trigger (not on highest model level)
Mux
Demux
Bus Selector
Selector
Sum
Ground
Terminator
From
Goto
Switch
Multiport Switch
Merge
Unit Delay</td><td>
</td></tr></table></td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...
a clear system structure.
readable models.</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...
• may cause problems with subsystem integration.
• may cause a lot of redesign work.
• may cause problems or errors with custom tools.</td></tr>
<tr><td>Author</td><td></td></tr>
<tr><td>Last Change</td><td>18.08.2003</td></tr>
</table>

4.3.1.9 0017: Unconnected signals

ID: Title	**0017: Unconnected signals**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A system has ... • no unconnected subsystem- or basic block inputs. • no unconnected subsystem- or basic block outputs.

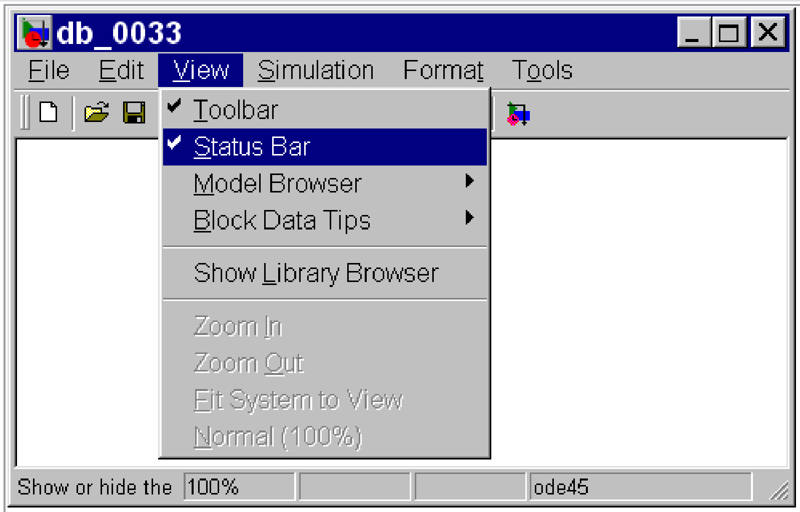
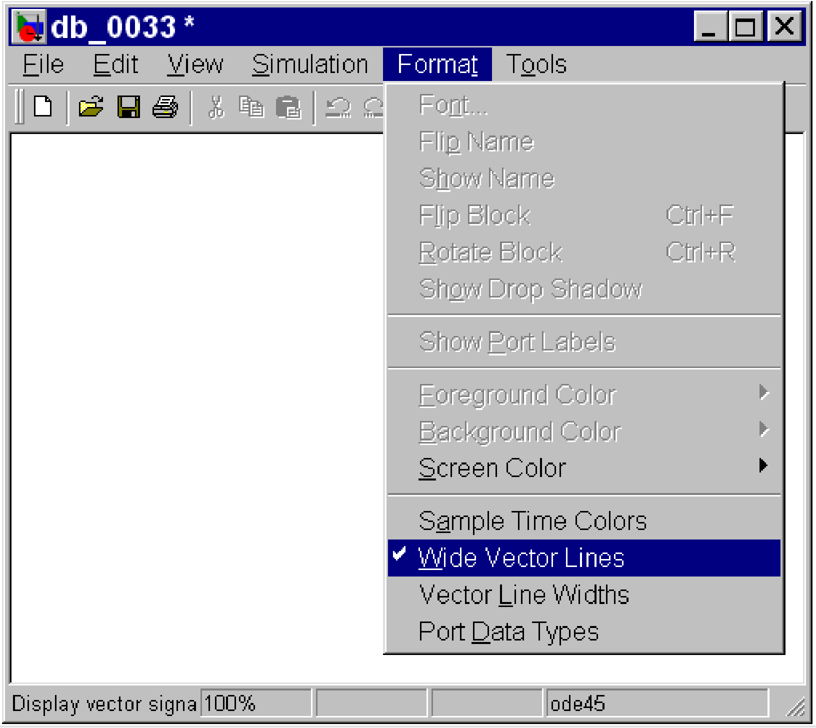
	• no unconnected signal lines.
Benefit	Respecting the guideline ensures ... • a clear system structure. • a clean subsystem interface. • exchangeable subsystem versions. • testable subsystems.
Penalty	Breaking the guideline ... may cause functional errors. may cause a lot of redesign work. may cause problems or errors with custom tools.
Author	
Last Change	18.08.2003

4.3.2 Naming

4.3.2.1 Options

4.3.2.1.1 0018: Simulink window options

ID: Title	**0018: Simulink window options**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The option "View / Toolbar" is activated. The option "View / Status Bar" is activated.

	 db_0033 File Edit View Simulation Format Tools ✔ Toolbar ✔ Status Bar Model Browser Block Data Tips Show Library Browser Zoom In Zoom Out Fit System to View Normal (100%) Show or hide the 100% ode45 The option "Format / Wide Vector Lines" is activated.  db_0033 * File Edit View Simulation Format Tools Font... Flip Name Show Name Flip Block Ctrl+F Rotate Block Ctrl+R Show Drop Shadow Show Port Labels Foreground Color Background Color Screen Color Sample Time Colors ✔ Wide Vector Lines Vector Line Widths Port Data Types Display vector signa 100% ode45
Benefit	Respecting the guideline ensures ... • uniform appearance of models. • the correct assignment of scalar signals and vectors. • readability
Penalty	Breaking the guideline ... • may cause problems with assignment of scalar signals and vectors.
Author	

Last Change	18.08.2003

4.3.2.2 General

4.3.2.2.1 0019: Model hierarchy

ID: Title	**0019: Model hierarchy**
Priority	Mandatory
Scope	PROJECT
Automation	none
Prerequisites	
Description	The model hierarchy corresponds to the logical structures of the system, or the controller.
Benefit	Respecting the guideline ensures ... • clear model structure. • understandable presentations.
Penalty	Breaking the guideline ... may cause problems with system integration. may cause a lot of redesign work. may cause confusing presentations.
Last Change	19.09.2008,

4.3.2.2.2 0019a: Model reference architecture

ID: Title	**0019a: Model reference architecture**
Priority	Mandatory
Scope	PROJECT
Automation	none
Prerequisites	
Description	Each top-level subsystem that corresponds to a Computer Software Unit (CSU) shall be implemented as a Model block. For example, the GNC for a typical system will consist of, as a minimum, an outer loop guidance CSU and an inner loop flight control CSU. This requirement ensures that the model structure is compatible with PROJECT techniques for generating CSUs from a model and “wrapping” those autogenerated CSUs as RTCF components. Another key benefit of using Model blocks is that the underlying model is a separate .mdl file, which facilitates the ability for several people to work on an overall

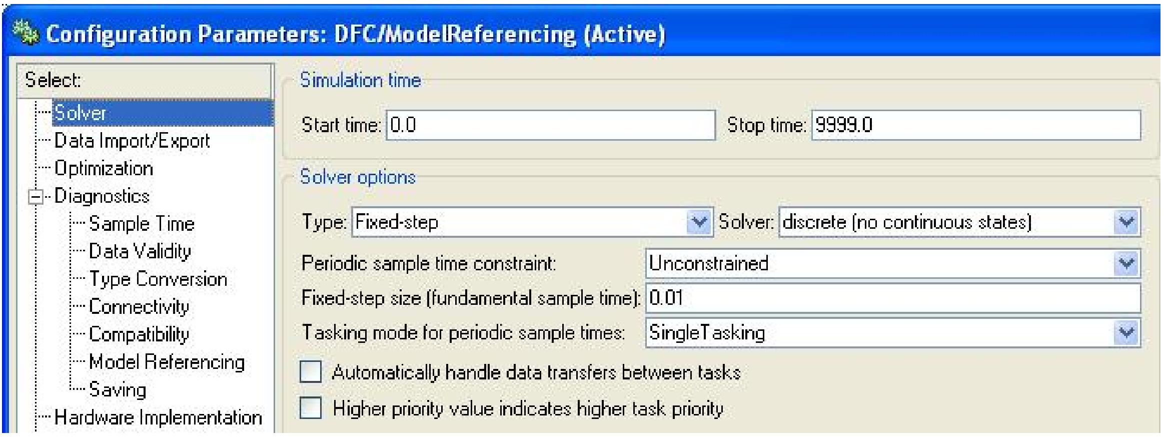
	system model at once (each one working on one model reference).
Benefit	Respecting the guideline ensures ... • Ability to distribute model development without time-consuming merge efforts • clear model structure. • understandable presentations. • Ability to generate code that is easily wrappable as RTCF components.
Penalty	Breaking the guideline ... • Will result in models that cannot be coded and turned into RTCF components without a great deal of reinventing-the-wheel. • may cause problems with system integration. • may cause a lot of redesign work. • may cause confusing presentations.
Author	
Last Change	

4.3.2.2.3 Model Reference Interfaces

ID: Title	**0019b: Model reference interfaces**
Priority	Mandatory
Scope	PROJECT
Automation	none
Prerequisites	
Description	Each model reference shall define and implement its interface using bus objects. Each bus object shall map to a logically related set of data and/or satisfy an external ICD. For example, GNC CSUs typically have IMU and navigation data as inputs. These data may be specified in an internal or vendor ICD, in which case the GNC interface would implement a bus object or objects that comply with the ICD. Note that input data that are being generated at different rates are necessarily different bus objects. Continuing with the navigator / IMU example, many navigators output IMU data (angular rates and linear accelerations) at one rate (e.g., 100 Hz in the case of a typical in use on several systems), but output navigation data (positions, velocities, attitudes) at as slower rate (50 Hz in the case of the typical IMU). This necessitates that, for proper cross-rate data flow, the navigation and IMU buses be separate. (note: add example diagram)
Benefit	Respecting the guideline ensures ... • clear model interfaces. • understandable presentations.

	• Ability to generate code that is easily wrappable as RTCF components.
Penalty	Breaking the guideline ... may cause problems with system integration. may cause a lot of redesign work. may cause confusing presentations.
Author	
Last Change	

4.3.2.2.4 Model Reference Sample Rate

ID: Title	**0019b: Model reference sample rate / tasking**
Priority	Mandatory
Scope	PROJECT
Automation	none
Prerequisites	
Description	Each model reference shall have only one sample rate or task. The tasking mode for each model reference shall be set to SingleTasking. Example:  Configuration Parameters: DFC/ModelReferencing (Active) Simulation time Start time: 0.0 Stop time: 9999.0 Solver options Type: Fixed-step Solver: discrete (no continuous states) Periodic sample time constraint: Unconstrained Fixed-step size (fundamental sample time): 0.01 Tasking mode for periodic sample times: SingleTasking □ Automatically handle data transfers between tasks □ Higher priority value indicates higher task priority Select: Solver Data Import/Export Optimization Diagnostics Sample Time Data Validity Type Conversion Connectivity Compatibility Model Referencing Saving Hardware Implementation This establishes the requirement that each model reference implements one CSU, which maps directly to one RTCF component. RTCF components are required to run at one rate and consist of one task (except for background tasks which are currently not mappable to Simulink constructs). Use of multiple tasks in a single model reference breaks the RTCF tasking model.
Benefit	Respecting the guideline ensures ... • clear model task semantics. • Ability to generate code that is easily wrappable as RTCF components.

Penalty	Breaking the guideline ... - Will results in models that cannot be turned into RTCF components. - Will cause a lot of redesign work.
Author	
Last Change	

4.4 System structure

4.4.1 General

4.4.1.1 0020: System controller

ID: Title	**0020: System controller**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A system should have no more than one dedicated control specification (i.e. a Stateflow diagram) that calculates the system state and sub-mode controllers be located in their respective subsystems and not as subcharts in a master chart. *Note*: Simple systems may not need a separate control specification. In complex systems, the control specification may only be broken down hierarchically but not split into parallel control subsystems on the same level.
Benefit	Respecting the guideline ensures ... clear system structure. understandable presentations.
Penalty	Breaking the guideline ... may cause problems with system integration. may cause a lot of redesign work. may cause confusing presentations.
Author	
Last Change	16.12.2003

4.4.2 Subsystems

4.4.3 General

4.4.3.1 0021: Use of subsystems

ID: Title	**0021: Use of subsystems**
Priority	Mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Subsystems shall be used in the following situations: • Enabled, triggered, and action subsystems – in these cases, Simulink requires that atomic subsystems be used • Library and / or masked blocks • Grouping of major related pieces of functionality within a model • Grouping of minor related pieces of functionality for block diagram readability purposes The use of subsystems promotes model modularity and hierarchy.
Benefit	Respecting the guideline ensures ... a clear system structure. exchangeable subsystems. testable subsystems.
Penalty	Breaking the guideline ... • may cause problems with subsystem integration. • may cause problems or errors with custom tools.
Author	
Last Change	19.09.2008

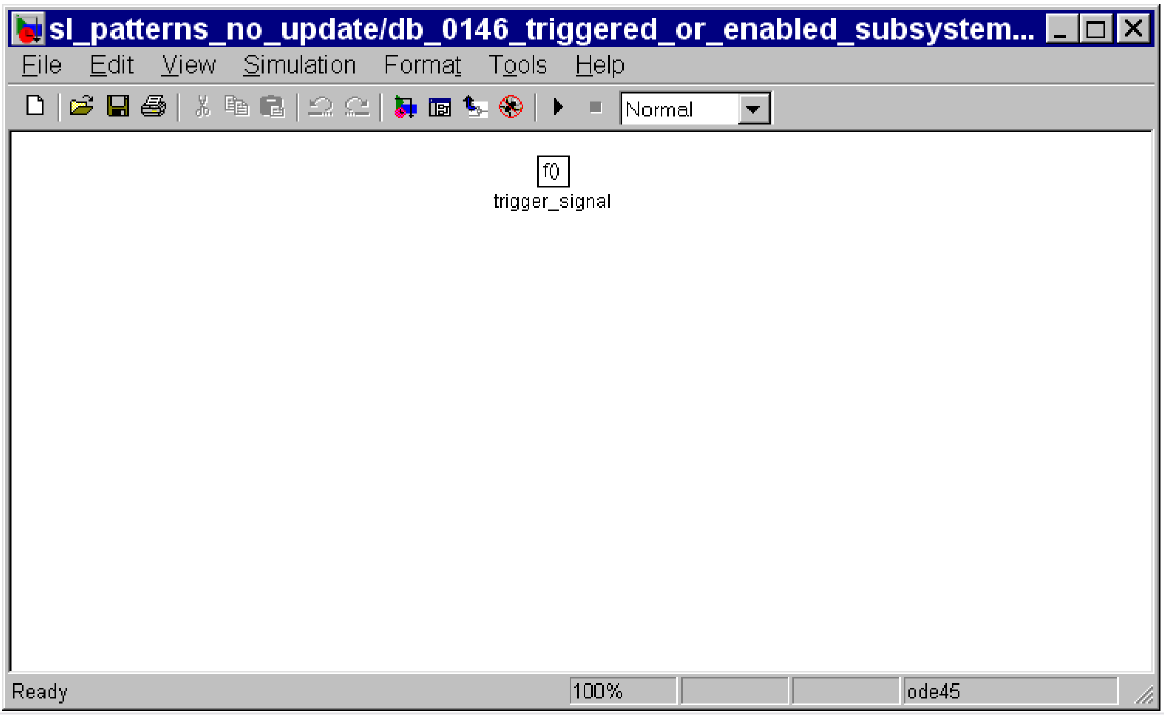
4.4.3.2 0021a: Atomic subsystems

ID: Title	**0021a: Atomic subsystems**
Priority	Strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	

Description	Atomic subsystems shall be used in the following situations: • Enabled, triggered, and action subsystems – in these cases, Simulink requires that atomic subsystems be used • Grouping of major related pieces of functionality within a model – use of atomic subsystems in these situations promotes modularity, testability, and allows for potentially more efficient code generation. • Grouping of related pieces of functionality for reusability, control of block sort order, and generation of reentrant and reusable code modules that may need to be called from outside the "step" function of the generated code. E.g., assume the outer loop controller defines maximum and minimum airspeed limits as a function of altitude and weight; further assume the reasonable proposition that these limits are also needed by the onboard contingency prediction logic. Making the blocks that compute the limits into an atomic subsystem allows that subsystem to be generated as a separate, fully reentrant function that can be called as a "library function" by the contingency predictor. The use of atomic subsystems promotes model reuse and potentially leads to more efficient and more modular generated code.
Benefit	Respecting the guideline ensures ... • a clear system structure. • exchangeable subsystems. • testable subsystems. • Ability to use generated code as "library" functions.
Penalty	Breaking the guideline ... • may cause problems with subsystem integration. • may cause problems or errors with custom tools.
Author	
Last Change	

4.4.3.3 0022: Triggered or enabled subsystems

ID: Title	**0022: Triggered or enabled subsystems**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A subsystem can be triggered, enabled or neither: • The trigger or enable block should be located at the top of the subsystem and centered within the subsystem.

	• The name of the trigger or enable block is the same as the name of the corresponding trigger or enable signal. • Triggered or enabled subsystems should be used in the following situations: TBD  sl_patterns_no_update/db_0146_triggered_or_enabled_subsystem... File Edit View Simulation Format Tools Help Normal f() trigger_signal Ready 100% ode45
Benefit	Respecting the guideline ensures ... • a clear system structure. • a clean subsystem interface. • exchangeable subsystem versions. • testable subsystems. • flexibility for all applications.
Penalty	Breaking the guideline ... • may cause problems with subsystem integration. • may cause a lot of redesign work. • may cause problems or errors with custom tools.
Author	
Last Change	19.09.

4.4.3.4 0022a: Action subsystems

ID: Title	**0022a: Action subsystems**
Priority	strongly recommended

Scope	PROJECT
Automation	possible
Prerequisites	
Description	Action subsystems are if-the-else and switch case constructs plus the atomic subsystems that implement the "action" part of these flow control constructs. Action subsystems are preferred over triggered or enabled subsystems in any situation where a flow of control is being expressed.
Benefit	Respecting the guideline ensures ... • a clear system structure. • a clean subsystem interface. • exchangeable subsystem versions. • testable subsystems. • flexibility for all applications.
Penalty	Breaking the guideline ... • may cause problems with subsystem integration. • may cause a lot of redesign work. • may cause problems or errors with custom tools.
Author	
Last Change	

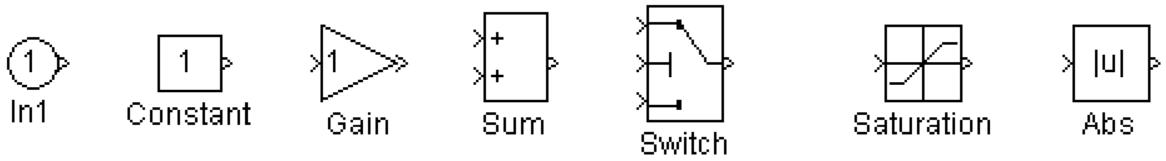
4.4.3.5 0023: Copying/Modification of Subsystems

ID: Title	**0023: Copying/Modification of Subsystems**
Priority	Strongly recommended
Scope	PROJECT
Automation	none
Prerequisites	
Description	If multiple copies of a subsystem are needed, this subsystem should be added to the appropriate library. Links to the master copy may not be disabled. If a variation of a library block is needed, create a new subsystem and copy the contents of the library block. This prevents modifying the master block by accident. Do workspace parameters inside a library block must point to variables contained in the primary .mat file for the library. This .mat file shall be loaded whenever the model is opened.
Benefit	Respecting the guideline ensures ...

	- Easy access to commonly used subsystems - Compatible code
Penalty	Breaking the guideline ... - may cause loss of important data - may cause a lot of redesign work.
Author	
Last Change	19.09.2008,

4.5 Basic blocks

Example basic blocks:



4.5.1 General

4.5.1.1 0024: Display of basic block mask parameters in the attributes format string

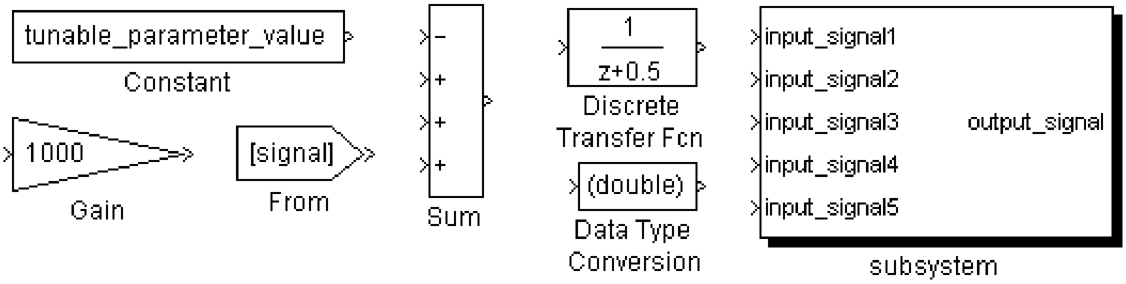
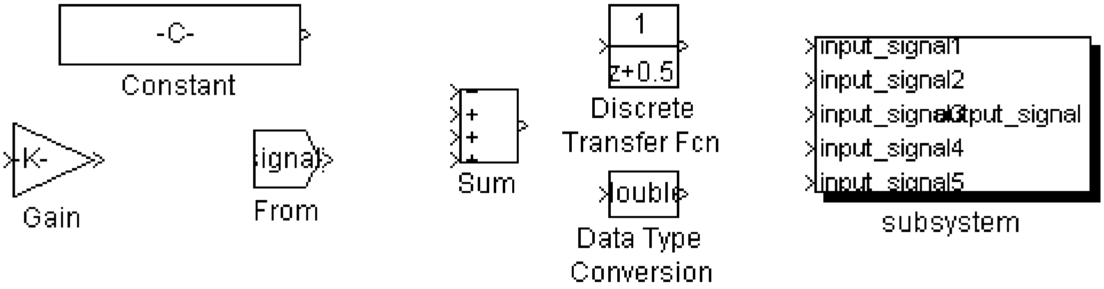
Deleted. This guideline came from the Automotive Advisory Board, has been deemed by experienced PROJECT model developers as counterproductive and full of fail, and has been deleted. Seriously, this leads to the most cluttered block diagrams IMAGINABLE, and is the Simulink equivalent of the dreaded Hungarian notation in C/C++ code. 'Nuff said.

4.5.1.2 0025: Basic block interface signals

ID: Title	**0025: Basic block interface signals**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A basic block ... • has no input busses. • has no output busses. • may have an input vector, if the block is vectorized. • may have an output vector, if the block is vectorized.
Benefit	Respecting the guideline ensures ... • accessible signals. • a clear system structure.
Penalty	Breaking the guideline ... may cause problems with non-accessible signals. may cause a lot of redesign work. may cause problems or errors with custom tools.

Author	
Last Change	18.08.2003

4.5.1.3 0026: Block Resizing

ID: Title	**0026: Block Resizing**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	All blocks in a model must be sized such that their icon is completely visible and recognizable. In particular, any text (e.g. tunable parameters, filenames, equations) in the icon must be readable. Note: this guideline requires resizing of blocks with variable icons or blocks with a variable number of inputs and outputs. Correct:  tunable_parameter_value Constant 1000 Gain [signal] From - + + + Sum 1 z+0.5 Discrete Transfer Fcn (double) Data Type Conversion input_signal1 input_signal2 input_signal3 input_signal4 input_signal5 output_signal subsystem Incorrect:  -C- Constant -K- Gain igna From + + Sum 1 z+0.5 Discrete Transfer Fcn ouble Data Type Conversion input_signal1 input_signal2 input_signal3output_signal input_signal4 input_signal5 subsystem
Benefit	Respecting the guideline ensures ... Readability within the model and the document
Penalty	Breaking the guideline ... Difficult for others to read
Author	
Last Change	18.08.2003

4.5.1.4 0027: Terminator Blocks

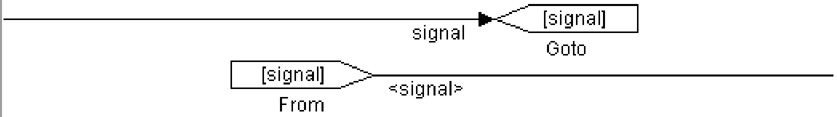
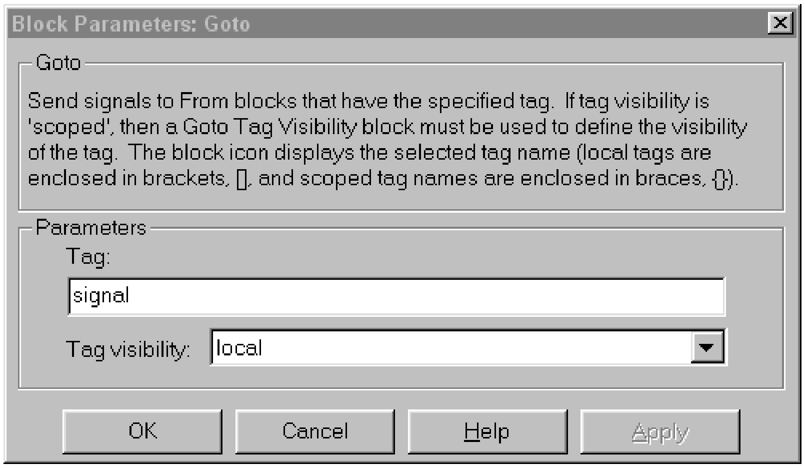
ID: Title	**0027: Terminator Blocks**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	0015
Description	All unconnected outputs should be connected to terminator blocks.
Benefit	Respecting the guideline ensures ... eliminates error messages
Penalty	Breaking the guideline ...
Author	
Last Change	19.09.2008,

4.5.1.5 0027a: Ground Blocks

ID: Title	**0027a: Ground Blocks**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	0015
Description	Ground blocks are not expressly prohibited but are not recommended. Instead, use Constant blocks with appropriate values and datatypes to terminate unconnected inputs. This provides more flexibility as well. There have been anectodal reports of unexpected behavior involving ground blocks, especially with linearization of models. Best practice is to avoid them.
Benefit	Respecting the guideline ensures ... eliminates error messages
Penalty	Breaking the guideline ... • May result in unpredictable linearization results • May result in “update diagram” or code generation errors
Author	
Last Change	

4.5.1.6 0028: Scope of From and Goto Blocks

ID: Title	**0028: Scope of From and Goto Blocks**
Priority	mandatory

Scope	PROJECT
Automation	none
Prerequisites	0009
Description	From and Goto blocks must **not** be used globally within a model. The Tag of From and Goto blocks is the name of the corresponding signal or bus. The Tag visibility of a Goto block is set to "local".   Goto Send signals to From blocks that have the specified tag. If tag visibility is 'scoped', then a Goto Tag Visibility block must be used to define the visibility of the tag. The block icon displays the selected tag name (local tags are enclosed in brackets, [], and scoped tag names are enclosed in braces, {}). Parameters Tag: signal Tag visibility: local Note:From and Goto blocks may be used: - as enable or disable flags - to eliminate signal line crossings - as an interface between rapid prototyping and software-in-the loop
Benefit	Respecting the guideline ensures ... - readability - maintainability - eliminates signal crossings
Penalty	Breaking the guideline ... - causes hidden and undocumented interfaces - requires modifying the model for each environment
Author	
Last Change	18.08.2003

4.5.1.7 0029: Data Output via Goto Blocks

Deleted. Use of GOTO blocks for global data output for TM has been superseded by the use of test points on signals defined with signal objects that have been set to the “Struct” custom storage class. This technique is described later.

4.5.1.8 0031: Parameters

<table>
<tr><th>ID: Title</th><th>0031: Parameters</th></tr>
<tr><td>Priority</td><td>Mandatory</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>none</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>When needed, parameters shall be added to the Matlab workspace or a Model workspace and referenced by Simulink blocks. These parameters must be defined and loaded in m-files that are called automatically when the model is loaded. To prevent large quantities of parameters from cluttering the workspace, they must be organized into structures as listed below:
<table>
<tr><th>Area</th><th>Structure</th><th>m-file</th></tr>
<tr><td>Atmosphere</td><td>TM.atmos</td><td>sim_params.m</td></tr>
<tr><td>Actuators</td><td>TM.act</td><td>sim_params.m</td></tr>
<tr><td>Aerodynamics</td><td>TM.aero</td><td>sim_params.m</td></tr>
<tr><td>AV dimensions</td><td>TM.av_dims</td><td>sim_params.m</td></tr>
<tr><td>Earth model</td><td>TM.earth</td><td>sim_params.m</td></tr>
<tr><td>Engine model</td><td>TM.engine</td><td>sim_params.m</td></tr>
<tr><td>Gear model</td><td>TM.gear</td><td>sim_params.m</td></tr>
</table>
Note that the need to set tunable parameters in model references requires use of ExternGlobal parameters at this time; such parameters cannot be structure members. Preserving parameter tunability in code generated from reference models also requires use of non-auto-storage-class parameters. It is hoped that the Mathworks will remove these limitations in the future, as having non-auto-storage-class parameters can impact the reusability of the generated code. Perhaps the C++ code generation options coming in R2008b will help as well.</td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...
• Model is organized and readable
• Easy access to parameters</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...
• May render model inoperable</td></tr>
</table>

	• May cause a lot of redesign work.
Author	
Last Change	19.09.2008,

4.6 Scalars, vectors, matrices and busses

4.6.1 0032: Default Units and Abbreviations

<table>
<tr><td><b>ID: Title</b></td><td><b>0032: Default Units and Abbreviations</b></td></tr>
<tr><td>Priority</td><td>Strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>none</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>A standard set of units is implemented throughout the aircraft vehicle model. Unless otherwise noted, the following units are used in all calculations and models:
<table>
<tr><th>Unit System</th><th>Measurement</th></tr>
<tr><td>degrees Rankine</td><td>temperature</td></tr>
<tr><td>feet</td><td>distance</td></tr>
<tr><td>feet per second</td><td>velocity</td></tr>
<tr><td>feet per second per second</td><td>acceleration</td></tr>
<tr><td>square feet</td><td>area</td></tr>
<tr><td>cubic feet</td><td>volume</td></tr>
<tr><td>radians</td><td>angle</td></tr>
<tr><td>radians per second</td><td>angular rate</td></tr>
<tr><td>radians per second per second</td><td>angular acceleration</td></tr>
<tr><td>pounds</td><td>force</td></tr>
<tr><td>foot pounds</td><td>moment</td></tr>
<tr><td>pounds per square foot</td><td>pressure</td></tr>
<tr><td>seconds</td><td>time</td></tr>
<tr><td>slugs</td><td>mass</td></tr>
<tr><td>slugs per cubic foot</td><td>density</td></tr>
</table>

The Simulink model uses abbreviations in many notes and naming constructs. The following table lists some of the most come abbreviations along with their respective meanings.
<table>
<tr><th>Abbreviation</th><th>Meaning</th></tr>
</table>
</td></tr>
</table>

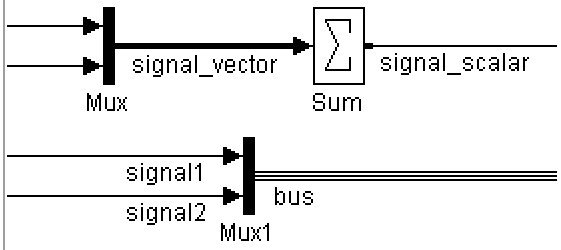
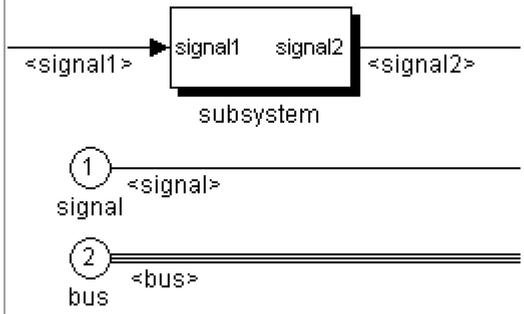
deg	degrees (angle)
degC	degrees Celsius
degF	degrees Fahrenheit
degR	degrees Rankin
dps	degrees per second
dps2	degrees per second per second
in	inches
ips	inches per second
kts	knots
g	acceleration in g’s
mi	statute miles
nm	nautical miles
pct	percent
rpm	revolutions per minute

Benefit	Respecting the guideline ensures ... • Model is compatible with other models/subsystems
Penalty	Breaking the guideline ... • May result in ambiguity over units • May cause a lot of redesign work.
Author	
Last Change	19.09.2008,

4.6.2 Signal labels

4.6.2.1 0032: Use of labels and label instances for signals and busses

ID: Title	**0032: Use of labels and label instances for signals and busses**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The correct use of labels and label instances is as follows: To name a signal at its source, **labels** must be used. To **redisplay** the name of a previously labeled signal on all levels other than the source

	level, **label instances** must be used. A label or a label instance may be shown several times within one level. The labels at the outputs of a bus selector are not editable and can be treated like label instances. No labels or label instances are required … under the mask of basic blocks. for signals of minor importance within one model level. Use of labels:  signal_vector Mux Sum signal_scalar signal1 signal2 bus Mux1 Use of label instances:  <signal1> signal1 signal2 <signal2> subsystem 1 signal <signal> 2 bus <bus> Notes: To create a label, double-click on the line and enter the signal or bus name. To create a label instance, double-click on the line and enter "<". After update diagram the label instance will be shown, enclosed by "<" and ">". To attach multiple copies of a label or label instance to the same line, you can "copy-drag" them with the mouse. If non-standard units are used, append the appropriate unit abbreviation to the end.
Benefit	Respecting the guideline ensures ... • a high quality model. • readable models. • professional documentation. • a clear system structure. • testable systems.
Penalty	Breaking the guideline ... • may cause connection errors.

	- may cause problems with non-accessible signals. - may cause errors in auto-c-code. - may cause problems or errors with custom tools. - may cause a lot of redesign work.
Author	
Last Change	18.08.2003

4.6.2.2 0033: Position of labels and label instances for signals and busses

ID: Title	**0033: Position of labels and label instances for signals and busses**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The labels and label instances must be visually associated with the corresponding signal and not overlap other labels, signals or blocks. Therefore **labels or label instances should be located consistently under or over horizontal lines and close to the corresponding source or destination block**.
Benefit	Respecting the guideline ensures ... - a high quality model. - readable models. - professional documentation. - a clear system structure. - testable systems.
Penalty	Breaking the guideline ... - may cause connection errors. - may cause problems with non-accessible signals. - may cause errors in auto-c-code. - may cause problems or errors with custom tools. - may cause a lot of redesign work.
Author	
Last Change	18.08.2003

4.6.2.3 0034: Signals and busses requiring labels or label instances

ID: Title	**0034:Signals and busses requiring labels or label instances**
Priority	mandatory

<table>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td>0026
0027: Positions of labels and label instances for signals and busses</td></tr>
<tr><td>Description</td><td>
Signal labels or label instances are required for all Simulink interfaces.
This applies to ...

- inports and outports in models
- interfaces and subsystems.
- interfaces of reusable and masked subsystems.
- interfaces of S-Function and Stateflow blocks.
- data transfer with from and goto blocks.

This does not apply to ...

- interfaces of basic blocks (also masked basic blocks).

Signal labels or label instances are required for busses.
This applies to ...

- the bus itself.
- all subbusses within the bus.
- all signals in the bus.

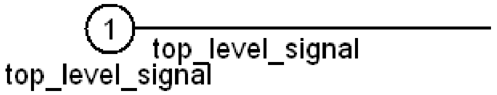
This does not apply to ...

- busses under the mask of basic blocks.

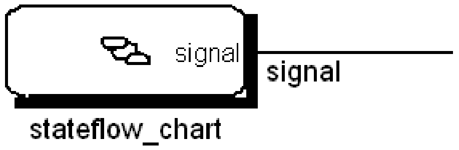
Labels are required ...

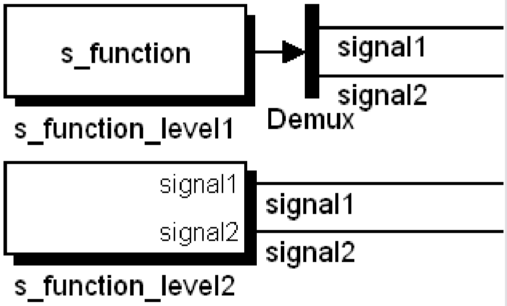
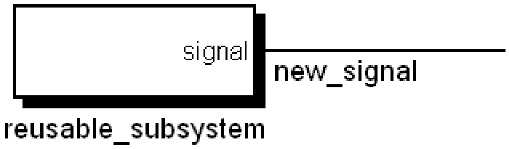
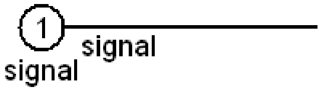
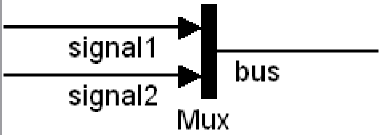
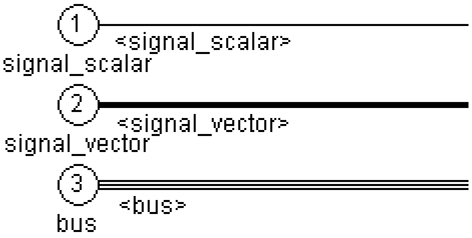
<table>
<tr><td>

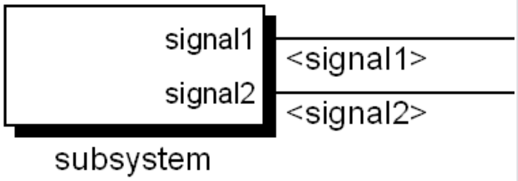
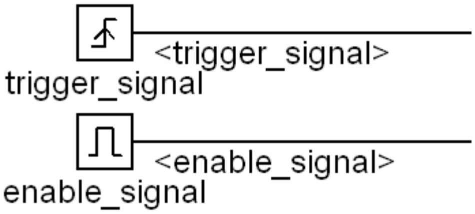
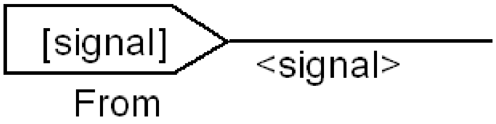
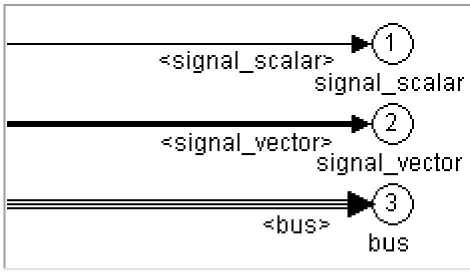
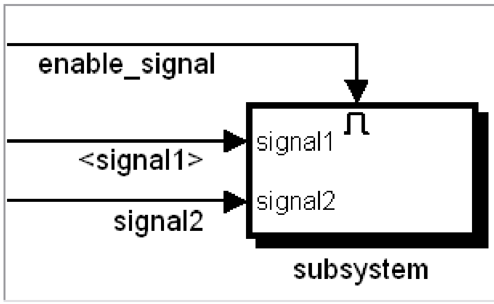
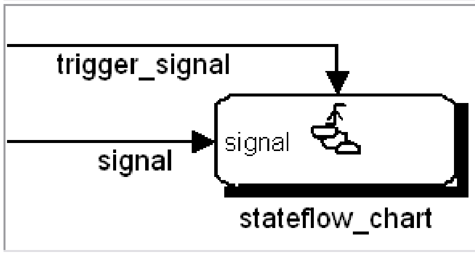
- following an inport block at the top level.

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

- following a Stateflow block.

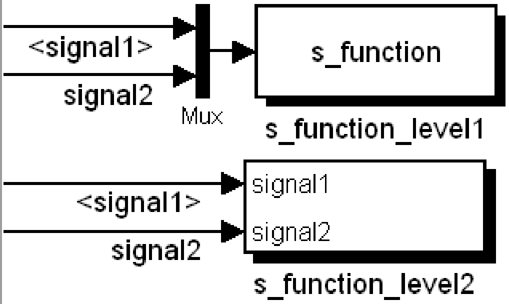
</td><td>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td rowspan="7"></td><td>• following a S-Function.</td><td>
</td></tr>
<tr><td>• following a reusable subsystem.</td><td>
</td></tr>
<tr><td>• following an inport block on the highest level of a reusable subsystem.</td><td>
</td></tr>
<tr><td>following a mux block which is the source of a bus.</td><td>
</td></tr>
<tr><td colspan="2">Labels instances are required ...</td></tr>
<tr><td>• following an inport block. Exceptions:
o Inports on the highest model level.
o Inports under the mask of basic blocks.
o Inports on the highest level of a reusable subsystem.</td><td>
</td></tr>
</table>

- following a system or subsystem block. Exception: o Reusable subsystems.	
- following enable block or a trigger block with a output port.	 <trigger_signal> trigger_signal <enable_signal> enable_signal
- following a from block.	
Labels OR label instances are required ...	
- before an outport block. Exception: o Outports under the mask of basic blocks.	 1 <signal_scalar> signal_scalar 2 <signal_vector> signal_vector 3 <bus> bus
- before an inport or trigger/enable port of a subsystem block.	 enable_signal <signal1> signal2 signal1 signal2 subsystem
- before an inport or trigger/enable port of a Stateflow block.	

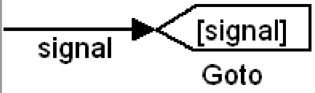
<table>
<tr><td></td><td>
<table>
<tr><td>

- before an inport of a S-Function block.

</td><td>


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

- before a goto block.

</td><td>


</td></tr>
</table>
</td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...

- a high quality model.
- readable models.
- professional documentation.
- a clear system structure.
- testable systems.

</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...

- may cause connection errors.
- may cause problems with non-accessible signals.
- may cause errors in auto-c-code.
- may cause problems or errors with custom tools.
- may cause a lot of redesign work.

</td></tr>
<tr><td>Author</td><td></td></tr>
<tr><td>Last Change</td><td>18.08.2003</td></tr>
</table>

4.6.3 Vector signals

4.6.3.1 0035: Types and use of vectors

ID: Title	**0035: Types and use of vectors**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A vector signal ...

	• is one-dimensional – i.e., no row or column vectors • has only one set of properties like size, data type, min, max, description and so on. It is not a bus, which is a group of signals. • may be split up in scalar signals with a demux-block, not with a bus-selector-block. • may be created by feeding scalar signals into a mux-block. • may contain named scalar signals. However, it is strongly recommended that vectors created with Mux blocks be restricted to closely related signals. An example of a vector that contains named scalar signals is a vector of the Euler angles roll, pitch, and yaw. To visualize vectors in Simulink models, the option "Format / Wide Vector Lines" is selected. To visualize vector sizes in Simulink models, the option "Format / Vector Line Widths" may be selected.
Benefit	Respecting the guideline ensures ... a high quality model. readable models. professional documentation. a clear system structure. testable systems. possible improvement in readability of code.
Penalty	Breaking the guideline ... • may cause problems with non-accessible signals. • may cause errors in auto-c-code. • may cause problems or errors with custom tools. • may cause a lot of redesign work.
Author	
Last Change	19.09.2008, K

4.6.4 Signal busses

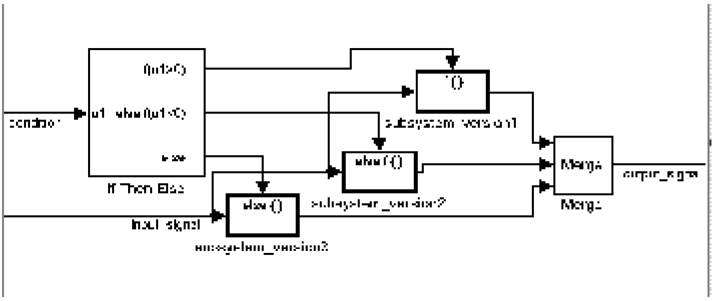
4.6.4.1 0036: Use of busses

ID: Title	**0036: Use of busses**

Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	A bus ... • is a group of scalar signals, vectors, arrays, and/or sub-busses. • Has a common sample time for all elements of the bus • is not constrained to have a common set of properties like size, data type, min, max, description and so on. • is defined by the data content. Generally buses consist of related heterogenous content. • may be split up in scalar signals, vectors or subbusses with a bus-selector-block. • must not split up with a demux-block. • must be created by feeding scalar signals, vectors or busses into a Bus Creator block. • must be named. • must not contain unnamed scalar signals or vectors. • Maps to a C structure in generated code if a bus object is used to define it (mandatory for reference model I/O) and is output as a nonvirtual bus. To visualize busses in Simulink models, the option "Format / Wide Vector Lines" is selected. To visualize bus sizes in Simulink models, the option "Format / Vector Line Widths" may be selected.
Benefit	Respecting the guideline ensures ... • a high quality model. • readable models. • professional documentation. • a clear system structure. • testable systems.
Penalty	Breaking the guideline ... • may cause problems with non-accessible bus signals. • may cause errors in auto-c-code. • may cause problems or errors with custom tools. • may cause a lot of redesign work.
Author	
Last Change	19.09.2008,

4.7 Patterns

4.7.1 0037: Simulink patterns for If-then-else-if constructs

<table>
<tr><td><b>ID: Title</b></td><td><b>0037: Simulink patterns for If-then-else-if constructs</b></td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for If-then-else-if constructs within Simulink:
<table>
<tr><th>Equivalent Functionality</th><th>Simulink pattern</th></tr>
<tr><td>IF THEN ELSE IF
with subsystems:
output_version1 =
function_version1(input_signal);
output_version2 =
function_version2(input_signal);
output_version3 =
function_version3(input_signal);
if (condition1) {
output_signal = output_version1;
}
else if (condition2) {
output_signal = output_version2;
}
else {
output_signal = output_version3;
}</td><td>
</td></tr>
</table>

- Use of the Merge block to combine the individual if-else subsystem output(s) is strongly recommended in most situations; the Merge block fully supports buses and is the Mathworks-documented way to perform the needed functionality. However, use of the Merge block can be problematical if the if-the-else resides in a signal path that needs to be linearized for analyses, trimming, etc. If the Merge block proves problematical in a particular situation, it can be replaced with a Multiport Switch block and extra logic added to set the switch index.
- The condition could be a signal or a tunable parameter. For tunable parameter conditions the constant-block is used.
</td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...

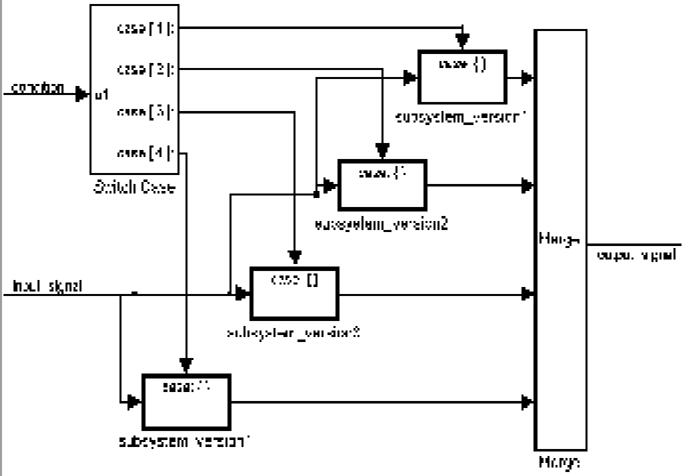
- uniform appearance of models, code and documentation.
- reusable models.
</td></tr>
</table>

	- readable models. - a clear system structure.
Penalty	Breaking the guideline ... - may cause errors in functionality. - may cause problems or errors with custom tools. - may cause a lot of redesign work. - may cause confusing presentations. - may cause problems with system integration.
Author	
Last Change	19.09.2008,

4.7.2 0038: Simulink patterns for case constructs

ID: Title	**0038: Simulink patterns for case constructs**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The following patterns are used for case constructs within Simulink:

Equivalent Functionality	**Simulink pattern**

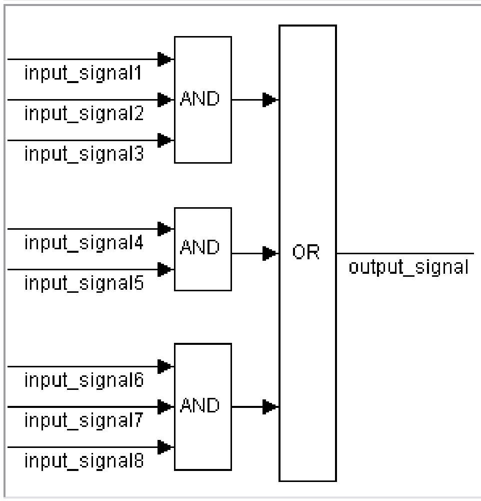
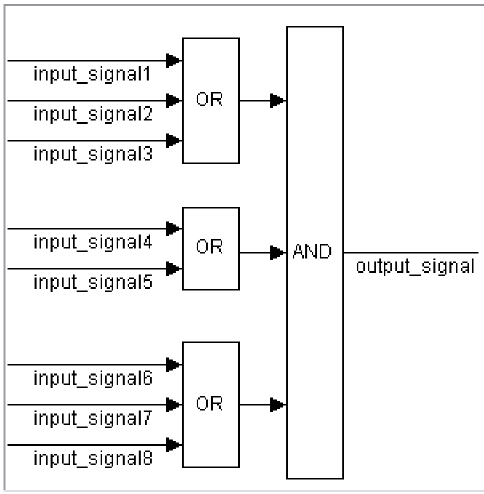
	CASE
	with subsystems:
	output_version1 =
	function_version1(input_signal);
	output_version2 =
	function_version2(input_signal);
	output_version3 =
	function_version3(input_signal);
	output_version4 =
	function_version4(input_signal);
	switch (selection) {
	case 1:
	output_signal = output_version1;
	break;
	case 2:
	output_signal = output_version2;
	break;
	case 3:
	output_signal = output_version3;
	break;
	case 4:
	output_signal = output_version4;
	}
	 case [1]:
	case [2]:
	case [3]:
	case [4]:
	condition
	u1
	Switch Case
	case { }
	subsystem_version1
	case { }
	subsystem_version2
	case { }
	subsystem_version3
	case { }
	subsystem_version1
	input_signal
	Merge
	output_signal
	Merge
	• Use of the Merge block to combine the individual case subsystem output(s) is strongly recommended in most situations; the Merge block fully supports buses and is the Mathworks-documented way to perform the needed functionality. However, use of the Merge block can be problematical if the case resides in a signal path that needs to be linearized for analyses, trimming, etc. If the Merge block proves problematical in a particular situation, it can be replaced with a Multiport Switch block and extra logic added to set the switch index.
	• The condition could be a signal or a tunable parameter. For tunable parameter conditions the constant-block is used.

Benefit	Respecting the guideline ensures ... • uniform appearance of models, code and documentation. • reusable models. • readable models.

	- a clear system structure.
Penalty	Breaking the guideline ... - may cause errors in functionality. - may cause problems or errors with custom tools. - may cause a lot of redesign work. - may cause confusing presentations. - may cause problems with system integration.
Author	
Last Change	16.12.2003

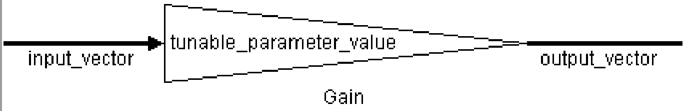
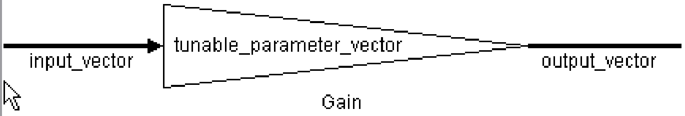
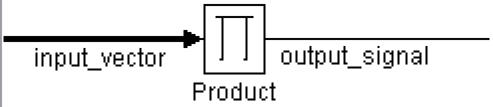
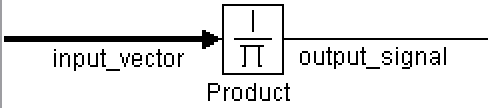
4.7.3 0039: Simulink patterns for logical constructs

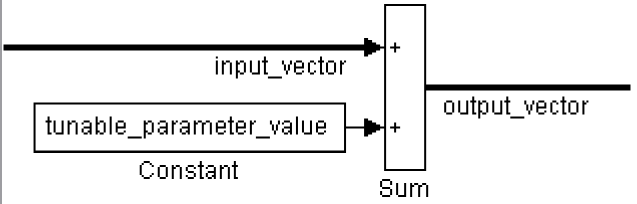
ID: Title	**0039: Simulink patterns for logical constructs**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	

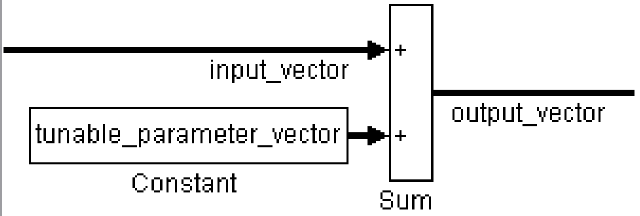
<table>
<tr><td>Description</td><td>
<table>
<tr><td colspan="2">The following patterns are used for logical combinations within Simulink:</td></tr>
<tr><td><b>Equivalent Functionality</b></td><td><b>Simulink pattern</b></td></tr>
<tr><td>Combination of logical signals: conjunctive</td><td>
</td></tr>
<tr><td>Combination of logical signals: disjunctive</td><td>
</td></tr>
</table>
</td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...
• uniform appearance of models, code and documentation.
• reusable models.
• readable models.
• a clear system structure</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...
• may cause errors in functionality.
• may cause problems or errors with custom tools.
• may cause a lot of redesign work.
• may cause confusing presentations.
• may cause problems with system integration</td></tr>
</table>

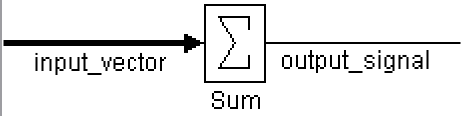
Author	
Last Change	18.08.2003

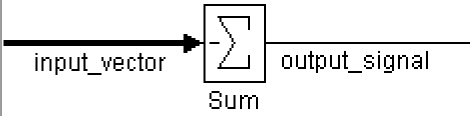
4.7.4 0040: Simulink patterns for vector signals

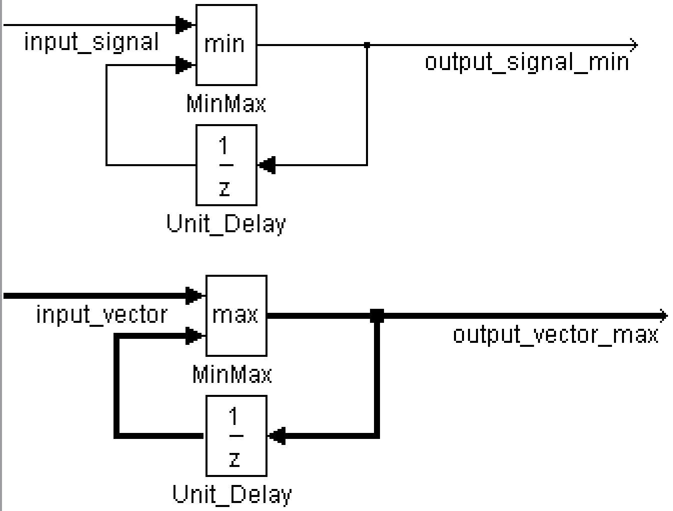
<table>
<tr><td><b>ID: Title</b></td><td><b>0040: Simulink patterns for vector signals</b></td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for vector signals within Simulink:
<table>
<tr><th>Equivalent Functionality</th><th>Simulink pattern</th></tr>
<tr><td>Vector loop:
for (i=0; i>input_vector_size; i++)
{
output_vector(i) = input_vector(i) * tunable_parameter_value;
}</td><td>
</td></tr>
<tr><td>Vector loop:
for (i=0; i>input_vector_size; i++)
{
output_vector(i) = input_vector(i) * tunable_parameter_vector(i);
}</td><td>
</td></tr>
<tr><td>Vector loop:
output_signal = 1;
for (i=0; i>input_vector_size; i++)
{
output_signal = output_signal * input_vector(i);
}</td><td>
</td></tr>
<tr><td>Vector loop:
output_signal = 1;
for (i=0; i>input_vector_size; i++)
{
output_signal = output_signal / input_vector(i);
}</td><td>
</td></tr>
</table></td></tr>
</table>

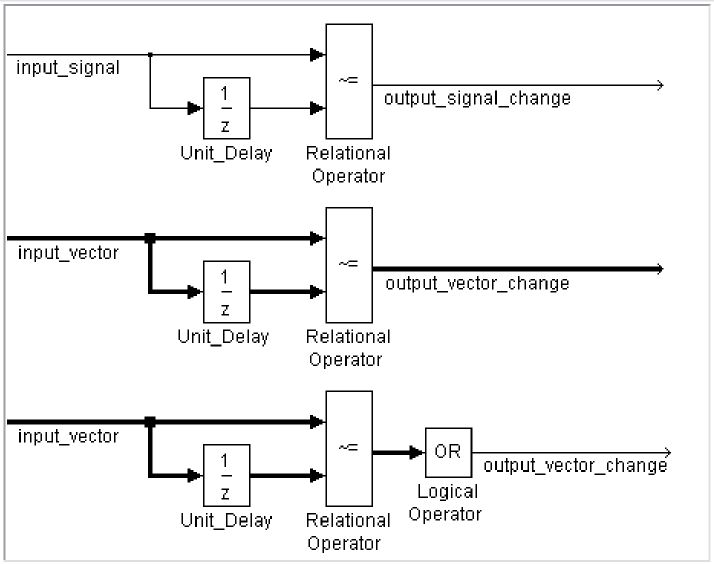
	Vector loop:
	for (i=0; i>input_vector_size; i++)
	{
	output_vector(i) = input_vector(i)
	+ tunable_parameter_value;
	}
		 input_vector
		+
		+
		output_vector
		tunable_parameter_value
		Constant
		Sum

	Vector loop:
	for (i=0; i>input_vector_size; i++)
	{
	output_vector(i) = input_vector(i)
	+ tunable_parameter_vector(i);
	}
		 input_vector
		+
		+
		output_vector
		tunable_parameter_vector
		Constant
		Sum

	Vector loop:
	output_signal = 0;
	for (i=0; i>input_vector_size; i++)
	{
	output_signal = output_signal +
	input_vector(i);
	}
		

	Vector loop:
	output_signal = 0;
	for (i=0; i>input_vector_size; i++)
	{
	output_signal = output_signal -
	input_vector(i);
	}
		

	Minimum or maximum of a signal or a vector over time:	 input_signal min output_signal_min MinMax 1 z Unit_Delay input_vector max output_vector_max MinMax 1 z Unit_Delay

	Change event of a signal or a vector:  input_signal 1 z Unit_Delay ~= Relational Operator output_signal_change input_vector 1 z Unit_Delay ~= Relational Operator output_vector_change input_vector 1 z Unit_Delay ~= Relational Operator OR Logical Operator output_vector_change
Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.
Penalty	Breaking the guideline ... • may cause errors in functionality. • may cause problems or errors with custom tools. • may cause a lot of redesign work. • may cause confusing presentations. • may cause problems with system integration. • may constrain autocode generation.
Author	
Last Change	18.08.2003

4.7.5 Design Patterns for Algebraic Systems (to be written)

Table lookups
Matrix-vector manipulations
Data stores
Iteration
Embedded Matlab

4.7.6 Design Patterns for Dynamic Systems (to be written)

Self-resetting or initializing filters / integrators
Anti-windup for integrators
Pseudo-slewers and/or faders

4.7.7 Design Patterns for Real-world Situations (to be written)

Supporting telemetry
In-flight trim, either as part of the sim environment or as part of an in-flight restart
Tunable parameters
Signal logging
Interface to redundant systems (e.g., voting mode transitions, voting and resetting states across a CCDL)
Interfaces to external commands (e.g., operator commands)
Guarding against divide-by-zero

4.7.8 Design Patterns for Testing (to be written)

Developing unit test frameworks
Supporting unit tests
Running unit tests
Supporting data collection in Simulink environment
Supporting coverage tests in Simulink environment

4.7.9 Supporting Code Generation (to be written)

Design patterns for code generation
Simulation parameter settings that affect code generation
Data type issues

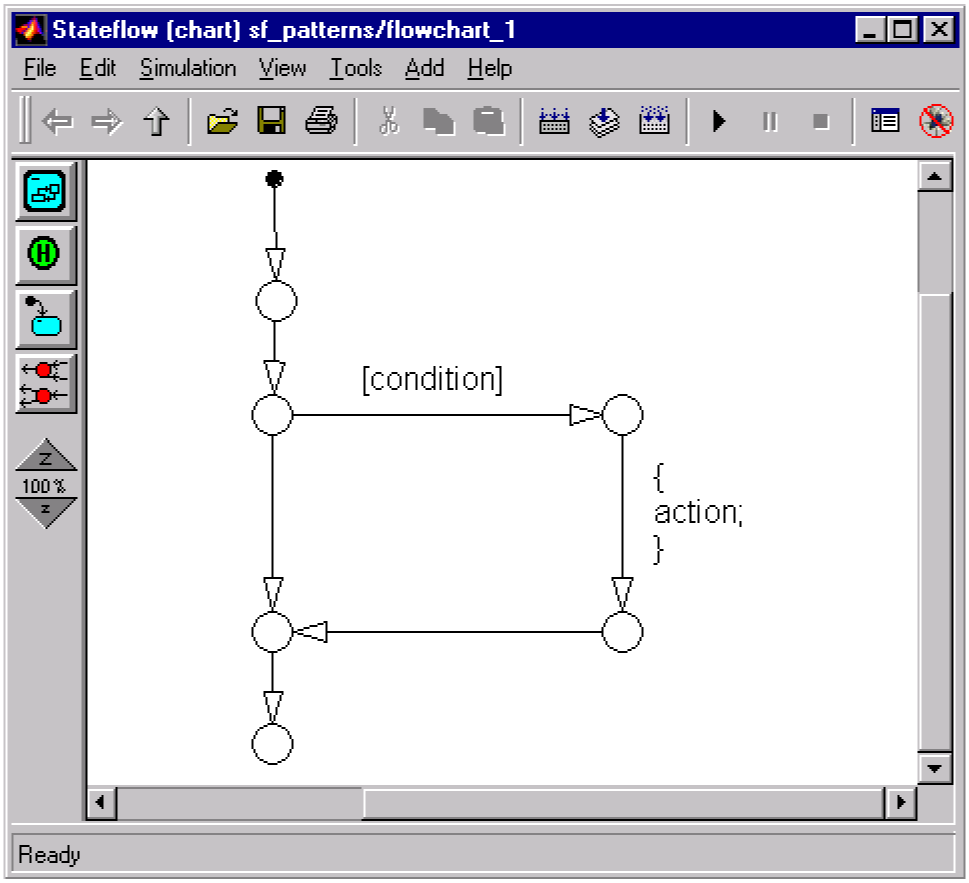
4.7.10 Dealing with Matlab / Simulink Epic Fail Situations (to be written)

Stupid Merge block tricks
How to stop Signal Logging from crashing Matlab
Tunable parameter limitations
Kludging block execution order using the While iterator and Switch Case action subsystems
Tricking the “log” function into not needing non-finite number support

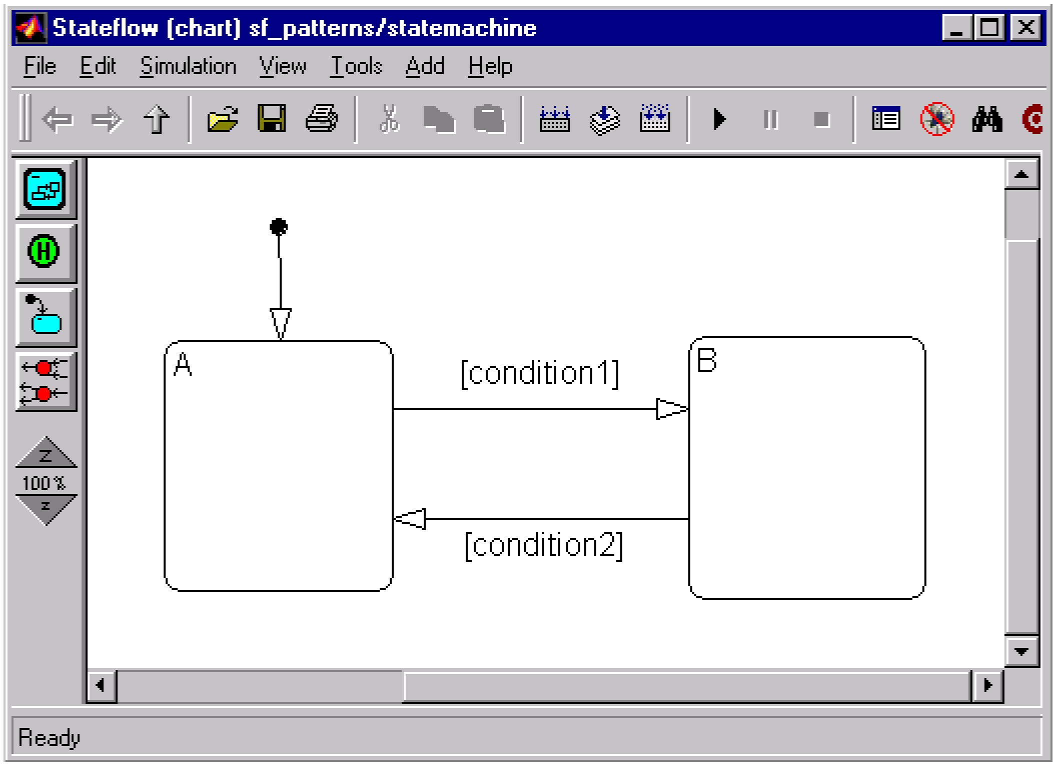
5 STATEFLOW

5.1 General

Stateflow-block with Flowchart:



Stateflow-block with statemachine:


Stateflow (chart) sf_patterns/statemachine
File Edit Simulation View Tools Add Help
100 %
A
B
[condition1]
[condition2]
Ready

5.1.1 0041: Use of Stateflow

ID: Title	**0041: Use of Stateflow**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Stateflow is used to ... • model control flow. • perform logical operations. • perform basic mathematical operations.
Benefit	Respecting the guideline ensures ... • readability. • efficient models and auto-c-code. • eliminates complex mathematical calculations within Stateflow.
Penalty	Breaking the guideline ...

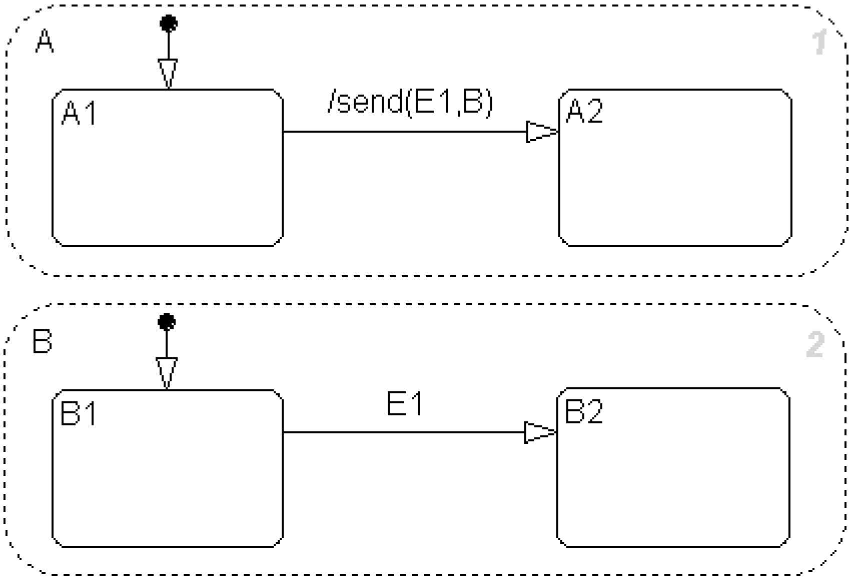
	• may cause inefficient models or auto-c-code.
Author	
Last Change	18.08.2003

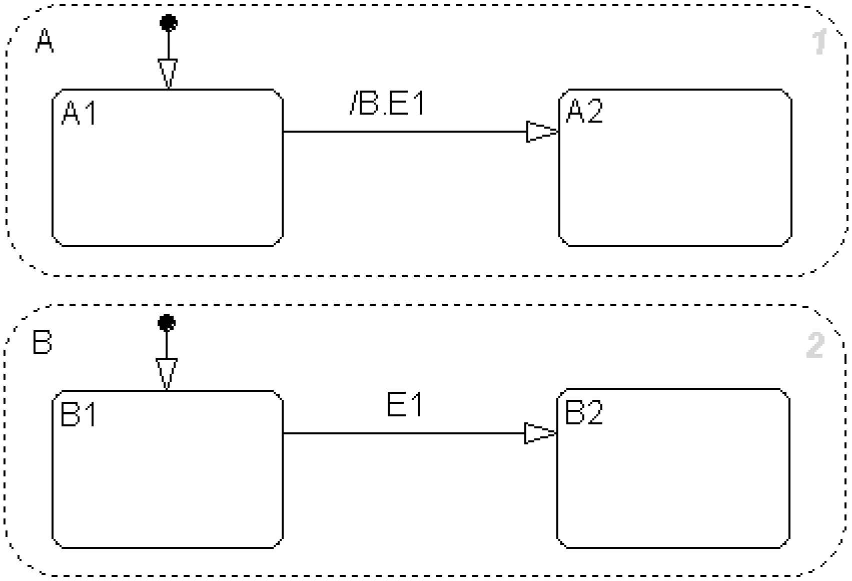
5.1.2 0042: Scope of events

ID: Title	**0042: Scope of events**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The following rules apply to events in Stateflow: • All events of a Stateflow-block 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).
Benefit	Respecting the guideline ensures ... accessible events in Stateflow. a clear system structure. reusable models. readable models. testable systems.
Penalty	Breaking the guideline ... may cause errors in functionality, e.g. auto-c-code generation. may cause problems or errors with custom tools. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

5.1.3 0043: Event Broadcasts

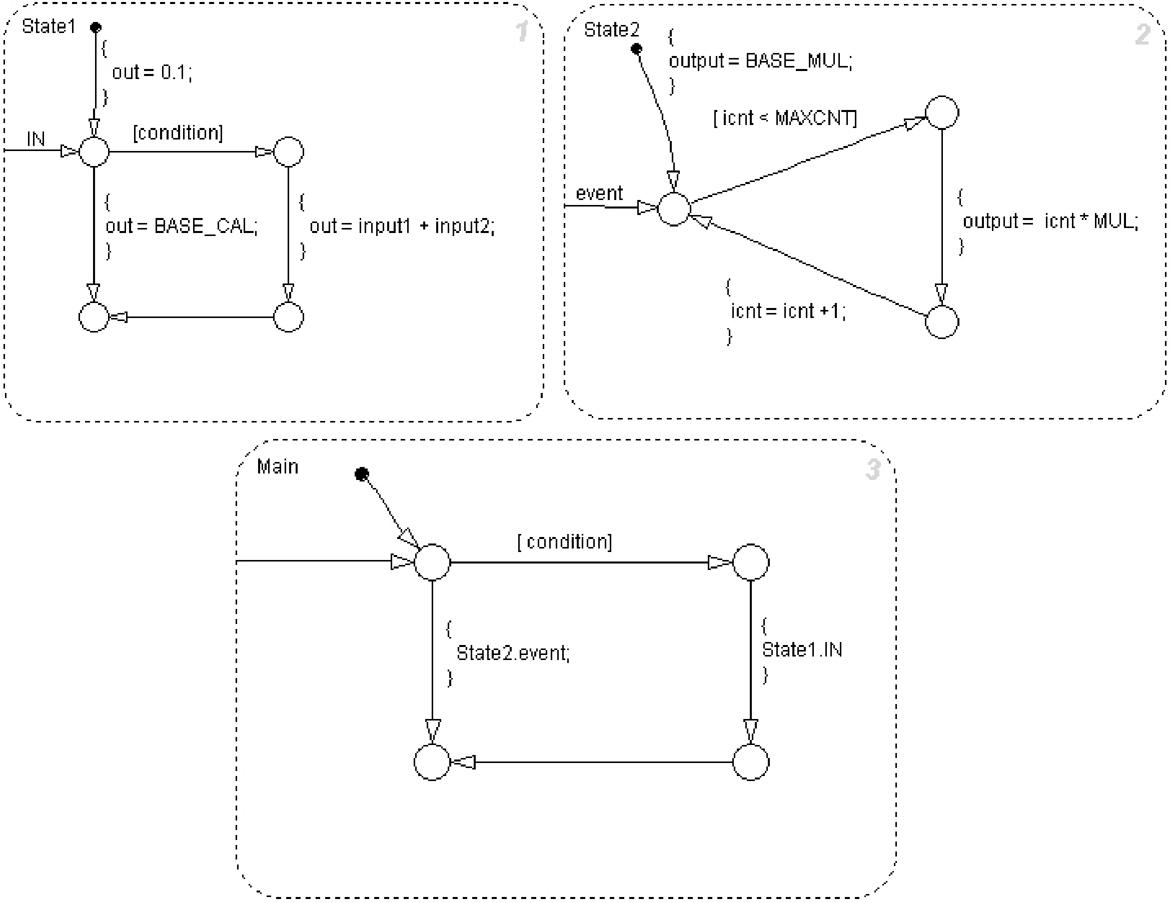
ID: Title	**0043: Event Broadcasts**

Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	0037
Description	The following rules apply to event broadcasts in Stateflow: Directed event broadcasts are the only type of event broadcasts allowed. The send syntax or qualified event names are used to direct the event to a particular state. Multiple send statements should be used to direct an event to more than one state. Example using the send syntax:  A 1 A1 /send(E1,B) A2 B 2 B1 E1 B2 Example using qualified event names:

	
Benefit	Respecting the guideline ensures ... • a clear system structure. • readable models. • efficient code generation • guarantees a deterministic system
Penalty	Breaking the guideline ... • may cause a lot of redesign work.
Author	
Last Change	18.08.2003

5.1.4 0044: Directed Events In Parallel States

ID: Title	**0039: Directed Events In Parallel States**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	0037 Scope_of_events 0038
Description	The following rules apply to directed events in parallel states: • The master state (Main in this example) that is directing the events must be located below the other states. (It must have a higher number in the upper right corner.) To

	ensure that any state that may be awoken by a directed event is activated prior to being awoken by the directed event. States that are not activated prior to being awoken, when awoken the first time by the directed event will be activated but not evaluated.
	Example using the qualified event names:
	
	 State1
	{
	out = 0.1;
	}
	IN
	[condition]
	{
	out = BASE_CAL;
	}
	{
	out = input1 + input2;
	}
	1
	State2
	{
	output = BASE_MUL;
	}
	[icnt < MAXCNT]
	event
	{
	output = icnt * MUL;
	}
	{
	icnt = icnt +1;
	}
	2
	Main
	[condition]
	{
	State2.event;
	}
	{
	State1.IN
	}
	3

Benefit	Respecting the guideline ensures ... • at initialization all states will be active.
Penalty	Breaking the guideline ... • at initialization the first states will not be activated, thus resulting in unexpected behavior. • may cause a lot of redesign work.
Author	
Last Change	18.08.2003

5.1.5 0045: MATLAB commands in Stateflow

ID: Title	**0045: MATLAB commands in Stateflow**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The following rules apply to logic in Stateflow: • MATLAB functions are not used. • MATLAB instructions are not used. • MATLAB operators are not used. • Project-specific MATLAB-functions are not used.
Benefit	Respecting the guideline ensures ... • system integration without problems. • a clear system structure. • reusable models. • readable models. • testable systems.
Penalty	Breaking the guideline ... may cause errors in functionality, e.g. auto-c-code generation. may cause problems or errors with custom tools. may cause a lot of redesign work.
Last Change	18.08.2003

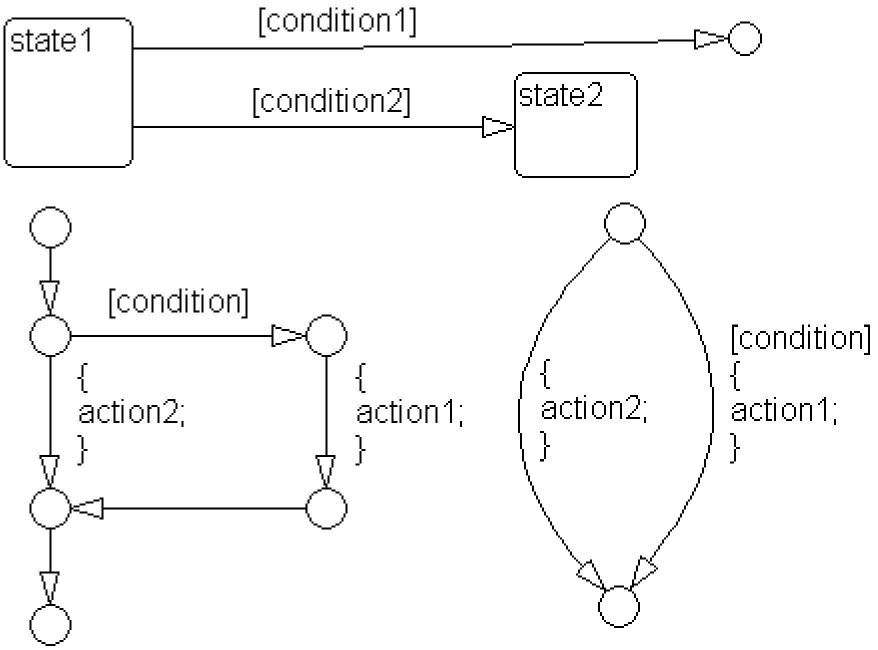
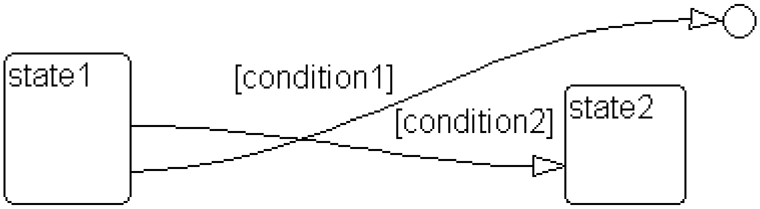
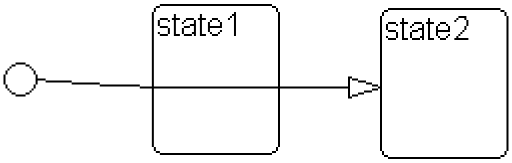
5.1.6 0046: Pointers in Stateflow

ID: Title	**0046: Pointers in Stateflow**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	In a Stateflow diagram, pointers to custom code variables are not used.
Benefit	Respecting the guideline ensures ...

	• system integration without problems. • a clear system structure. • reusable models. • readable models. • testable systems.
Penalty	Breaking the guideline ... • may cause errors in functionality, e.g. auto-c-code generation. • may cause problems or errors with custom tools. • may cause a lot of redesign work.
Author	
Last Change	18.08.2003

5.1.7 0047: Stateflow transition appearance

ID: Title	**0047: Stateflow transition appearance**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Transitions in Stateflow ... • do not cross each other, if possible. • are not drawn one upon the other. • do not cross any states, junctions or text-fields. Transition labels can be visually associated to the corresponding transition. Correct:

	
	 state1
	[condition1]
	[condition2]
	state2
	[condition]
	{
	action2;
	}
	{
	action1;
	}
	{
	action2;
	}
	[condition]
	{
	action1;
	}
	Incorrect.
	
	
	

Benefit	Respecting the guideline ensures ... • readable models. • uniform appearance of models. • reusable models. • understandable presentations.
Penalty	Breaking the guideline ... • may cause unreadable models. • may cause a lot of redesign work.
Author	

Last Change	18.08.2003

5.1.8 0048: History Junction

ID: Title	**db_0138: History Junction**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	History Junctions are not used.
Benefit	Respecting the guideline ensures ... a clear system structure. reusable models. readable models. testable systems.
Penalty	Breaking the guideline ... uses extra memory. difficult concept for the casual user to grasp. may cause interpretation errors in hand code. may cause problems or errors with custom tools. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

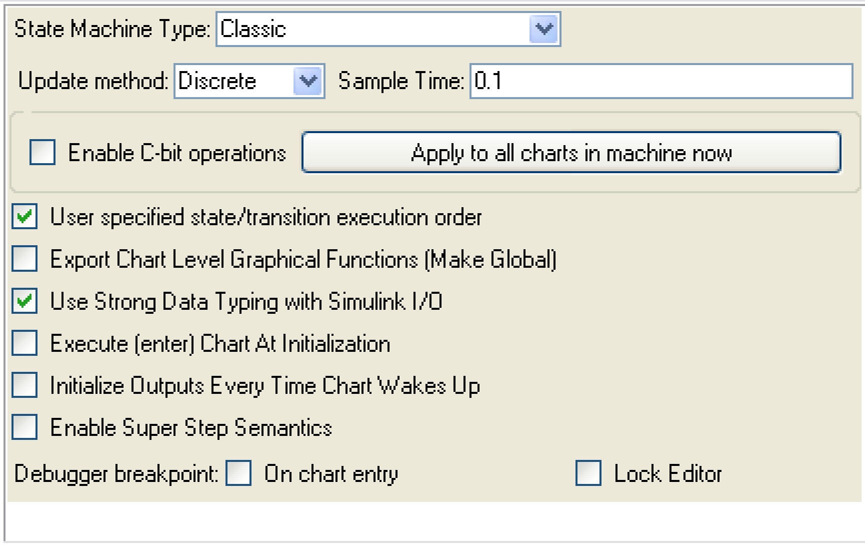
5.1.9 0048a: Logical operators in Stateflow Action Language

ID: Title	**0048a: Logical operators in Stateflow Action Language**
Priority	Mandatory
Scope	PROJECT
Automation	possible
Prerequisites	

Description	In Stateflow Action Language (used in transition conditionals, block actions, etc), the logical AND (&&), logical OR (&&), and logical negation (!) shall be used to mean the logical AND of two operands, the logical OR of two operands, or the logical negation of a single operand. Use of the single &, |, ~ to mean logical AND, OR, NOT is prohibited. This rule prevents hard-to-find breakage if, for example, the ~ was originally used in a chart to mean logical NOT, then at some point "enable C-bit operations" is turned on in the chart. The ~ now means bitwise complement instead of logical negation, and the meaning of a chart can be totally changed.
Benefit	Respecting the guideline ensures ... the Stateflow chart will not break if C-bit ops are enabled.
Penalty	Breaking the guideline ... may cause subtle and difficult-to-find safety-critical errors in your chart. may cause problems with some custom tools.
Author	
Last Change	

5.1.10 0048b: User-specified transition order

ID: Title	**0048b: User-specified transition order**
Priority	Mandatory
Scope	PROJECT
Automation	possible
Prerequisites	
Description	All charts shall have user-specified transition order enabled. In the chart Properties dialog, ensure that the "User specified state/transition execution order" checkbox is checked.

	 State Machine Type: Classic Update method: Discrete Sample Time: 0.1 □ Enable C-bit operations ☑ User specified state/transition execution order □ Export Chart Level Graphical Functions (Make Global) ☑ Use Strong Data Typing with Simulink I/O □ Execute (enter) Chart At Initialization □ Initialize Outputs Every Time Chart Wakes Up □ Enable Super Step Semantics Debugger breakpoint: □ On chart entry □ Lock Editor
Benefit	Respecting the guideline ensures ... Full control of state/execution ordering lies with the diagram developer.
Penalty	Breaking the guideline ... may cause subtle and difficult-to-find safety-critical errors in your chart due to Stateflow’s arcane and brittle logic for default determination of state/transition execution order. may cause problems with some custom tools.
Author	
Last Change	

5.2 Naming

5.2.1 0049: Stateflow port names

ID: Title	**0049: Stateflow port names**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The name of a Stateflow input/output is the same as the corresponding signal. Exception: Reusable Stateflow-blocks may have different port names. Those port names are constructed like Simulink signal names.
Benefit	Respecting the guideline ensures ...

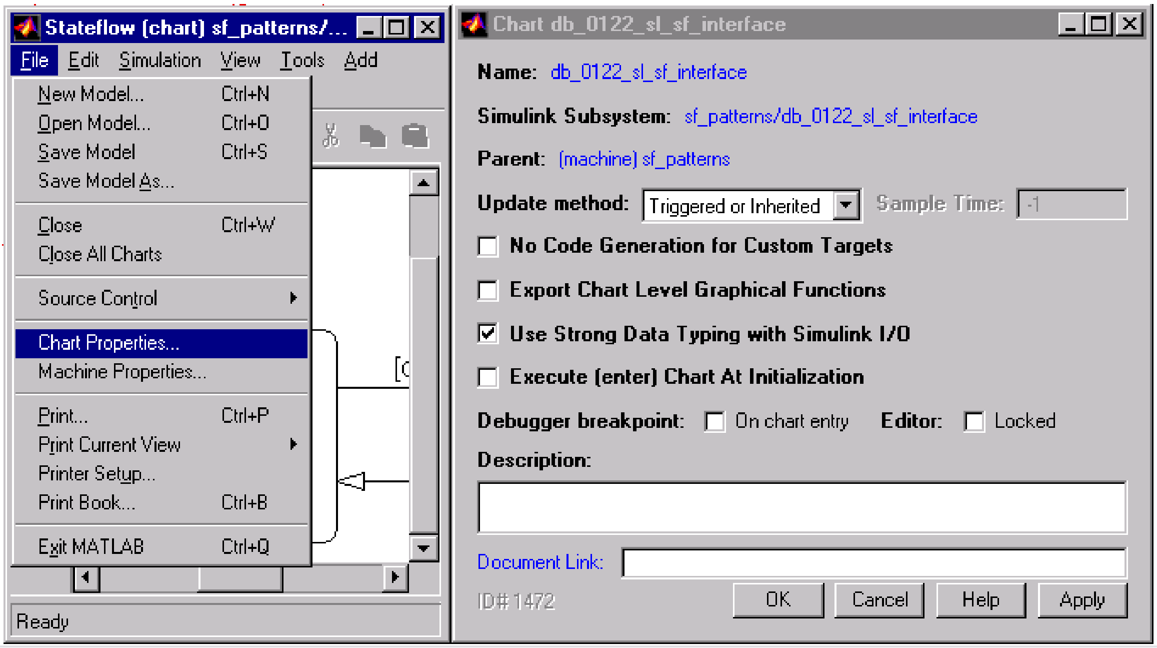
	accessible Stateflow signals and tunable parameters. common name elements..
Penalty	Breaking the guideline ... may cause problems with non-accessible Stateflow signals. may cause problems with some custom tools.
Author	
Last Change	18.08.2003

5.3 Interface to Simulink

5.3.1 General

5.3.1.1 0050: Stateflow/Simulink interface signals and parameters

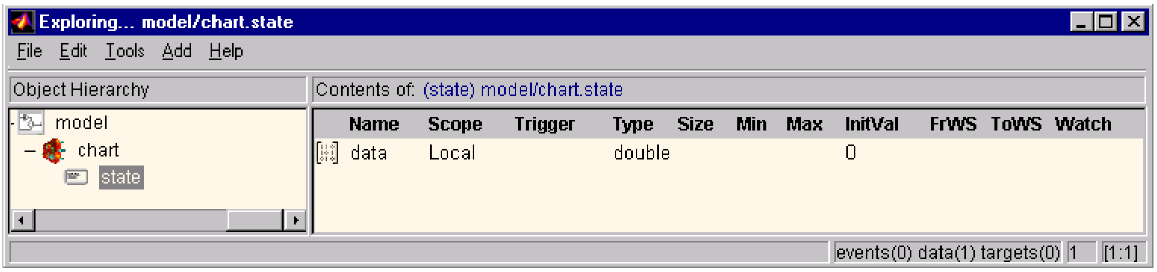
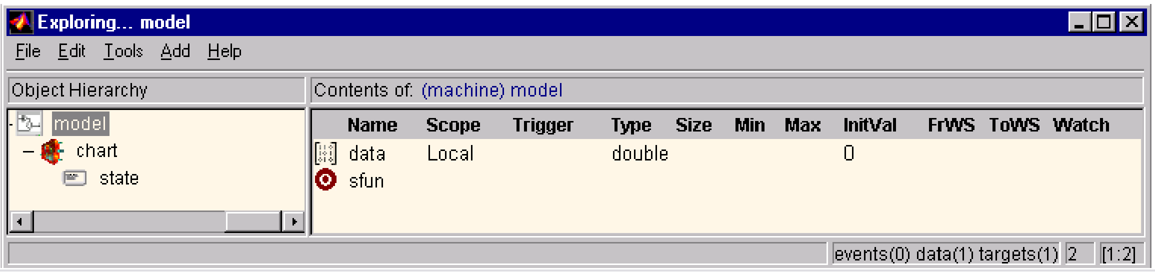
ID: Title	**0050: Stateflow/Simulink interface signals and parameters**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	0044 0001
Description	A Stateflow-block ... has only input signals, input vectors,and input busses. has a labeled input signal. can have tunable parameter inputs but cannot handle parameters that are part of a workspace structure has only output signals, output buses, and output vectors. has a labeled output signal. can have trigger outputs can be triggered ("File / Chart properties / Update method / triggered"). uses strong data typing with Simulink (The option "Use Strong Data Typing with

	Simulink I/O" is selected).  Stateflow (chart) sf_patterns/... File Edit Simulation View Tools Add New Model... Ctrl+N Open Model... Ctrl+O Save Model Ctrl+S Save Model As... Close Ctrl+W Close All Charts Source Control Chart Properties... Machine Properties... Print... Ctrl+P Print Current View Printer Setup... Print Book... Ctrl+B Exit MATLAB Ctrl+Q Ready Chart db_0122_sl_sf_interface Name: db_0122_sl_sf_interface Simulink Subsystem: sf_patterns/db_0122_sl_sf_interface Parent: (machine) sf_patterns Update method: Triggered or Inherited Sample Time: -1 □ No Code Generation for Custom Targets □ Export Chart Level Graphical Functions ☑ Use Strong Data Typing with Simulink I/O □ Execute (enter) Chart At Initialization Debugger breakpoint: □ On chart entry Editor: □ Locked Description: Document Link: ID# 1472
Benefit	Respecting the guideline ensures ... a clear system structure. a clean Stateflow interface. exchangeable subsystem versions. testable subsystems.
Penalty	Breaking the guideline ... may cause problems with subsystem integration. may cause a lot of redesign work. may cause problems or errors with custom tools.
Author	
Last Change	18.08.2003

5.4 Internal signals

5.4.1 General

5.4.1.1 0051: Scope of internal signals and local auxiliary variables

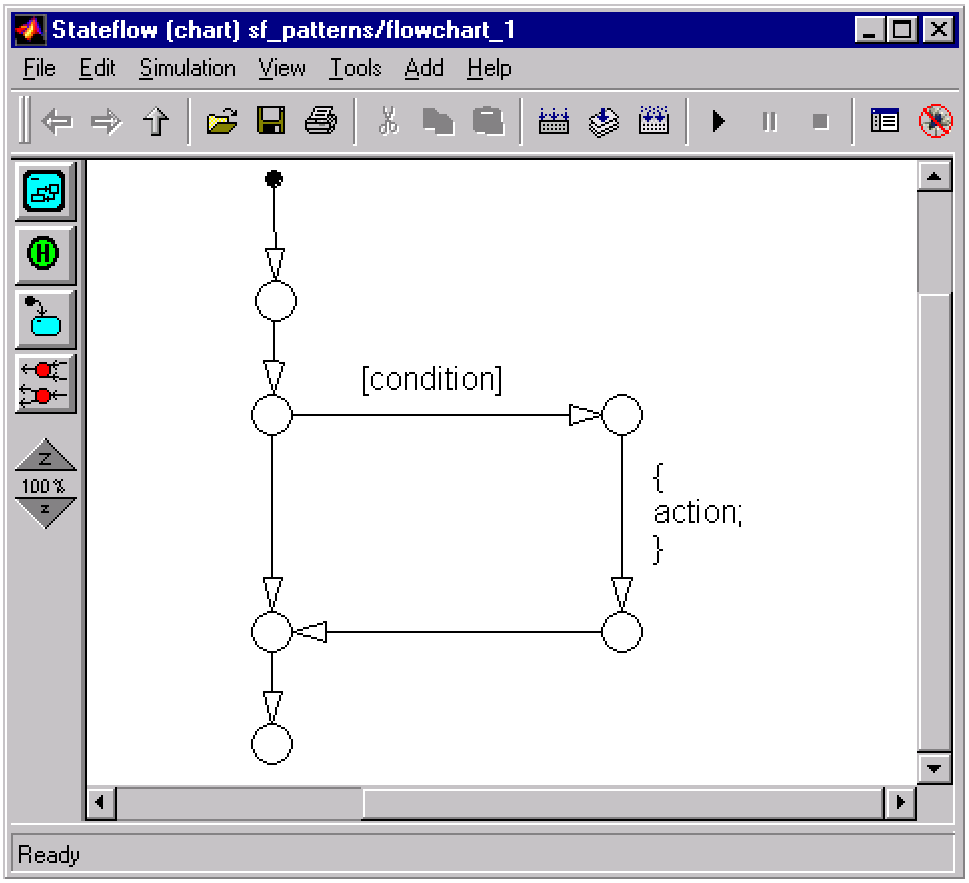
ID: Title	**0051: Scope of internal signals and local auxiliary variables**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	Internal signals and local auxiliary variables are "Local data" in Stateflow: All local data of a Stateflow-block must be defined on the chart level or below the Object Hierarchy. There is no local data on the machine level (i.e. there is no interaction between local data in different charts). Correct:  Exploring... model/chart.state File Edit Tools Add Help Object Hierarchy model chart state Contents of: (state) model/chart.state Name Scope Trigger Type Size Min Max InitVal FrWS ToWS Watch data Local double 0 events(0) data(1) targets(0) 1 [1:1] Incorrect:  Exploring... model File Edit Tools Add Help Object Hierarchy model chart state Contents of: (machine) model Name Scope Trigger Type Size Min Max InitVal FrWS ToWS Watch data Local double 0 sfun events(0) data(1) targets(1) 2 [1:2]
Benefit	Respecting the guideline ensures ... accessible local data in Stateflow. a clear system structure. reusable models. readable models.

	testable systems.
Penalty	Breaking the guideline ... may cause errors in functionality, e.g. auto-c-code generation. may cause problems or errors with custom tools. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

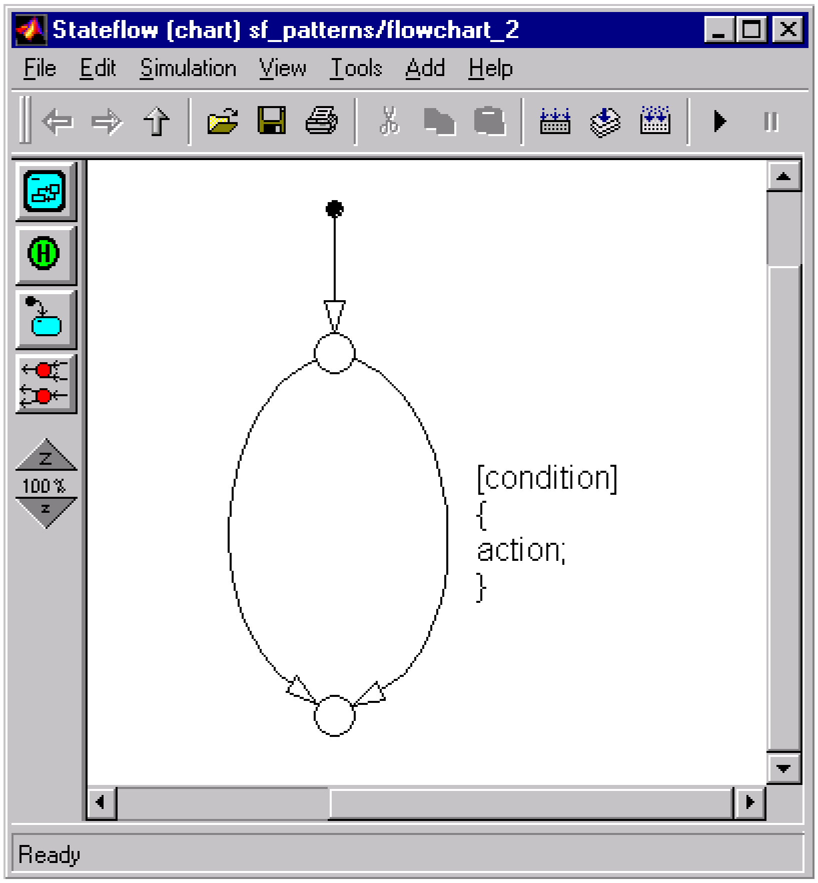
5.5 Flowcharts

5.5.1 General

Stateflow Flowchart:


Stateflow (chart) sf_patterns/flowchart_1
File Edit Simulation View Tools Add Help
100 %
[condition]
{
action;
}
Ready

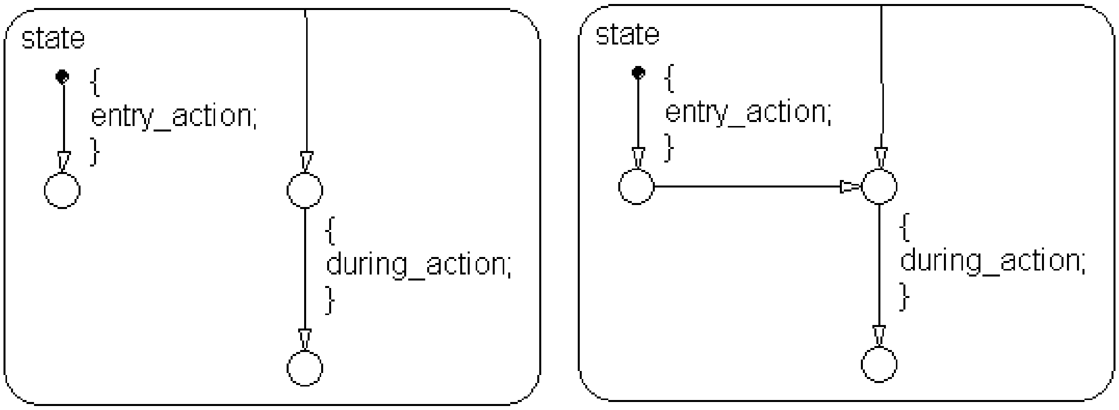
OR



Stateflow (chart) sf_patterns/flowchart_2
File Edit Simulation View Tools Add Help
100 %
[condition]
{
action;
}
Ready

For Stateflow blocks with a Flowchart, see Appendix A.Appendix A is a glossary that compares straight line Flowcharts to curved line Flowcharts.PROJECT Guidelines supports both styles; however, one style must be chosen for consistency.

States with Flowcharts:

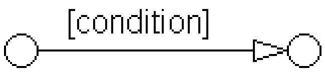


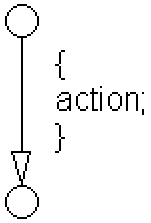
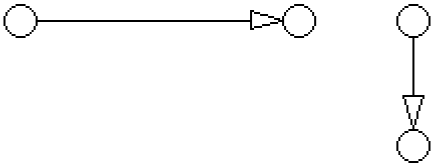
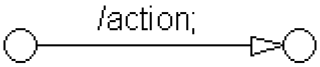
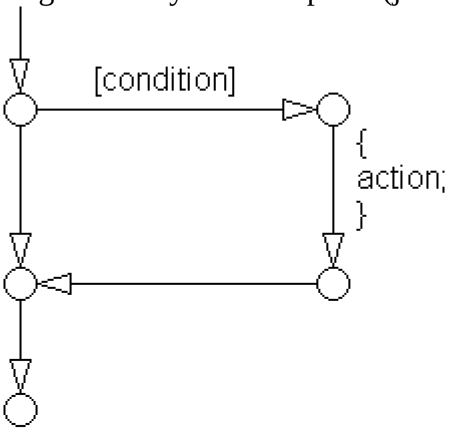
5.5.1.1 0052: Use of patterns for Flowcharts

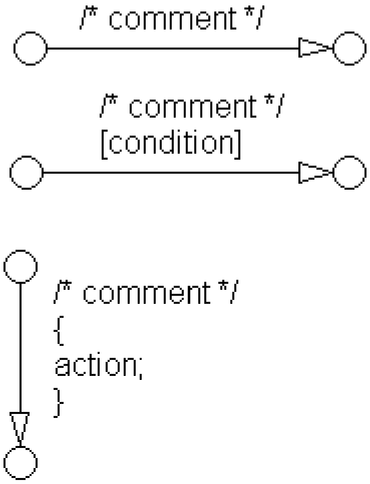
ID: Title	**0052: Use of patterns for Flowcharts**
Priority	strongly recommended
Scope	PROJECT

Automation	possible
Prerequisites	
Description	A Flowchart is built with the help of Flowchart patterns (e.g. IF THEN ELSE, FOR LOOP, ...): The data flow is oriented from the top to the bottom. Patterns are connected with empty transitions.
Benefit	Respecting the guideline ensures ... a clear system structure. reusable models. readable models. testable systems.
Penalty	Breaking the guideline ... may cause errors in auto-c-code. may cause problems or errors with custom tools. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

5.5.1.2 0053: Transitions in Flowcharts

ID: Title	**0053: Transitions in Flowcharts**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	
Description	The following rules apply to transitions in Flowcharts: A transition in a Flowchart has either a condition, a condition action or an empty transition. Transition with condition: 

	Transition with condition action:  Empty transition:  Transition actions are not used in Flowcharts.Transitions Actions are only valid when used in transitions between state machines, otherwise they are not activated because of the inherent dependency on a valid state to state transition to activate them. Transition action:  At every junction, except for the last junction of a flow diagram, exactly one unconditional transition begins. Every decision point (junction) must have a default path  A transition may have a comment:

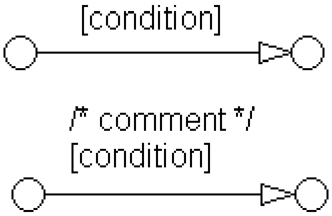
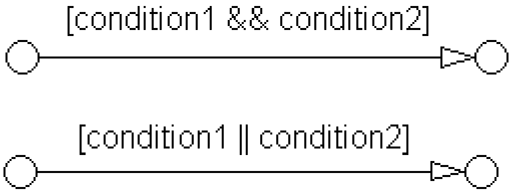
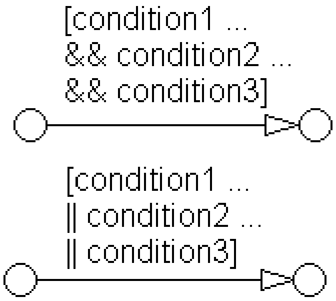
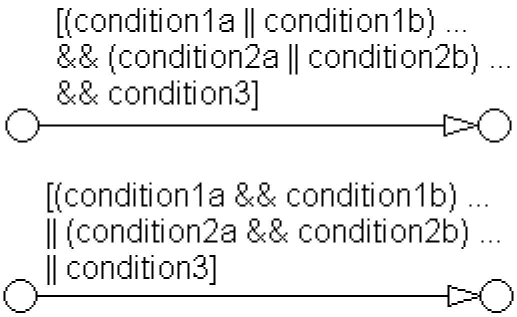
	 /* comment */
	/* comment */
	[condition]
	/* comment */
	{
	action;
	}

Benefit	Respecting the guideline ensures ... a clear system structure. ensures a valid path from top to bottom of a Flowchart reusable models. readable models. testable systems.
Penalty	Breaking the guideline ... may cause errors in auto-c-code. may cause problems or errors with custom tools. may cause a lot of redesign work.
Author	
Last Change	18.08.2003

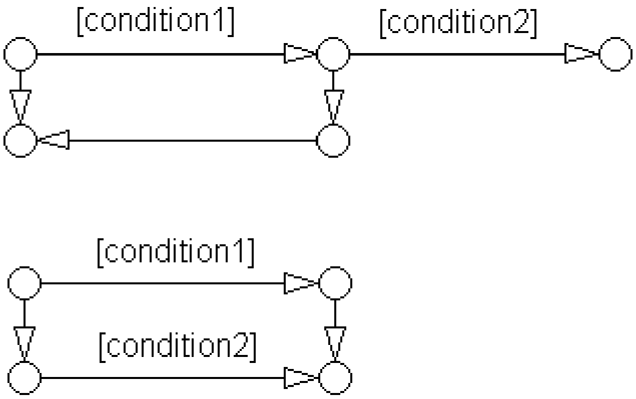
5.5.2 Patterns

5.5.2.1 0054: Flowchart patterns for conditions

ID: Title	**0054: Flowchart patterns for conditions**
Priority	strongly recommended
Scope	PROJECT
Automation	possible

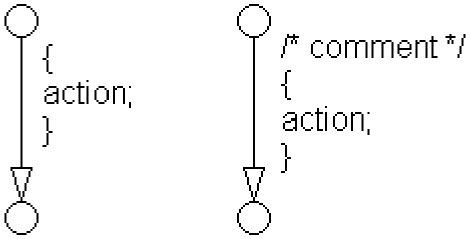
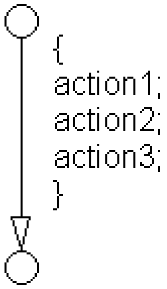
<table>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for conditions within Stateflow Flowcharts:
<table>
<tr><th>5.5.2.1.1.1 Equivalent Functionality</th><th>Flowchart pattern</th></tr>
<tr><td>ONE CONDITION:
[condition]</td><td>
</td></tr>
<tr><td>UP TO THREE CONDITIONS, SHORT FORM:
[condition1 && condition2]
[condition1 || condition2]</td><td>
</td></tr>
<tr><td>TWO OR MORE CONDITIONS, MULTILINE FORM:
(The use of different operators in this form is not allowed, use subconditions instead!)
[condition1 ...
&& condition2 ...
&& condition3]
[condition1 ...
|| condition2 ...
|| condition3]</td><td>
</td></tr>
<tr><td>CONDITIONS WITH SUBCONDITIONS:
(The use of different operators to connect subconditions is not allowed! The use of brackets is mandatory!)
[(condition1a || condition1b) ...
&& (condition2a || condition2b) ...
&& (condition3)]
[(condition1a && condition1b) ...
|| (condition2a && condition2b) ...
|| (condition3)]</td><td>
[(condition1a || condition1b) ...
&& (condition2a || condition2b) ...
&& condition3]
[(condition1a && condition1b) ...
|| (condition2a && condition2b) ...
|| condition3]

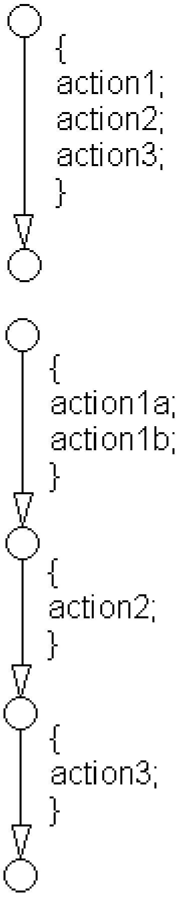
</td></tr>
</table></td></tr>
</table>

	CONDITIONS, WHICH ARE VISUALLY SEPARATED: (This form can be mixed up with the patterns listed above!) [condition1 && condition2] [condition1 || condition2]  [condition1] [condition2] [condition1] [condition2]
Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.
Penalty	Breaking the guideline ... may cause errors in functionality. may cause problems or errors with custom tools. may cause a lot of redesign work. may cause confusing presentations. may cause problems with system integration.
Author	
Last Change	18.08.2003

5.5.2.2 0055: Flowchart patterns for condition actions

ID: Title	**0055: Flowchart patterns for condition actions**
Priority	strongly recommended
Scope	PROJECT
Automation	possible
Prerequisites	

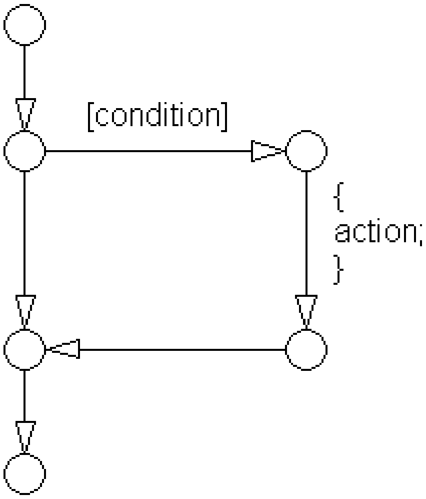
<table>
<tr><td rowspan="4">Description</td><td colspan="2">The following patterns are used for condition actions within Stateflow Flowcharts:</td></tr>
<tr><td><b>Equivalent Functionality</b></td><td><b>Flowchart pattern</b></td></tr>
<tr><td>ONE CONDITION ACTION:
action;</td><td>
</td></tr>
<tr><td>TWO OR MORE CONDITION ACTIONS, MULTILINE FORM:
(Two or more condition actions in one line are not allowed!)
action1; ...
action2; ...
action3; ...</td><td>
</td></tr>
</table>

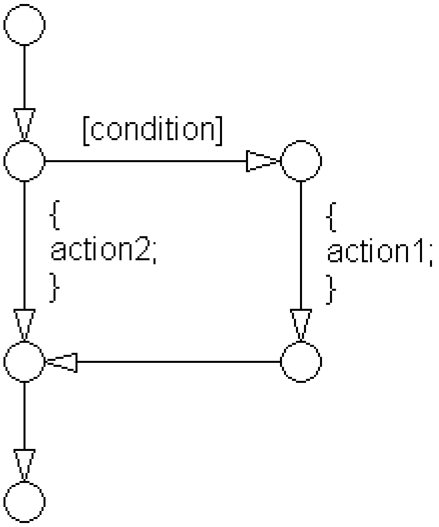
	CONDITION ACTIONS, WHICH ARE VISUALLY SEPARATED:
	(This form can be mixed up with the patterns listed above!)
	action1a;
	action1b;
	action2;
	action3;
		 {
		action1;
		action2;
		action3;
		}
		{
		action1a;
		action1b;
		}
		{
		action2;
		}
		{
		action3;
		}

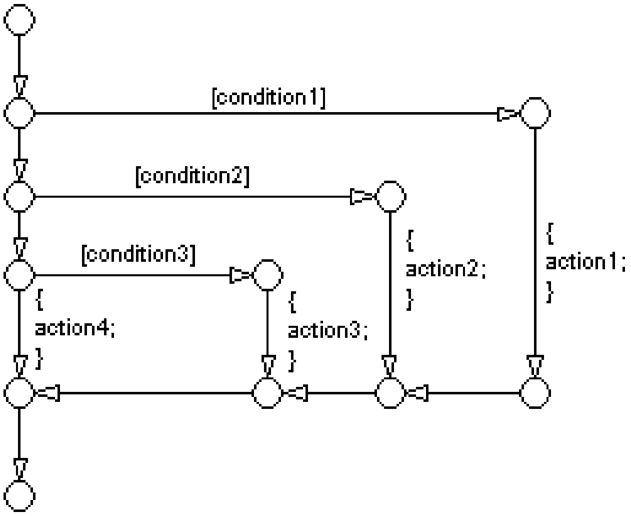
Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.	
Penalty	Breaking the guideline ... may cause errors in functionality. may cause problems or errors with custom tools. may cause a lot of redesign work. may cause confusing presentations.	

	may cause problems with system integration.
Author	
Last Change	18.08.2003

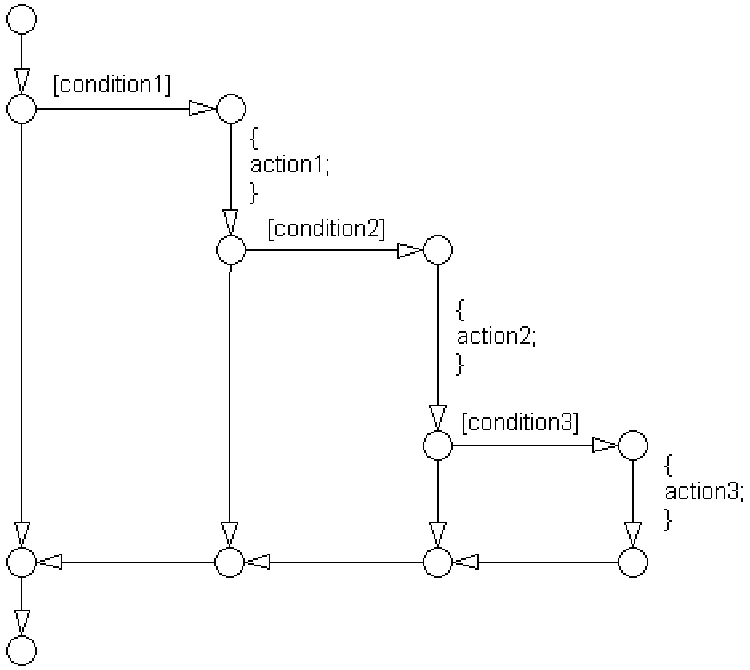
5.5.2.3 0056: Flowchart patterns for If-then-else-if constructs

<table>
<tr><td><b>ID: Title</b></td><td><b>0056: Flowchart patterns for If-then-else-if constructs</b></td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td>0048
0049</td></tr>
<tr><td>Description</td><td>The following patterns are used for If-then-else-if constructs within Stateflow Flowcharts:
<table>
<tr><td><b>Equivalent Functionality</b></td><td><b>Flowchart pattern</b></td></tr>
<tr><td>IF THEN
if (condition) {
action;
}</td><td>
</td></tr>
</table>
</td></tr>
</table>

	IF THEN ELSE
	if (condition) {
	action1;
	}
	else {
	action2;
	}
		

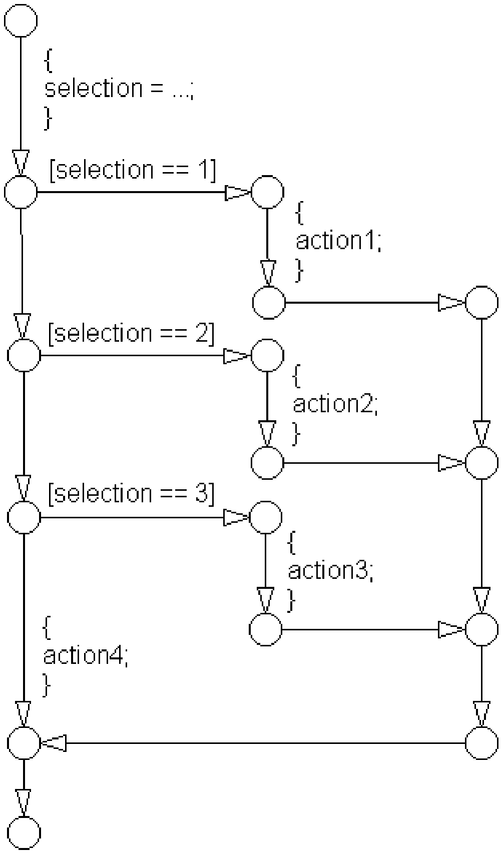
	IF THEN ELSE IF
	if (condition1) {
	action1;
	}
	else if (condition2) {
	action2;
	}
	else if (condition3) {
	action3;
	}
	else {
	action4;
	}
		 [condition1]
		[condition2]
		[condition3]
		{
		action4;
		}
		{
		action3;
		}
		{
		action2;
		}
		{
		action1;
		}

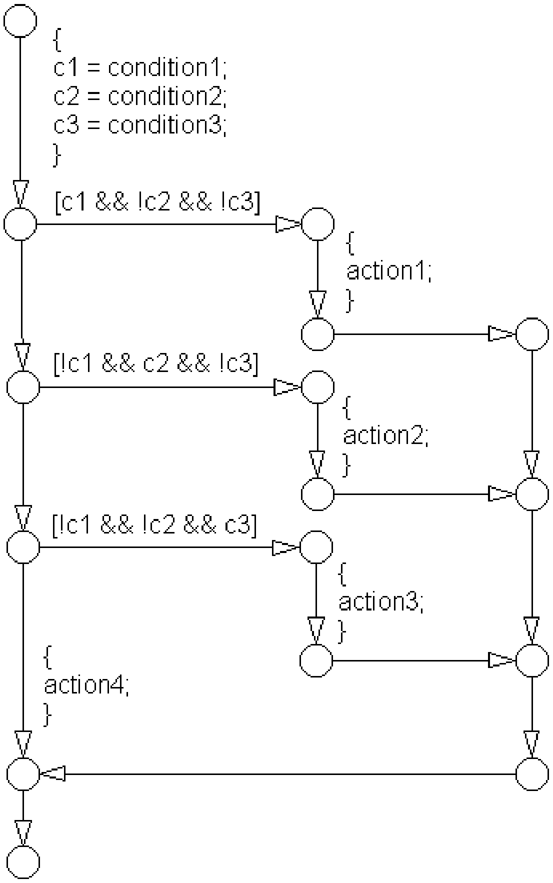
	Cascade of IF THEN
	if (condition1) {
	action1;
	if (condition2) {
	action2;
	if (condition3) {
	action3;
	}
	}
	}

	 [condition1]
	{
	action1;
	}
	[condition2]
	{
	action2;
	}
	[condition3]
	{
	action3;
	}

Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.
Penalty	Breaking the guideline ... may cause errors in functionality. may cause problems or errors with custom tools. may cause a lot of redesign work. may cause confusing presentations. may cause problems with system integration.
Author	
Last Change	18.08.2003

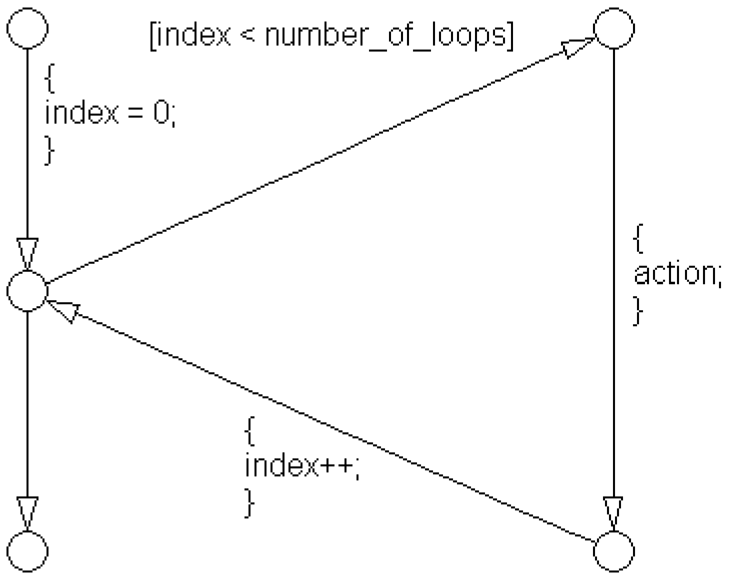
5.5.2.4 0057: Flowchart patterns for case constructs

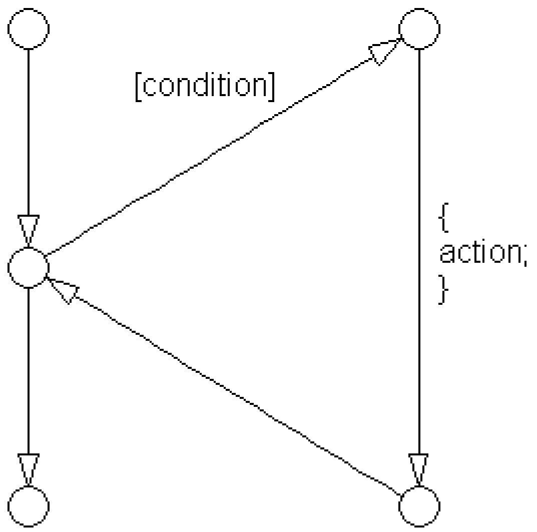
<table>
<tr><td><b>ID: Title</b></td><td><b>0057: Flowchart patterns for case constructs</b></td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td>0048
0049</td></tr>
<tr><td>Description</td><td>The following patterns are used for If-then-else-if constructs within Stateflow Flowcharts:
<table>
<tr><td><b>Equivalent Functionality</b></td><td><b>Flowchart pattern</b></td></tr>
<tr><td>CASE with exclusive selection
selection = ...;
switch (selection) {
case 1:
action1;
break;
case 2:
action2;
break;
case 3:
action3;
break;
default:
action4;
}</td><td>
</td></tr>
</table></td></tr>
</table>

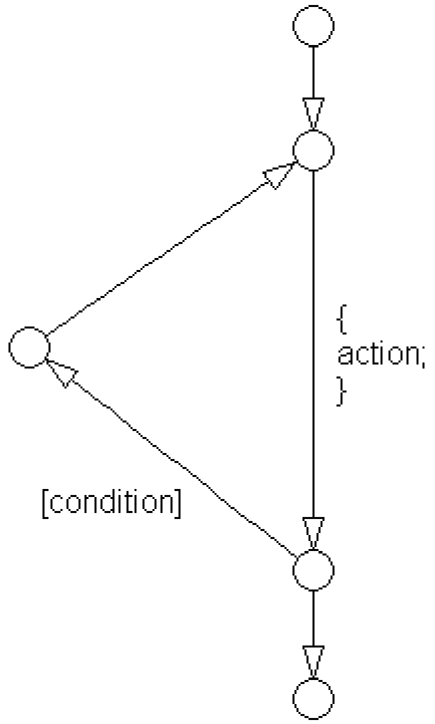
<table>
<tr><td></td><td>CASE with exclusive conditions
c1 = condition1;
c2 = condition2;
c3 = condition3;
if (c1 && !c2 && !c3) {
action1;
}
elseif (!c1 && c2 && !c3) {
action2;
}
elseif (!c1 && !c2 && c3) {
action3;
}
else {
action4;
}</td><td>
</td></tr>
<tr><td>Benefit</td><td colspan="2">Respecting the guideline ensures ...
uniform appearance of models, code and documentation.
reusable models.
readable models.
a clear system structure.</td></tr>
<tr><td>Penalty</td><td colspan="2">Breaking the guideline ...
may cause errors in functionality.
may cause problems or errors with custom tools.
may cause a lot of redesign work.
may cause confusing presentations.
may cause problems with system integration.</td></tr>
</table>

Last Change	18.08.2003

5.5.2.5 0058: Flowchart patterns For Loops constructs

<table>
<tr><td><b>ID: Title</b></td><td><b>0058: Flowchart patterns For Loops constructs</b></td></tr>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td>0048
0049</td></tr>
<tr><td>Description</td><td>The following patterns are used for For Loops within Stateflow Flowcharts:
<table>
<tr><th>Equivalent Functionality</th><th>Flowchart pattern</th></tr>
<tr><td>FOR LOOP
for (index=0;
index<number_of_loops;
index++) {
action;
}</td><td>
</td></tr>
</table>
</td></tr>
</table>

	WHILE LOOP
	while (condition) {
	action;
	}
		

	DO WHILE LOOP
	do {
	action;
	}
	while (condition);
		

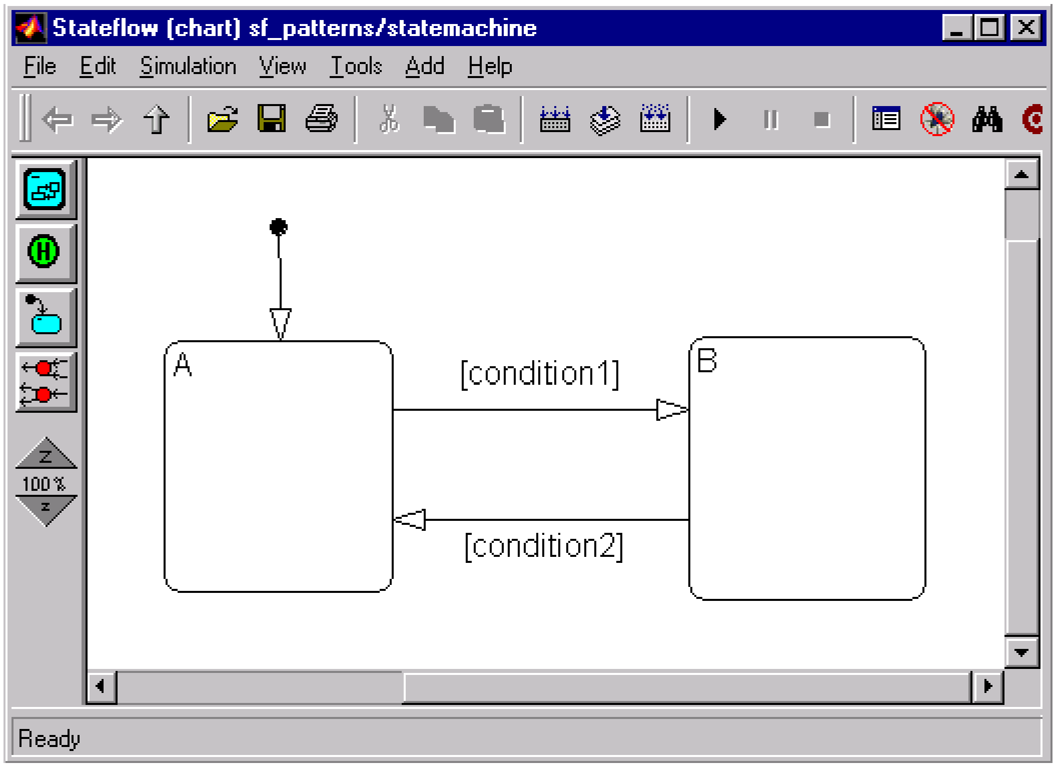
Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.	

Penalty	Breaking the guideline ... may cause errors in functionality. may cause problems or errors with custom tools. may cause a lot of redesign work. may cause confusing presentations. may cause problems with system integration.
Author	
Last Change	18.08.2003

5.6 Statecharts

5.6.1 General

Stateflow-block with statemachine:

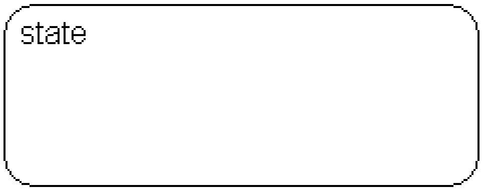


5.6.1.1 0059: States in statemachines

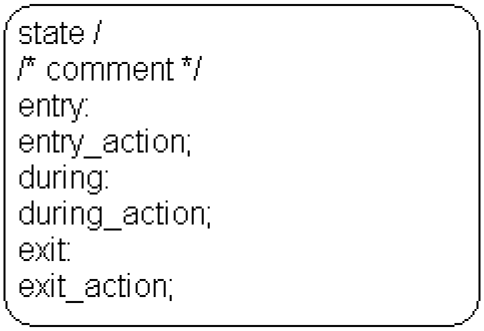
ID: Title	**0059: States in statemachines**
Priority	mandatory
Scope	PROJECT
Automation	possible
Prerequisites	0049: Flowchart patterns for condition actions
Description	In statemachines ... there are at least two exclusive states. a state has either zero or more than one substates. the initial state of a hierarchical level with exclusive states is clearly defined by a default transition.

a state may be grouped or subcharted.

Empty state:

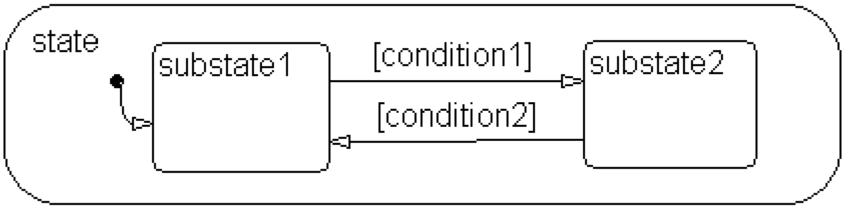


State with actions (Every line apart from the state name is optional!):

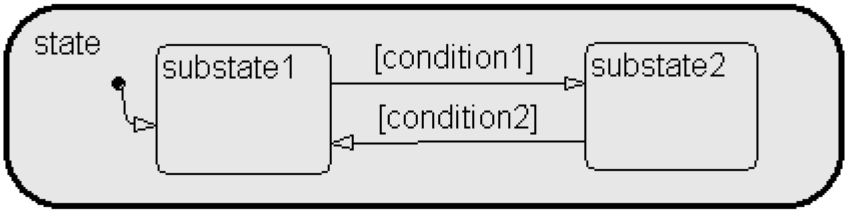


```
state /
/* comment */
entry:
entry_action;
during:
during_action;
exit:
exit_action;
```

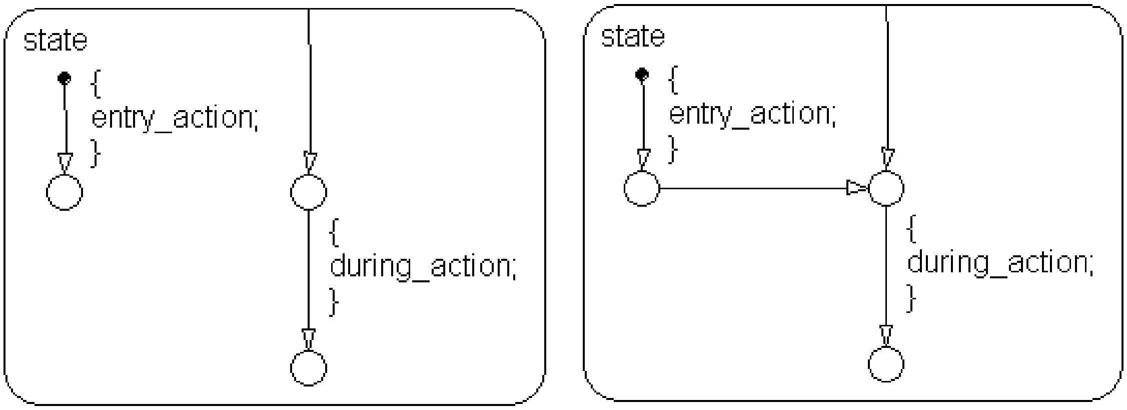
State with substates:



Grouped state with substates:



States with Flowcharts:

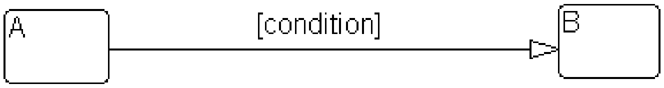
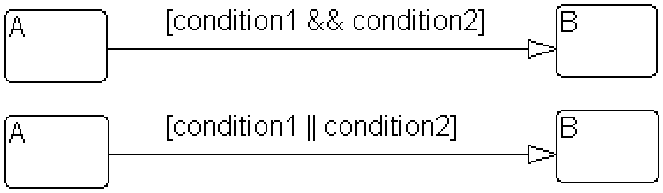
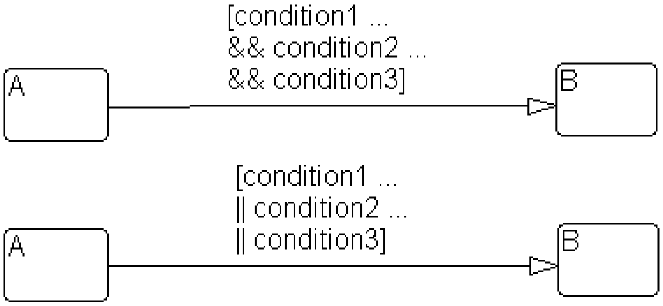
	 state
	{
	entry_action;
	}
	{
	during_action;
	}
	state
	{
	entry_action;
	}
	{
	during_action;
	}
	Note:
	In the second example, the during actions are executed the first time right after the entry actions.
	A state with a Flowchart has only a name, but no actions in the state-label.

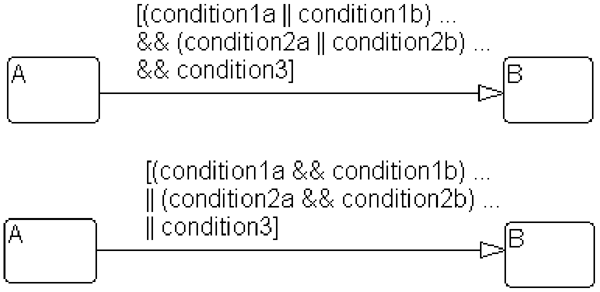
Benefit	Respecting the guideline ensures ... a clear system structure. reusable models. readable models. testable systems.
Penalty	Breaking the guideline ... may cause errors in auto-c-code. may cause problems or errors with custom tools. may cause a lot of redesign work.
Last Change	19.09.2008,

5.6.2 Patterns

5.6.2.1 0060: Statemachine patterns for conditions

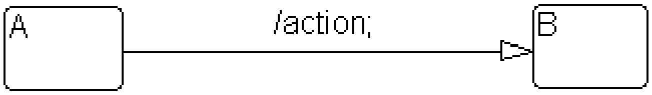
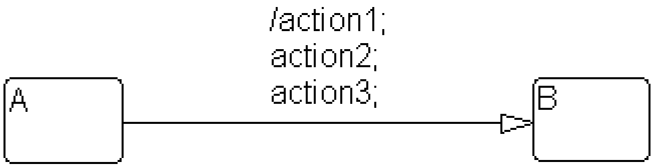
ID: Title	**0060: Statemachine patterns for conditions**
Priority	strongly recommended
Scope	PROJECT
Automation	possible

<table>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for conditions within Stateflow statemachines:
<table>
<tr><th>Equivalent Functionality</th><th>Statemachine pattern</th></tr>
<tr><td>ONE CONDITION:
(condition)</td><td>
</td></tr>
<tr><td>UP TO THREE CONDITIONS, SHORT FORM:
(condition1 && condition2)
(condition1 || condition2)</td><td>
</td></tr>
<tr><td>TWO OR MORE CONDITIONS, MULTILINE FORM:
(The use of different operators in this form is not allowed, use subconditions instead!)
(condition1 ...
&& condition2 ...
&& condition3)
(condition1 ...
|| condition2 ...
|| condition3)</td><td>
</td></tr>
</table></td></tr>
</table>

	CONDITIONS WITH SUBCONDITIONS: (The use of different operators to connect subconditions is not allowed! The use of brackets is mandatory!) ((condition1a || condition1b) ... && (condition2a || condition2b) ... && (condition3)) ((condition1a && condition1b) ... || (condition2a && condition2b) ... || (condition3))  [(condition1a || condition1b) ... && (condition2a || condition2b) ... && condition3] A B [(condition1a && condition1b) ... || (condition2a && condition2b) ... || condition3] A B
Benefit	Respecting the guideline ensures ... uniform appearance of models, code and documentation. reusable models. readable models. a clear system structure.
Penalty	Breaking the guideline ... may cause errors in functionality. may cause problems or errors with custom tools. may cause a lot of redesign work. may cause confusing presentations. may cause problems with system integration.
Author	
Last Change	18.08.2003

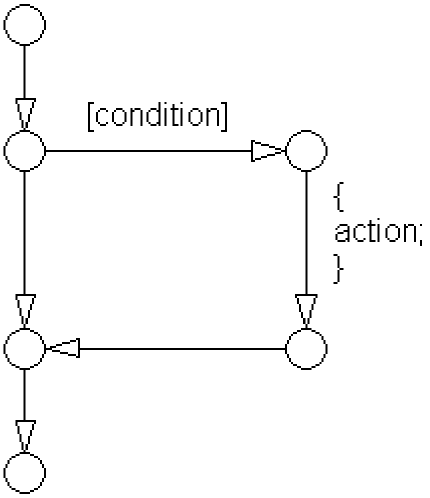
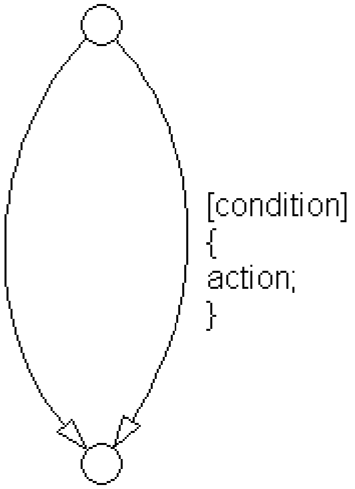
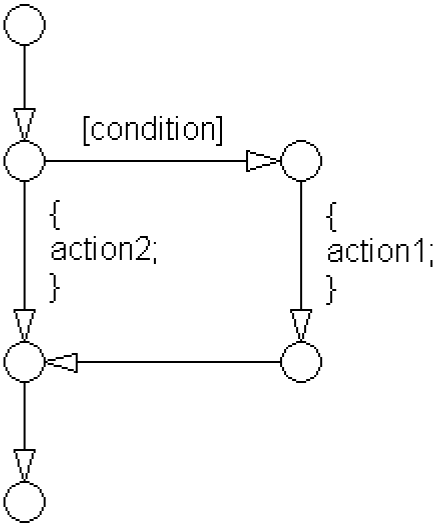
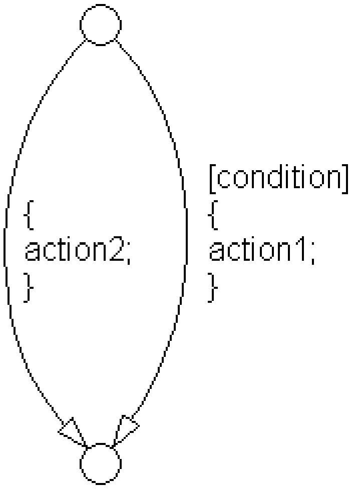
5.6.2.2 0061: Statemachine patterns for transition actions

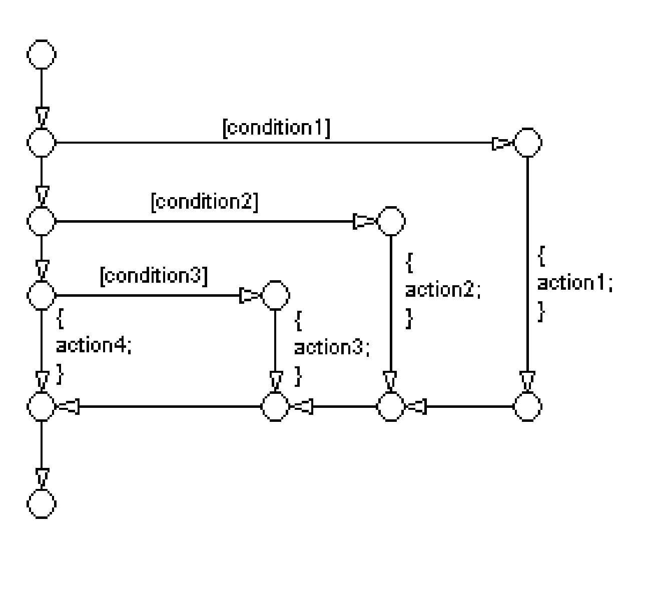
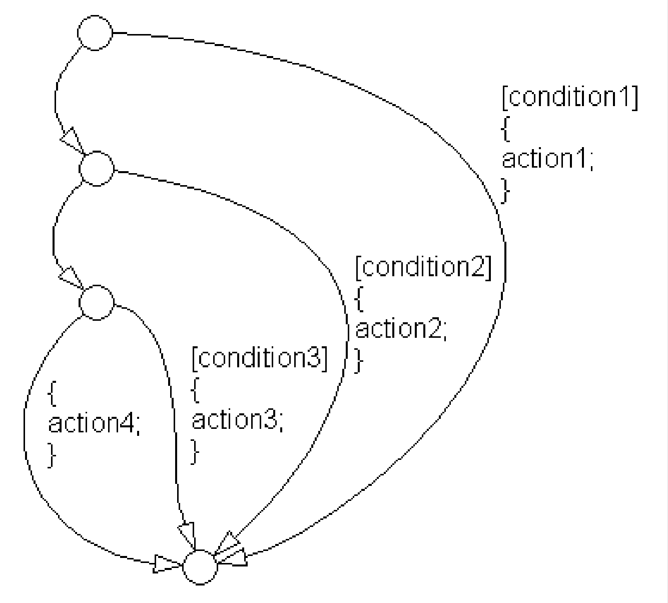
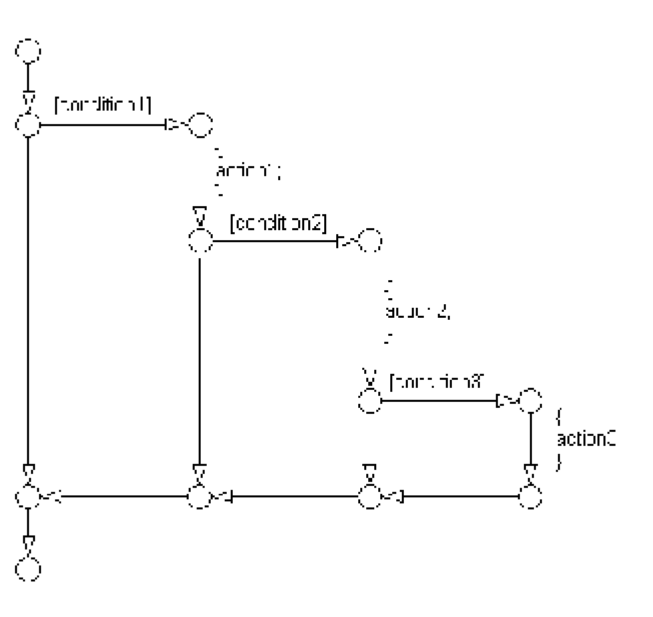
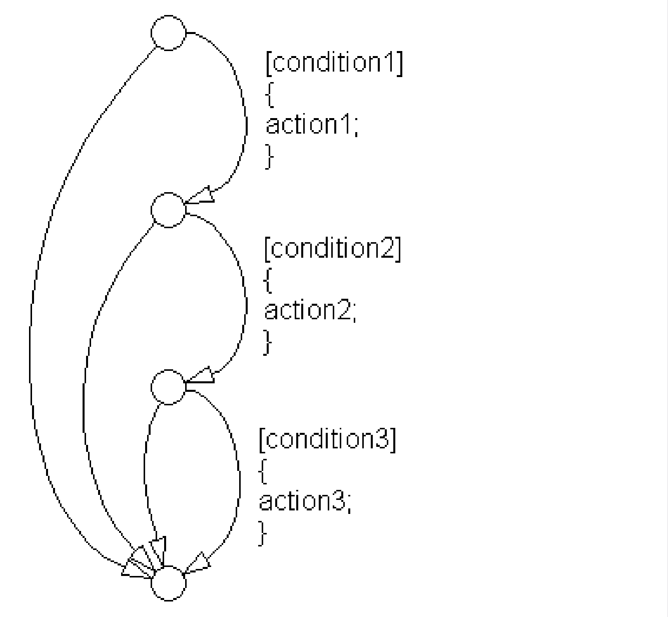
ID: Title	**0061: Statemachine patterns for transition actions**

<table>
<tr><td>Priority</td><td>strongly recommended</td></tr>
<tr><td>Scope</td><td>PROJECT</td></tr>
<tr><td>Automation</td><td>possible</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns are used for transition actions within Stateflow statemachines:
<table>
<tr><th>Equivalent Functionality</th><th>Statemachine pattern</th></tr>
<tr><td>ONE TRANSITION ACTION:
action;</td><td>
</td></tr>
<tr><td>TWO OR MORE TRANSITION ACTIONS, MULTILINE FORM:
(Two or more transition actions in one line are not allowed!)
action1;
action2;
action3;</td><td>
</td></tr>
</table></td></tr>
<tr><td>Benefit</td><td>Respecting the guideline ensures ...
uniform appearance of models, code and documentation.
reusable models.
readable models.
a clear system structure.</td></tr>
<tr><td>Penalty</td><td>Breaking the guideline ...
may cause errors in functionality.
may cause problems or errors with custom tools.
may cause a lot of redesign work.
may cause confusing presentations.
may cause problems with system integration.</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Last Change</td><td>18.08.2003</td></tr>
</table>

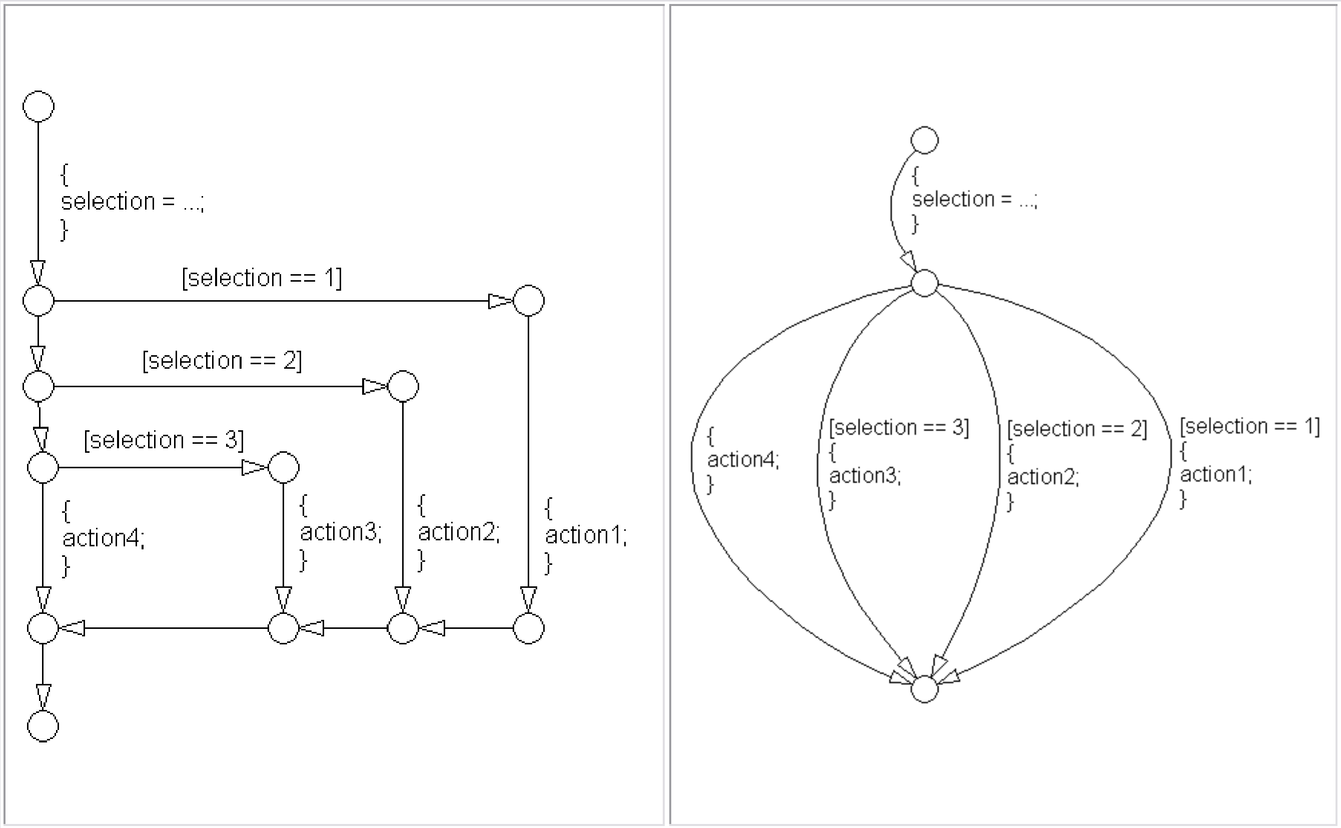
6 FLOWCHART REFERENCE

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

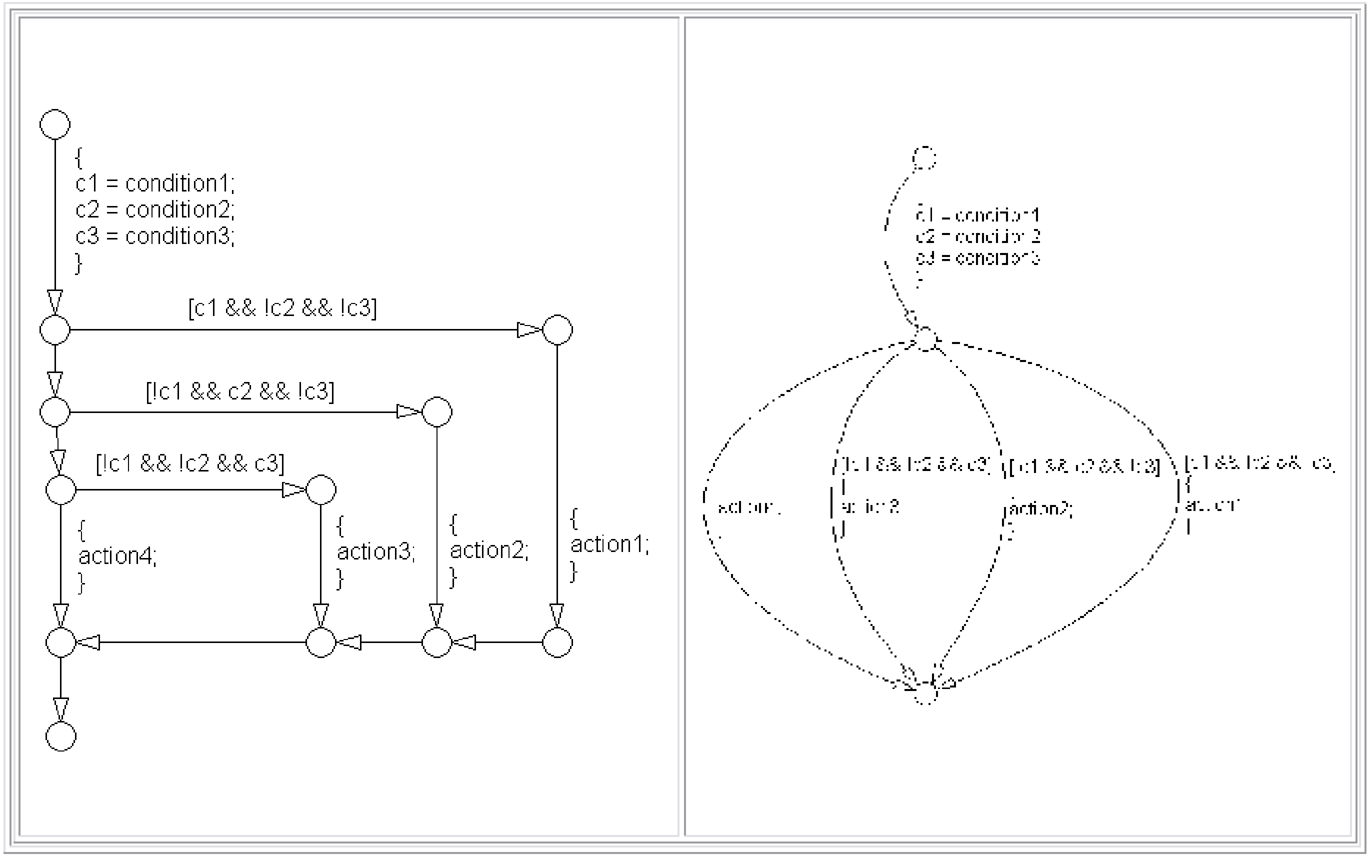
Straight Line Flow Chart Pattern	**Curved Line Flow Chart Pattern**
IF THEN	
	
IF THEN ELSE	
	
IF THEN ELSE IF	

<table>
<tr>
<td> </td>
<td> </td>
</tr>
<tr>
<td colspan="2">Cascade of IF THEN</td>
</tr>
<tr>
<td> </td>
<td> </td>
</tr>
</table>

<table>
<tr>
<td colspan="2">The following patterns are used for case constructs within Stateflow Flowcharts:</td>
</tr>
<tr>
<td><strong>Straight Line Flow Chart Pattern</strong></td>
<td><strong>Curved Line Flow Chart Pattern</strong></td>
</tr>
<tr>
<td colspan="2">CASE with exclusive selection</td>
</tr>
</table>

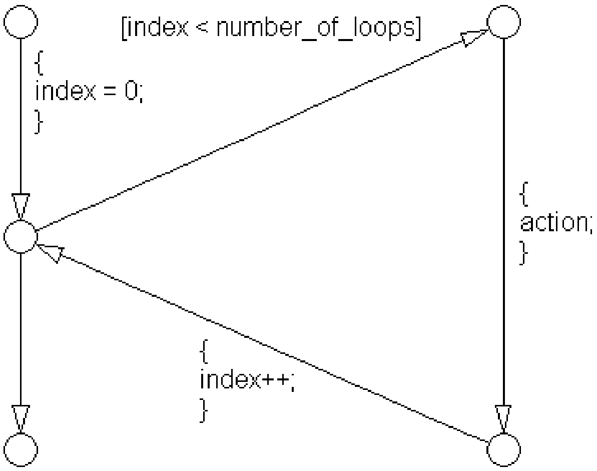
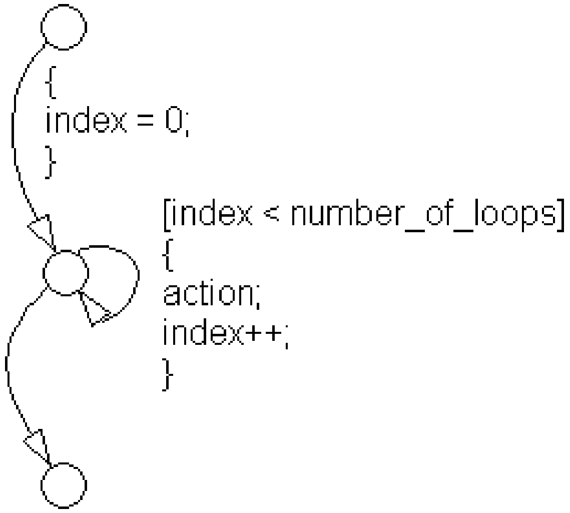


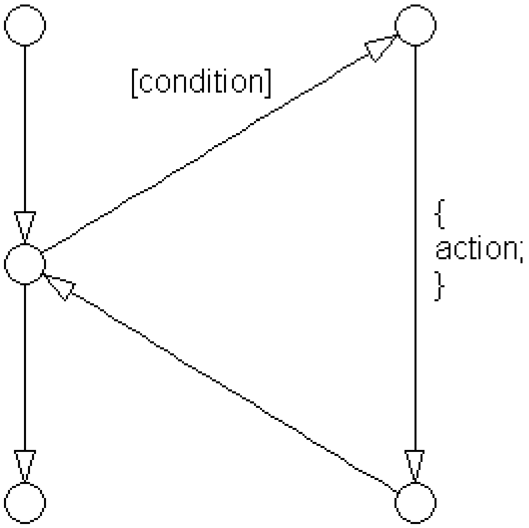
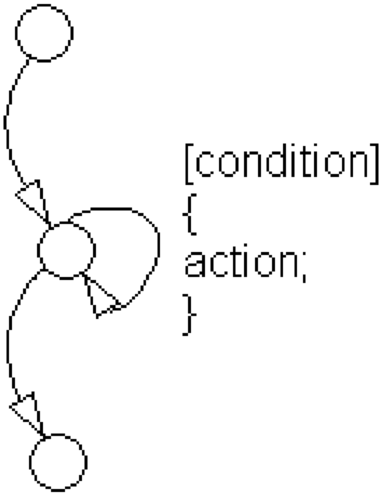
CASE with exclusive conditions

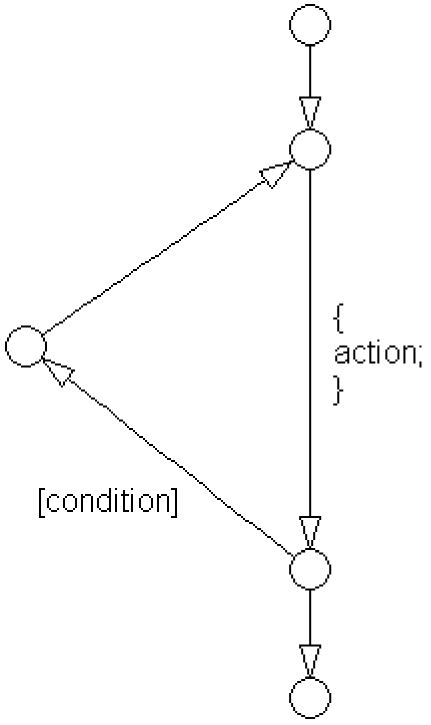
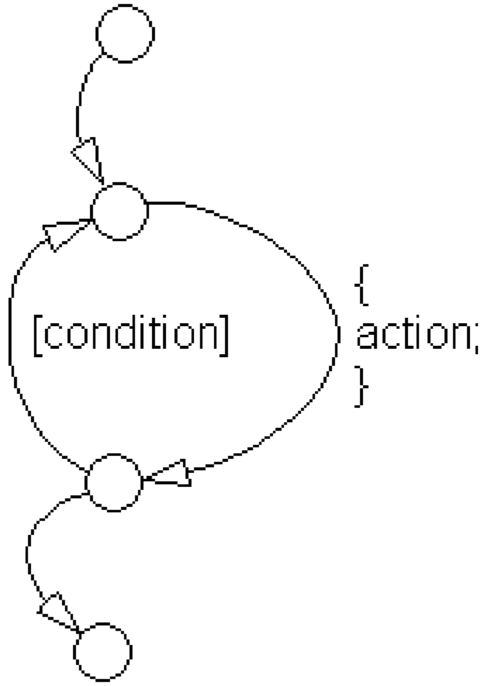

{
c1 = condition1;
c2 = condition2;
c3 = condition3;
}
[c1 && !c2 && !c3]
[!c1 && c2 && !c3]
[!c1 && !c2 && c3]
{
action4;
}
{
action3;
}
{
action2;
}
{
action1;
}
c1 = condition1
c2 = condition2
c3 = condition3
[!c1 && !c2 && c3]
[!c1 && c2 && !c3]
[c1 && !c2 && !c3]
action4;
action3
action2;
action1

The following patterns are used for For Loops within Stateflow Flowcharts:

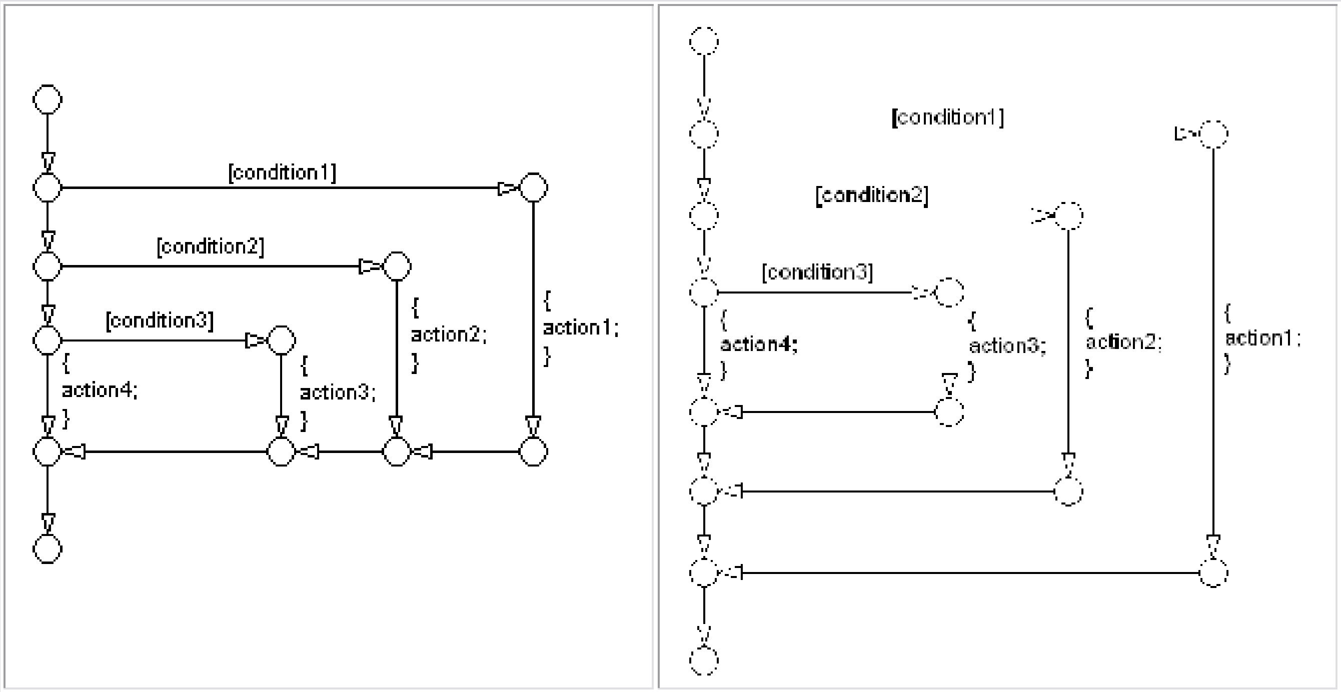
Straight Line Flow Chart Pattern	**Curved Line Flow Chart Pattern**
FOR LOOP	


 [index < number_of_loops]
{
index = 0;
}
{
action;
}
{
index++;
}
	
	 {
	index = 0;
	}
	[index < number_of_loops]
	{
	action;
	index++;
	}

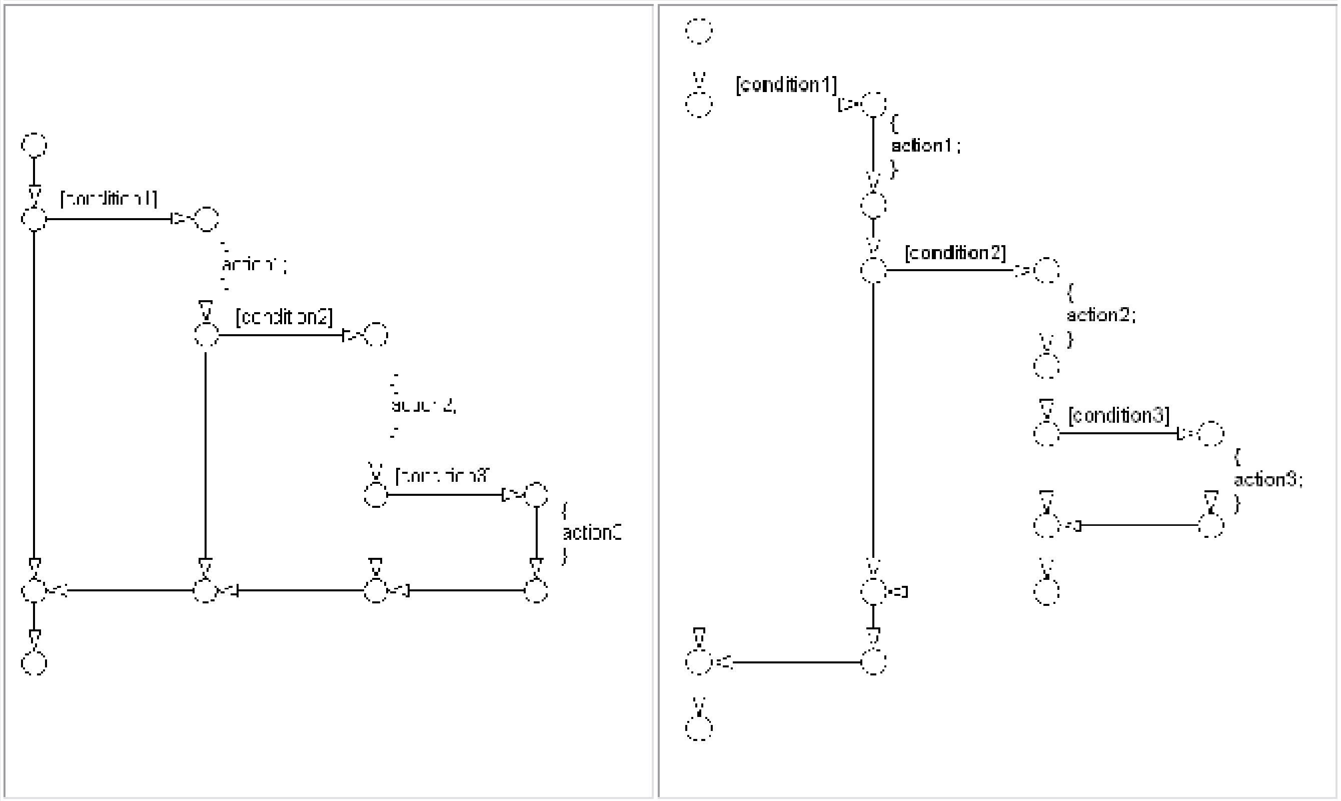
WHILE LOOP	
 	
DO WHILE LOOP	

The following patterns are alternately used for If-then-else-if constructs within Stateflow Flowcharts:	
Straight Line Flow Chart Pattern	**Alternate Straight Line Flow Chart Pattern**
IF THEN ELSE IF	



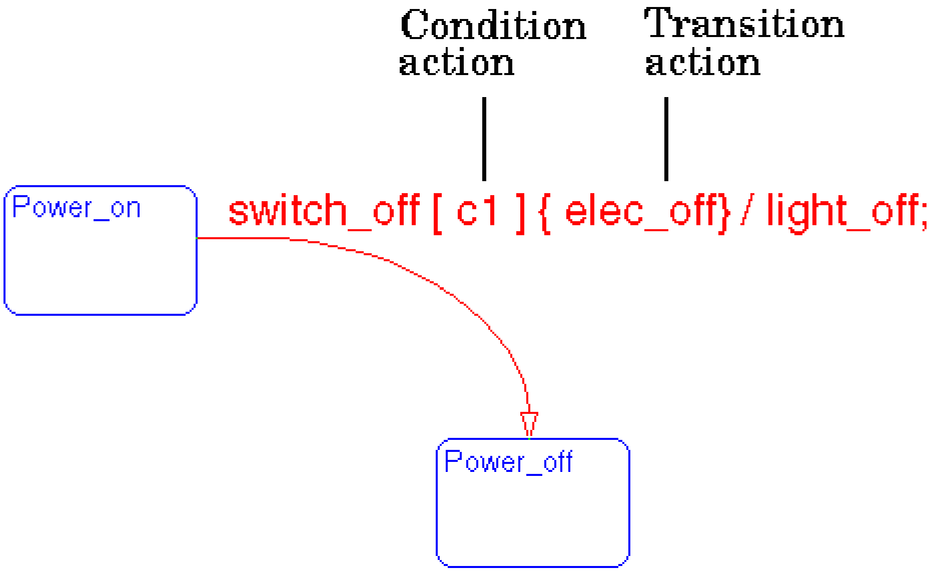
Cascade of IF THEN



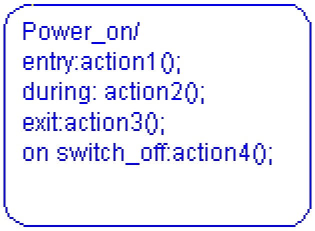
7 GLOSSARY

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,



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



```
Power_on/
entry:action1();
during: action2();
exit: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

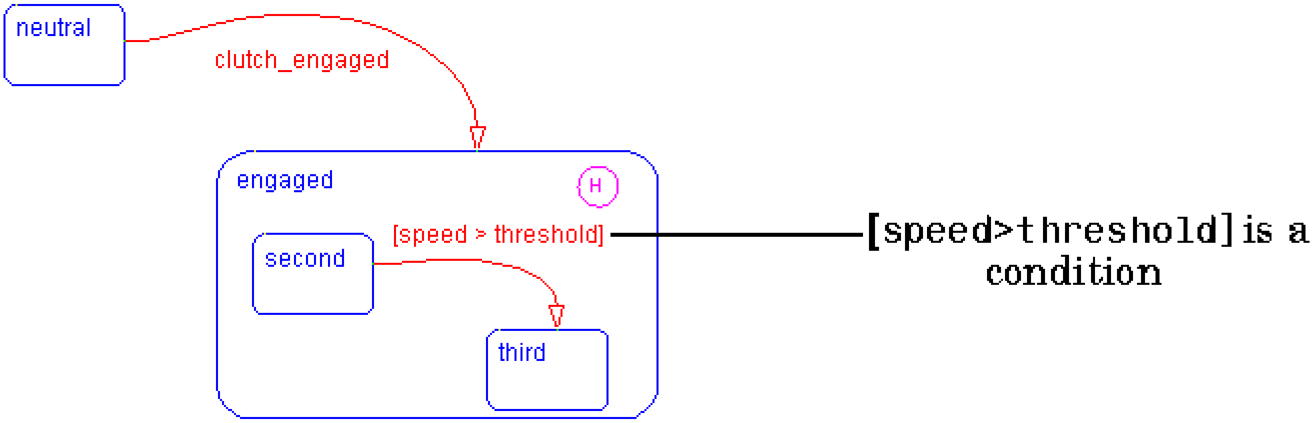
on the activity status of a state. Transitions can have condition actions and transition actions. States can have `entry`, `during`, `exit`, and, `on` *event_name* actions.
An action can be a function call, an event to be broadcast, a variable to be assigned a value, etc. The *action language* defines 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 occurs given that the specified expression is true. For example,

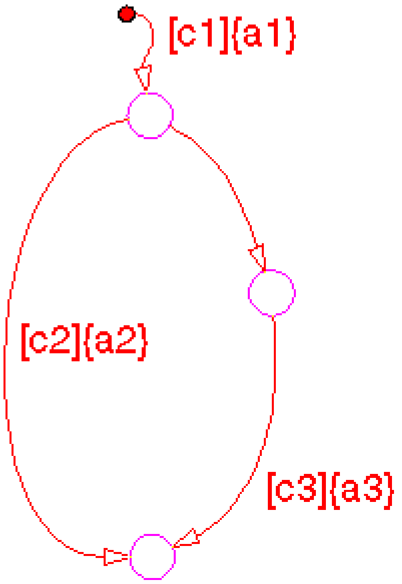


[speed>threshold]is a condition

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

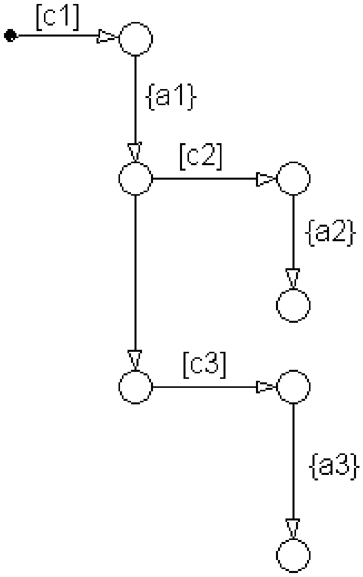
Connective Junction

Connective junctions are decision points in the system. A connective junction is a graphical object that simplifies Stateflow diagram representations and facilitates generation of efficient code. Connective junctions provide alternative ways to represent desired system behavior.
This example shows how connective junctions (displayed as small circles) are used to represent the flow of an `if` code structure.



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

Or the equivalent squared style



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

Name	**Button Icon**	**Description**
Connective junction		One use of a Connective junction is to handle situations where transitions out of one state into two or

		more states are taken based on the same event but guarded by different conditions.

Data

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

Defining Data

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

Data Dictionary

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

Decomposition

A state has *decomposition* when it consists of one or more substates. A Stateflow diagram that contains at least one state also has decomposition. Representing hierarchy necessitates some rules around how states can be grouped in the hierarchy. A superstate has either parallel (AND) or exclusive (OR) decomposition. All substates at a particular level in the hierarchy must be of the same decomposition.
Parallel (AND) State Decomposition. Parallel (AND) state decomposition is indicated when states have dashed borders. This representation is appropriate if all states at that same level in the hierarchy are active at the same time. The activity within parallel states is essentially independent.
Exclusive (OR) State Decomposition. Exclusive (OR) state decomposition is represented by states with solid borders. Exclusive (OR) decomposition is used to describe system modes that are mutually exclusive. Only one state, at the same level in the hierarchy, can be active at a time.

Default Transition

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

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

Events

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

Finite State Machine

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

Flow Graph

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

Flowchart (also known as Flow Path)

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

Flow Subgraph

A f*low subgraph* is the set of Flowcharts that start on the same transition segment.

Hierarchy

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

History Junction

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

Name	**Button Icon**	**Description**
History Junction	H	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 RTW 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

Any block that performs some algorithm, such as a Gain block.

Notation

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

A set of graphical objects

A set of nongraphical text-based objects

Defined relationships between those objects

Parallelism

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

Real-Time Workshop®

The Real-Time Workshop is an automatic C language code generator for Simulink. It produces C code directly from Simulink block diagram models. This automatically generated code is useful only for simulations. The quality is not sufficient for production code.

Embedded Coder®

The Embedded coder is an automatic C language code generator for Simulink and Stateflow. This is the automatic C language code generator used for production code

S-Function

A customized Simulink block written in C or M-code. C-code S-functions can be inlined in the 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 `sfun` 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

Simulink

Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate, 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 blocksets. These combined models are simulated using Simulink.

State

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

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

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

Stateflow Block

The *Stateflow block* is a masked Simulink model and is equivalent to an empty, untitled Stateflow diagram. Use the Stateflow block to include a Stateflow diagram in a Simulink model.
The control behavior that Stateflow models complements the algorithmic behavior modeled in Simulink block diagrams. By incorporating Stateflow blocks into Simulink models, you can add complex event-driven behavior to Simulink simulations. You create models that represent both data and control flow by combining Stateflow blocks with the standard Simulink and toolbox block libraries. These combined models are simulated using Simulink.

Stateflow Debugger

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

Stateflow Diagram

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

Stateflow Explorer

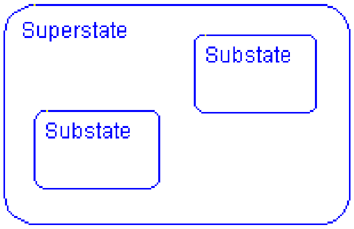
Use the *Stateflow Explorer* to add, remove, and modify data, event, and target objects.

Stateflow Finder

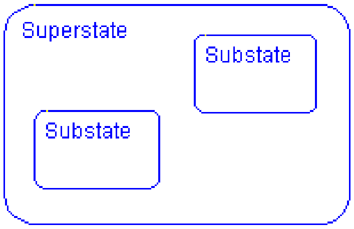
Use the *Finder* to display a list of objects based on search criteria you specify. You can directly access the properties dialog box of any object in the search output display by clicking on that object.

Substate

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



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



Target
An executable program built from code generated by Embedded Coder

Topdown Processing

Topdown 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.

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 Simulink's virtual and conditionally virtual blocks

Virtual Blocks

Block Name	**Condition Under Which Block Will Be Virtual**
Bus Selector	Always virtual.
Data Store Memory	Always virtual.
Demux	Always virtual.
Enable Port	Always virtual.
From	Always virtual.
Goto	Always virtual.
Goto Tag Visibility	Always virtual.
Ground	Always virtual.
Inport	Always virtual *unless* the block resides in a conditionally executed subsystem *and* has a direct connection to an outport block.
Mux	Always virtual.
Outport	Virtual if the block resides within any subsystem block (conditional or not), and does *not* reside in the root (top-level) Simulink window.
Selector	Always virtual.
Subsystem	Virtual if the block is not conditionally executed.
Terminator	Always virtual.

Test Point	Always virtual.
Trigger Port	Virtual if the outport port is not present.

Virtual Scrollbar

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

8 INDEX

__Ports_in_Simulink_models__Unconnected_signals__Unconnected_signals__Simulink_signal_appearance: Use of labels and label instances for signals and busses Scope of events__Event_Broadcasts__Stateflow_port_names__Indexing: Flowchart patterns for conditions: Flowchart patterns for condition actions: Flowchart patterns for conditions: Flowchart patterns for condition actions: Flowchart patterns for conditions: Flowchart patterns for condition actions