Maintain Level-1 MATLAB S-Functions

**About the Maintenance of Level-1 MATLAB S-Functions**

**Note**

The information provided in this section is intended only for use in maintaining existing Level-1 MATLAB® S-functions. Use the more capable Level-2 API to develop new MATLAB S-functions (see [Write Level-2 MATLAB S-Functions](https://www.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html)). Level-1 MATLAB S-functions support a much smaller subset of the S-function API then Level-2 MATLAB S-functions, and their features are limited compared to built-in blocks.

A Level-1 MATLAB S-function is a MATLAB function of the following form

[sys,x0,str,ts]=*f*(t,x,u,flag,p1,p2,...)

where *f* is the name of the S-function. During simulation of a model, the Simulink® engine repeatedly invokes *f*, using the flag argument to indicate the task (or tasks) to be performed for a particular invocation. The S-function performs the task and returns the results in an output vector.

A template implementation of a Level-1 MATLAB S-function, [sfuntmpl.m](matlab:edit(fullfile(matlabroot,'/toolbox/simulink/blocks/sfuntmpl.m'))" \t "_blank), resides in the folder *matlabroot*/toolbox/simulink/blocks. The template consists of a top-level function and a set of skeleton local functions, called S-function callback methods, each of which corresponds to a particular value of flag. The top-level function invokes the local function indicated by flag. The local functions perform the actual tasks required of the S-function during simulation.

**Level-1 MATLAB S-Function Arguments**

The Simulink engine passes the following arguments to a Level-1 MATLAB S-function:

|  |  |
| --- | --- |
| t | Current time |
| x | State vector |
| u | Input vector |
| flag | Integer value that indicates the task to be performed by the S-function |

The following table describes the values that flag can assume and lists the corresponding Level-2 MATLAB S-function method for each value.

**Flag Argument**

| **Level-1 Flag** | **Level-2 Callback Method** | **Description** |
| --- | --- | --- |
| 0 | setup | Defines basic S-Function block characteristics, including sample times, initial conditions of continuous and discrete states, and the sizes array (see [Define S-Function Block Characteristics](https://www.mathworks.com/help/simulink/sfg/maintaining-level-1-matlab-s-functions.html#f7-13642) for a description of the sizes array). |
| 1 | mdlDerivatives | Calculates the derivatives of the continuous state variables. |
| 2 | mdlUpdate | Updates discrete states, sample times, and major time step requirements. |
| 3 | mdlOutputs | Calculates the outputs of the S-function. |
| 4 | mdlOutputs method updates the run-time object NextTimeHit property | Calculates the time of the next hit in absolute time. This routine is used only when you specify a variable discrete-time sample time in the setup method. |
| 9 | mdlTerminate | Performs any necessary end-of-simulation tasks. |

**Level-1 MATLAB S-Function Outputs**

A Level-1 MATLAB S-function returns an output vector containing the following elements:

* sys, a generic return argument. The values returned depend on the flag value. For example, for flag = 3, sys contains the S-function outputs.
* x0, the initial state values (an empty vector if there are no states in the system). x0 is ignored, except when flag = 0.
* str, originally intended for future use. Level-1 MATLAB S-functions must set this to the empty matrix, [].
* ts, a two-column matrix containing the sample times and offsets of the block (see [Specify Sample Time](https://www.mathworks.com/help/simulink/ug/how-to-specify-the-sample-time.html) in *Using Simulink* for information on how to specify a sample times and offsets).

For example, if you want your S-function to run at every time step (continuous sample time), set ts to [0 0]. If you want your S-function to run at the same rate as the block to which it is connected (inherited sample time), set ts to [-1 0]. If you want it to run every 0.25 seconds (discrete sample time) starting at 0.1 seconds after the simulation start time, set ts to [0.25 0.1].

You can create S-functions that do multiple tasks, each at a different sample rate (i.e., a multirate S-function). In this case, ts should specify all the sample rates used by your S-function in ascending order by sample time. For example, suppose your S-function performs one task every 0.25 second starting from the simulation start time and another task every 1 second starting 0.1 second after the simulation start time. In this case, your S-function should set ts equal to [.25 0; 1.0 .1]. This will cause the Simulink engine to execute the S-function at the following times: [0 0.1 0.25 0.5 0.75 1 1.1 ...]. Your S-function must decide at every sample time which task to perform at that sample time.

You can also create an S-function that performs some tasks continuously (i.e., at every time step) and others at discrete intervals.

**Define S-Function Block Characteristics**

For the Simulink engine to recognize a Level-1 MATLAB S-function, you must provide it with specific information about the S-function. This information includes the number of inputs, outputs, states, and other block characteristics.

To provide this information, call the simsizes function at the beginning of the S-function.

sizes = simsizes;

This function returns an uninitialized sizes structure. You must load the sizes structure with information about the S-function. The table below lists the fields of the sizes structure and describes the information contained in each field.

**Fields in the sizes Structure**

| **Field Name** | **Description** |
| --- | --- |
| sizes.NumContStates | Number of continuous states |
| sizes.NumDiscStates | Number of discrete states |
| sizes.NumOutputs | Number of outputs |
| sizes.NumInputs | Number of inputs |
| sizes.DirFeedthrough | Flag for direct feedthrough |
| sizes.NumSampleTimes | Number of sample times |

After you initialize the sizes structure, call simsizes again:

sys = simsizes(sizes);

This passes the information in the sizes structure to sys, a vector that holds the information for use by the Simulink engine.

**Processing S-Function Parameters**

When invoking a Level-1 MATLAB S-function, the Simulink engine always passes the standard block parameters, t, x, u, and flag, to the S-function as function arguments. The engine can pass additional block-specific parameters specified by the user to the S-function. The user specifies the parameters in the **S-function parameters** field of the S-Function Block Parameters dialog box (see [Passing Parameters to S-Functions](https://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html#f6-54365)). If the block dialog specifies additional parameters, the engine passes the parameters to the S-function as additional function arguments. The additional arguments follow the standard arguments in the S-function argument list in the order in which the corresponding parameters appear in the block dialog. You can use this block-specific S-function parameter capability to allow the same S-function to implement various processing options. See the [limintm.m](matlab:edit(fullfile(matlabroot,'/toolbox/simulink/simdemos/simfeatures/limintm.m'))" \t "_blank) example in the folder *matlabroot*/toolbox/simulink/simdemos/simfeatures for an example of an S-function that uses block-specific parameters.

**Convert Level-1 MATLAB S-Functions to Level-2**

You can convert Level-1 MATLAB S-functions to Level-2 MATLAB S-functions by mapping the code associated with each Level-1 S-function flag to the appropriate Level-2 S-function callback method. See the [Flag Arguments](https://www.mathworks.com/help/simulink/sfg/maintaining-level-1-matlab-s-functions.html#f7-55187) table for a mapping of Level-1 flags to Level-2 callback methods. In addition:

* Store discrete state information for Level-2 MATLAB S-functions in DWork vectors, initialized in the PostPropagationSetup method.
* Access Level-2 MATLAB S-function dialog parameters using the DialogPrm run-time object property, instead of passing them into the S-function as function arguments.
* For S-functions with variable sample times, update the NextTimeHit run-time object property in the Outputs method to set the next sample time hit for the Level-2 MATLAB S-function.

For example, the following table shows how to convert the Level-1 MATLAB S-function [sfundsc2.m](matlab:edit('sfundsc2.m');) to a Level-2 MATLAB S-function. The example uses the Level-2 MATLAB S-function template msfuntmpl\_basic.m as a starting point when converting the Level-1 MATLAB S-function. The line numbers in the table corresponds to the lines of code in sfundsc2.m.

[](https://www.mathworks.com/help/simulink/sfg/maintaining-level-1-matlab-s-functions.html) [sfundsc2\_level2.m](https://www.mathworks.com/help/simulink/sfg/maintaining-level-1-matlab-s-functions.html)

| **Line Number** | **Code in sfundsc2.m** | **Code in Level-2 MATLAB file (sfundsc2\_level2.m)** |
| --- | --- | --- |
| 1 | function [sys,x0,str,ts]= ...  sfundsc2(t,x,u,flag) | function sfundsc2(block)  setup(block);  The syntax for the function line changes to accept one input argument block, which is the Level-2 MATLAB S-Function block's run-time object. The main body of the Level-2 MATLAB S-function contains a single line that calls the local setup function. |
| 13 - 19 | switch flag,  case 0,  [sys,x0,str,ts] = ...  mdlInitializeSizes; | function setup(block)  The flag value of zero corresponds to calling the setup method. A Level-2 MATLAB S-function does not use a switch statement to invoke the callback methods. Instead, the local setup function registers callback methods that are directly called during simulation. |
| 24 - 31 | case 2,  sys = mdlUpdate(t,x,u);  case 3,  sys = mdlOutputs(t,x,u); | The setup function registers two local functions associated with flag values of 2 and 3.  block.RegBlockMethod('Outputs' ,@Output);  block.RegBlockMethod('Update' ,@Update); |
| 53 - 66 | sizes = simsizes;  sizes.NumContStates = 0;  sizes.NumDiscStates = 1;  sizes.NumOutputs = 1;  sizes.NumInputs = 1;  sizes.DirFeedthrough = 0;  sizes.NumSampleTimes = 1;  sys = simsizes(sizes);  x0 = 0;  str = [];  ts = [.1 0]; | The setup function also initializes the attributes of the Level-2 MATLAB S-function:  block.NumInputPorts = 1;  block.NumOutputPorts = 1;  block.InputPort(1).Dimensions = 1;  block.InputPort(1).DirectFeedthrough = false;  block.OutputPort(1).Dimensions = 1;  block.NumDialogPrms = 0;  block.SampleTimes = [0.1 0];  Because this S-function has discrete states, the setup method registers the PostPropagationSetup callback method to initialize a DWork vector and the InitializeConditions callback method to set the initial state value.  block.RegBlockMethod('PostPropagationSetup',...  @DoPostPropSetup);  block.RegBlockMethod('InitializeConditions', ...  @InitConditions); |
| 56 | sizes.NumDiscStates = 1; | The PostPropagationSetup method initializes the DWork vector that stores the single discrete state.  function DoPostPropSetup(block)  %% Setup Dwork  block.NumDworks = 1;  block.Dwork(1).Name = 'x0';  block.Dwork(1).Dimensions = 1;  block.Dwork(1).DatatypeID = 0;  block.Dwork(1).Complexity = 'Real';  block.Dwork(1).UsedAsDiscState = true; |
| 64 | x0 = 0; | The InitializeConditions method initializes the discrete state value.  function InitConditions(block)  %% Initialize Dwork  block.Dwork(1).Data = 0 |
| 77 - 78 | function sys = ...  mdlUpdate(t,x,u)  sys = u; | The Update method calculates the next value of the discrete state.  function Update(block)  block.Dwork(1).Data = block.InputPort(1).Data; |
| 88 - 89 | function sys = ...  mdlOutputs(t,x,u)  sys = x; | The Outputs method calculates the S-function's output.  function Outputs(block)  block.OutputPort(1).Data = block.Dwork(1).Data; |