Discrete event dynamic systems (DEDS)

In control engineering, a **discrete event dynamic system (DEDS)** is a discrete-state, event-driven system of which the state evolution depends entirely on the occurrence of asynchronous discrete events over time. Although similar to **continuous-variable dynamic systems (CVDS)**, DEDS consists solely of discrete state spaces and event-driven state transition mechanisms.

STATEFLOW

- Development environment for *modeling, designing and simulating (& animating)* complex **combinatorial and sequential decision logic** (Reactive or event-driven or discrete-event Systems) based on (finite) **state machines** and **flow charts**.
- Allows combining graphical and tabular representations of ***Classic*** [default: full set of semantics], ***Mealy*** [output is a function of inputs *and* state] and ***Moore*** [output is a function of state only] **finite-state machines**: state transition diagrams, flow charts, state transition tables/matrices, and truth tables.
- Deterministic execution semantics with hierarchy, parallelism, temporal operators, and events
  $\rightarrow$
- Model how the system **reacts** to events, time-based conditions, and external input signals.
- State diagram animation, state activity logging, data logging, and integrated debugging: **static and run-time checks** for transition conflicts, cyclic problems, state inconsistencies, data-range violations, and overflow conditions (for testing design consistency and completeness, and detecting runtime errors before implementation).

Applications: supervisory control, task scheduling, and fault management applications.

Difference between **Flow Graphs** and **State Charts**
A **flow graph** is a *stateless flow chart* because it cannot maintain its active state between updates. As a result, a flow graph always begins executing from a default transition and ends at a *terminating junction* (a junction that has no valid outgoing transitions).

By contrast, a **state chart** stores its current state in memory to preserve local data and activity between updates. As a result, state charts can begin executing where they left off in the previous time step, making them suitable for modeling reactive or supervisory systems that depend on history. In these kinds of systems, the current result depends on a previous result.

Finite State Machine Concepts

What Is a Finite State Machine?

A Stateflow chart is an example of a finite state machine.

A **finite state machine** is a representation of an event-driven (reactive) system.
In an **event-driven (reactive) system**, the system makes a transition from one state (mode) to another, if the condition defining the change is true.

For example, you can use a state machine to represent the **automatic transmission** of a car. The transmission has these operating states: park, reverse, neutral, drive, and low. As the driver shifts from one position to another, the system makes a transition from one state to another, for example, from park to reverse.

Finite State Machine Representations

Traditionally, designers used **truth tables** to represent relationships among the inputs, outputs, and states of a finite state machine. The resulting table describes the logic necessary to control the behavior of the system under study.

Another approach to designing event-driven systems is to model the behavior of the system by describing it in terms of transitions among states. The occurrence of events under certain conditions determine the state that is active. **State-transition charts** and **bubble charts** are graphical representations based on this approach.

Stateflow Chart Representations

A Stateflow chart uses a variant of the finite state machine notation established by Harel [1]. A chart is a graphical representation of a finite state machine, where *states* and *transitions* form the basic building blocks of the system.

You can also represent **stateless charts (flow graphs)**. You can include Stateflow charts as blocks in a Simulink® model. The collection of Stateflow blocks in a Simulink model is the Stateflow machine. A **Stateflow chart** enables the representation of **hierarchy, parallelism, and history**. You can organize complex systems by defining a parent and offspring object structure [2]. For example, you can organize states within other higher-level states. A system with parallelism can have two or more orthogonal states active at the same time. You can specify the destination state of a transition based on historical information. These characteristics go beyond what state-transition charts and bubble charts provide.

Notation

Notation defines a set of objects and the rules that govern the relationships between those objects. Stateflow chart notation provides a way to communicate the design information in a Stateflow chart. Stateflow chart notation consists of these elements:

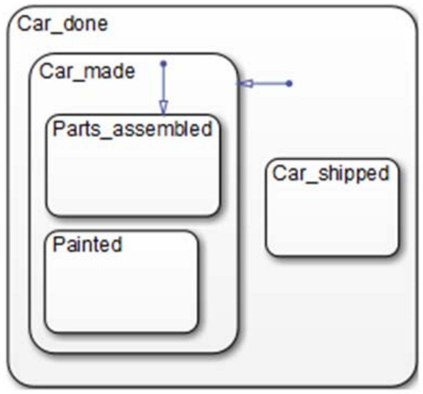
- A set of graphical objects
- A set of nongraphical text-based objects
- Defined relationships between those objects

Semantics
Semantics describe how to interpret chart notation. A typical Stateflow chart contains actions associated with transitions and states. The semantics describe the sequence of these actions during chart execution.

What Is a State?
A state describes an operating mode of a reactive system. In a Stateflow chart, states represent operating modes.
States can be active or inactive. The activity or inactivity of a state can change depending on events and conditions. The occurrence of an event drives the execution of the state chart by making states become active or inactive. At any point (time instant) during chart execution, active and inactive states exist.

State Hierarchy
To manage multilevel state complexity, use hierarchy in your state chart. With hierarchy, you can represent multiple levels of subcomponents in a system.

State Hierarchy **Example**
In the following example, three levels of hierarchy appear in the chart. Drawing one state within the boundaries of another state indicates that the inner state is a substate (or child) of the outer state (or superstate). The outer state is the parent of the inner state.



In this example, the chart is the parent of the state Car_done. The state Car_done is the parent state of the Car_made and Car_shipped states. The state Car_made is also the parent of the Parts_assembled and Painted states. You can also say that the states Parts_assembled and Painted are children of the Car_made state. To represent the Stateflow hierarchy textually, use a slash character (/) to represent the chart and use a period (.) to separate each level in the hierarchy of states. The following list is a textual representation of the hierarchy of objects in the preceding example:

- /Car_done
- /Car_done.Car_made
- /Car_done.Car_shipped
- /Car_done.Car_made.Parts_assembled
- /Car_done.Car_made.Painted

Objects That a State Can Contain

States can contain all other Stateflow objects except targets. Stateflow chart notation supports the representation of graphical object **hierarchy** in Stateflow charts with containment. A state is a **superstate** if it contains other states. A state is a **substate** if it is contained by another state. A state that is neither a superstate nor a substate of another state is a state whose parent is the Stateflow chart itself.

States can also contain nongraphical data and **event** objects. The hierarchy of this containment appears in the Model Explorer. You define data and event containment by specifying the parent object of the data or event.

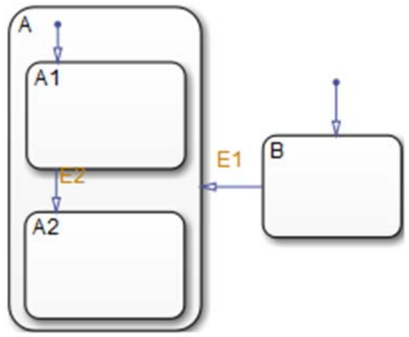
State Decomposition

Every state (or chart) has a decomposition that dictates what type of substates the state (or chart) can contain. All substates of a superstate must be of the same type as the superstate decomposition. State decomposition can be exclusive (OR) or parallel (AND).

Exclusive (OR) State Decomposition

Substates with solid borders indicate exclusive (OR) state decomposition. Use this decomposition to describe operating modes that are **mutually exclusive**. When a state has exclusive (OR) decomposition, only one substate can be active at a time.

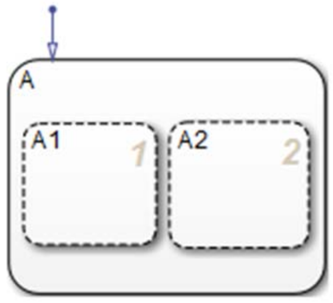
In the following example, either state A or state B can be active. If state A is active, either state A1 or state A2 can be active at a given time.



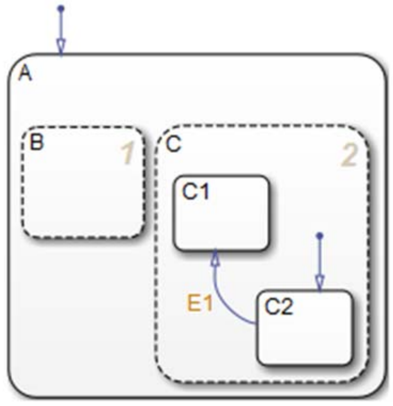
Parallel (AND) State Decomposition

Substates with dashed borders indicate parallel (AND) decomposition. Use this decomposition to describe **concurrent** operating modes. When a state has parallel (AND) decomposition, all substates are active at the same time.

In the following example, when state A is active, A1 and A2 are both active at the same time.



The activity within parallel states is essentially independent, as demonstrated in the following example.
In the following example, when state A becomes active, both states B and C become active at the same time. When state C becomes active, either state C1 or state C2 can be active.

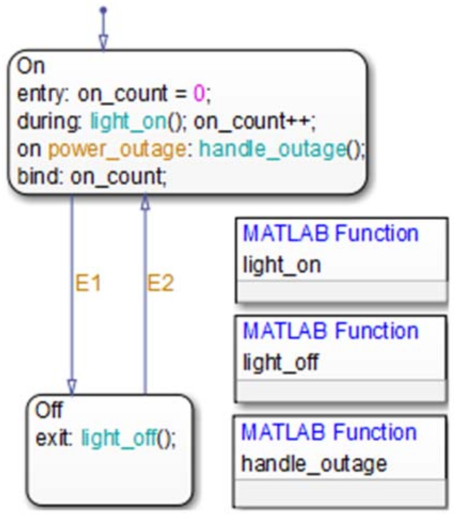


State Labels

The label for a state appears on the top left corner of the state rectangle with the following general format:

name/
entry:entry actions
during:during actions
exit:exit actions
on event_name:on event_name actions
bind:events, data

The following example demonstrates the components of a state label.



Each action in the state label appears in the subtopics that follow. For more information on state actions, see:

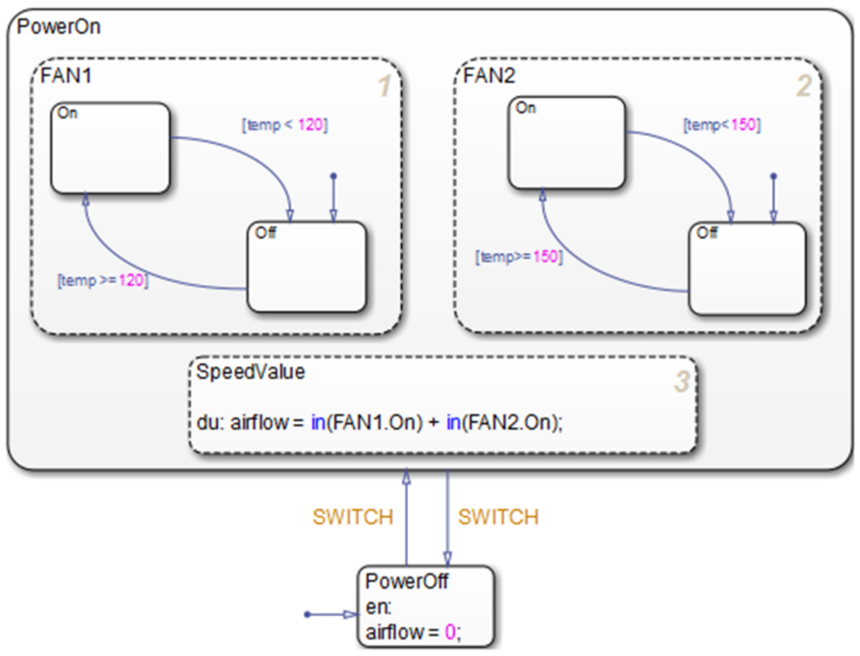
- Process for Entering, Executing, and Exiting States — Describes how and when entry, during, exit, and on event_name actions occur.
- State Action Types — Gives more detailed descriptions of each type of state action.

State Name

A state label starts with the name of the state followed by an optional / character. In the preceding example, the state names are On and Off. Valid state names consist of alphanumeric characters and can include the underscore (_) character. For more information, see Rules for Naming Stateflow Objects.

Hierarchy provides some flexibility in naming states. The name that you enter on the state label must be unique when preceded by ancestor states. The name in the Stateflow hierarchy is the text you enter as the label on the state, preceded by the names of parent states separated by periods. Each state can have the same name appear in the label, as long as their full names within the hierarchy are unique. Otherwise, the parser indicates an error.

The following example shows how unique naming of states works.



Each of these states has a unique name because of its location in the chart. The full names for the states in FAN1 and FAN2 are:

- PowerOn.FAN1.On
- PowerOn.FAN1.Off
- PowerOn.FAN2.On
- PowerOn.FAN2.Off

State Actions

After the name, you enter optional action statements for the state with a keyword label that identifies the type of action. You can specify none, some, or all of them. The colon after each keyword is required. The slash following the state name is optional as long as it is followed by a carriage return.

For each type of action, you can enter more than one action by separating each action with a carriage return, semicolon, or a comma. You can specify actions for more than one event by adding additional on event_name lines for different events.

If you enter the name and slash followed directly by actions, the actions are interpreted as entry action(s). This shorthand is useful if you are specifying only entry actions.

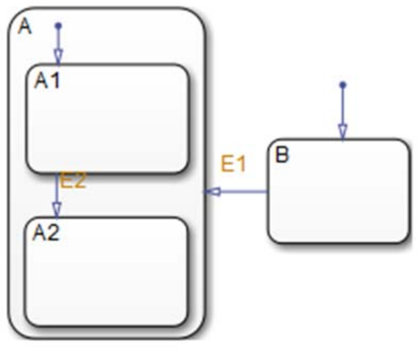
Entry Action. Preceded by the prefix entry or en for short. In the preceding example, state On has entry action on_count=0. This means that the value of on_count is reset to 0 whenever state On becomes active (entered).

During Action. Preceded by the prefix during or du for short. In the preceding label example, state On has two during actions, light_on() and on_count++. These actions are executed whenever state On is already active and any event occurs.

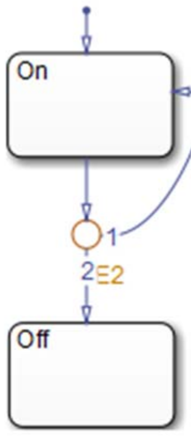
Exit Action. Preceded by the prefix exit or ex for short. In the preceding label example, state Off has the exit action light_off(). If the state Off is active, but becomes inactive (exited), this action is executed.

On Event_Name Action. Preceded by the prefix on event_name, where event_name is a unique event. In the preceding label example, state On has an on power_outage action. If state On is active and the event power_outage occurs, the action handle_outage() is executed.
Bind Action. Preceded by the prefix bind. In the preceding label example, the data on_count is bound to the state On. This means that only the state On or a child of On can change the value of on_count. Other states, such as the state Off, can use on_count in its actions, but it cannot change its value in doing so.

What Is a Transition?
A transition is a line with an arrowhead that links one graphical object to another. In most cases, a transition represents the passage of the system from one mode (state) object to another. A transition typically connects a source and a destination object. The source object is where the transition begins and the destination object is where the transition ends. The following chart shows a transition from a source state, B, to a destination state, A.



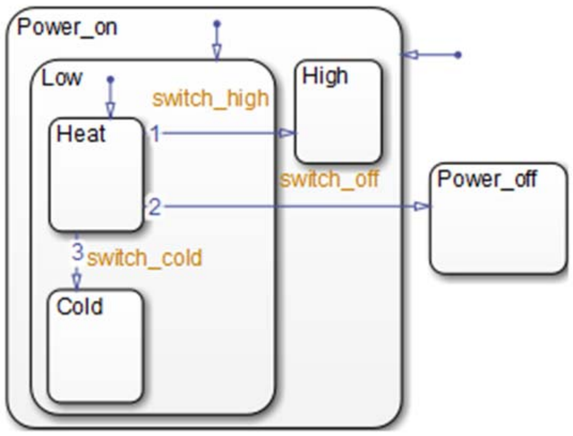
Junctions divide a transition into transition segments. In this case, a full transition consists of the segments taken from the origin to the destination state. Each segment is evaluated in the process of determining the validity of a full transition.
The following example has two segmented transitions: one from state On to state Off, and the other from state On to itself:



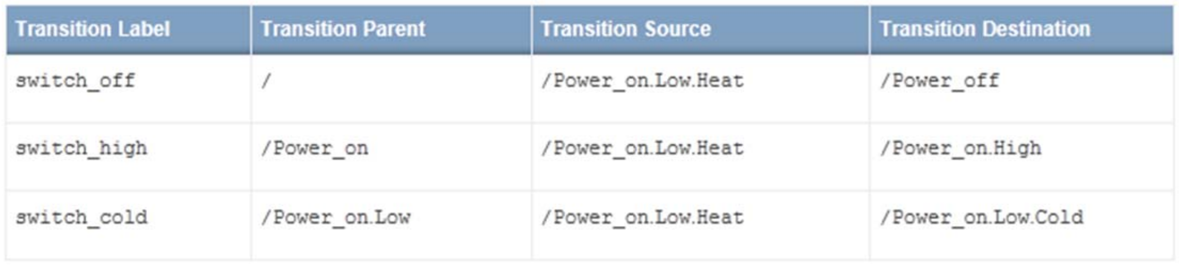
A default transition is a special type of transition that has no source object. See Default Transitions for details.

Transition Hierarchy

Transitions cannot contain other objects the way that states can. However, transitions are contained by states. A transition's hierarchy is described in terms of the transition's parent, source, and destination. The parent is the lowest level that contains the source and destination of the transition. Consider the parents for the transitions in the following example:



The following table resolves the parentage of each transition in the preceding example. The / character represents the chart. Each level in the hierarchy of states is separated by the period (.) character.



Transition Label	Transition Parent	Transition Source	Transition Destination
switch_off	/	/Power_on.Low.Heat	/Power_off
switch_high	/Power_on	/Power_on.Low.Heat	/Power_on.High
switch_cold	/Power_on.Low	/Power_on.Low.Heat	/Power_on.Low.Cold

Transition Label Notation

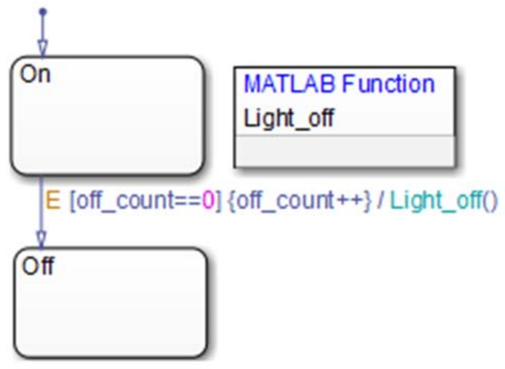
A transition is characterized by its label. The label can consist of an event, a condition, a condition action, and/or a transition action. The ? character is the default transition label. Transition labels have the following general format:

event[condition]{condition_action}/transition_action

You replace the names for event, condition, condition_action, and transition_action with appropriate contents as shown in the example Transition Label Example. Each part of the label is optional.

Transition Label Example

Use the following example to understand the parts of a transition label.



Event Trigger. Specifies an event that causes the transition to be taken, provided the condition, if specified, is true. Specifying an event is optional. The absence of an event indicates that the transition is taken upon the occurrence of any event. Specify multiple events using the OR logical operator (|).

In the preceding example, the broadcast of event E triggers the transition from On to Off as long as the condition [off_count==0] is true.

Condition. Specifies a Boolean expression that, when true, validates a transition to be taken for the specified event trigger. Enclose the condition in square brackets ([]). See Conditions for information on the condition notation.

In the preceding example, the condition [off_count==0] must evaluate as true for the condition action to be executed and for the transition from the source to the destination to be valid.

Condition Action. Follows the condition for a transition and is enclosed in curly braces ({}). It is executed as soon as the condition is evaluated as true and before the transition destination has been determined to be valid. If no condition is specified, an implied condition evaluates to true and the condition action is executed.

In the preceding example, if the condition [off_count==0] is true, the condition action off_count++ is immediately executed.

Transition Action. Executes after the transition destination has been determined to be valid provided the condition, if specified, is true. If the transition consists of multiple segments, the transition action is only executed when the entire transition path to the final destination is determined to be valid. Precede the transition action with a /.

In the preceding example, if the condition [off_count==0] is true, and the destination state Off is valid, the transition action Light_off is executed.

Valid Transitions

In most cases, a transition is valid when the source state of the transition is active and the transition label is valid. Default transitions are different because there is no source state. Validity of a default transition to a substate is evaluated when there is a transition to its superstate, assuming the superstate is active. This labeling criterion applies to both default transitions and general case transitions. The following table lists possible combinations of valid transition labels.

Transition Label	Is Valid If...
Event only	That event occurs
Event and condition	That event occurs and the condition is true
Condition only	Any event occurs and the condition is true
Action only	Any event occurs
Not specified	Any event occurs