

**Simulink<sup>®</sup> Compiler<sup>™</sup>**

Reference



**MATLAB<sup>®</sup>&SIMULINK<sup>®</sup>**

R2021b



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



The MathWorks, Inc.  
1 Apple Hill Drive  
Natick, MA 01760-2098

## *Simulink® Compiler™ Reference*

© COPYRIGHT 2020–2021 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

March 2020	Online only	New for Version 1.0 (Release 2020a)
September 2020	Online only	Revised for Version 1.1 (Release 2020b)
March 2021	Online only	Revised for Version 1.2 (Release 2021a)
September 2021	Online only	Revised for Version 1.3 (Release 2021b)

<b>1</b>	<b>Functions</b>
----------	------------------



# Functions

---

# simulink.compiler.configureForDeployment

Configure Simulink.SimulationInput object for deployment with Simulink Compiler

## Syntax

```
simulink.compiler.configureForDeployment(in)
```

## Description

`simulink.compiler.configureForDeployment(in)` configures the Simulink.SimulationInput object, `in`, to be compatible for deployment with Simulink® Compiler™. `simulink.compiler.configureForDeployment` sets the simulation mode to Rapid Accelerator and the model parameter, `RapidAcceleratorUpToDateCheck` to `off` for the model used in Simulink.SimulationInput object, `in`, these settings ensure that the specified inputs does not require the deployed applications to be rebuilt.

## Examples

### Configure the Simulink.SimulationInput Object for Deployment with Simulink Compiler

This example shows how to configure a Simulink.SimulationInput object for deployment as a command line executable or an application with Simulink Compiler.

Create a function that you want to deploy as a standalone executable. In the function, create a Simulink.SimulationInput object for the model `sldemo_suspn_3dof`. Using the `setVariable` method of the Simulink.SimulationInput object, set the variable `Mb` to 1000.

Use the `simulink.compiler.configureForDeployment` function to make the Simulink.SimulationInput object compatible for deployment. Once the Simulink.SimulationInput object is configured for deployment, simulate it with the `sim` command.

```
function deployedScript()
    in = Simulink.SimulationInput('sldemo_suspn_3dof');
    in = in.setVariable('Mb', 1000);
    in = simulink.compiler.configureForDeployment(in);
    out = sim(in);
end
```

## Input Arguments

### **in** — Simulink.SimulationInput object

Simulink.SimulationInputobject | array of Simulink.SimulationInput objects

A Simulink.SimulationInput object or an array of Simulink.SimulationInput objects that is used to specify changes to the model for a simulation.

Example: `in = Simulink.SimulationInput('vdp')`

## **See Also**

sim | Simulink.SimulationInput | mcc | deploytool | exportToFMU2CS

## **Topics**

*"Simulink Compiler Workflow Overview"*

*"Create and Deploy a Script with Simulink Compiler"*

*"Deploy an App Designer Simulation with Simulink Compiler"*

*"Deploy Simulations with Tunable Parameters"*

## **Introduced in R2020a**

## exportToFMU2CS

Export Simulink model to Functional Mock-up Unit (FMU)

### Syntax

```
exportToFMU2CS mdl  
exportToFMU2CS mdl, Name, Value)
```

### Description

`exportToFMU2CS(mdl)` exports a model to a Functional Mock-Up Unit (FMU) and creates a model, `mdl_fm.u slx`, that contains a FMU Co-Simulation block with the original model. Model solver type must be a `fixed-step` solver.

`exportToFMU2CS(mdl, Name, Value)` exports a model to a Functional Mock-Up Unit (FMU) using one or more `Name, Value` pair arguments.

### Examples

#### Export a Model to FMU

Export the model, `vdp` to a Functional Mock-Up Unit.

Open the model.

```
open_system('vdp')
```

Set the solver type of the model to `fixed-step`.

```
set_param('vdp', 'SolverType', 'Fixed-step')
```

Export the model to a FMU. The model, `vdp_fm.u slx` is created from the exported FMU.

```
exportToFMU2CS('vdp')
```

### Input Arguments

#### **mdl** — Name of Model

model name

Name of the model to be exported to a Functional Mock-Up Unit

#### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name, Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `'CreateModelAfterGeneratingFMU', 'off'`



**CreateModelAfterGeneratingFMU – Create model after export**

'on' (default) | 'off'

By default, creates a model, `mdl_fm_u.slx`, that contains a FMU Co-Simulation block with the original model. Create this model to check the integrity of the exported FMU.

When set to 'off', does not create a model.

**AddIcon – Block icon or exported FMU image**

'snapshot' (default) | 'off' | 'filename'

Block icon image, character vector specified as one of the following values:

- 'off' - No block icon image
- 'snapshot' - Use image of model as block icon
- 'filename' - An image file to be used as a block icon.

**Generate32BitDLL – Option to generate 32-bit DLL**

'off' (default) | 'on'

Set the option to 'on' to support exporting Co-simulations of FMUs with 32-bit binaries.

**See Also**`configureForDeployment` | `sim`**Topics**

“Import FMUs”

“Export Simulink Models to Functional Mock-up Units”

**Introduced in R2020a**

## simulink.compiler.genapp

Generate MATLAB App to simulate model and deploy application

### Syntax

```
simulink.compiler.genapp('modelName')  
simulink.compiler.genapp(modelName,Name,Value)
```

### Description

`simulink.compiler.genapp('modelName')` analyzes a Simulink model and generates a deployable MATLAB® app, to simulate the model in rapid accelerator simulation mode with different inputs, parameters, and initial states and plot the results.

`simulink.compiler.genapp(modelName,Name,Value)` generates a deployable MATLAB app with the specified options.

While generating an app, ensure that the current working folder does not contain older generated app artifacts.

### Examples

#### Generate a MATLAB App for a Simulink Model

This example shows how to generate a MATLAB app using the `simulink.compiler.genapp` function for the model, `sldemo_suspn_3dof`.

Open the model

```
open_system('sldemo_suspn_3dof')
```

Generate a MATLAB app for the model with app name, `suspn_3dof_app`.

```
simulink.compiler.genapp('sldemo_suspn_3dof', 'AppName', 'suspn_3dof_app')
```

Once the app is generated, click **Simulate** to view the simulation result of the model

#### Generate App Using the SLSimApp2 Template

The `simulink.compiler.genapp` function also allows you to generate an app with the `SLSimApp2` template. The `SLSimApp2` template uses pop-up windows for editing reference variables and selecting signals to plot. To generate an app using this template, use the name-value pair along with the model name as arguments in the `simulink.compiler.genapp` function.

Before generating the app, clear the generated artifacts from the Current Folder and the workspace.

```
myApp = simulink.compiler.genapp('sldemo_suspn_3dof', 'Template', 'SLSimApp2')
```

## Input Arguments

### modelName — Name of model

string

Name of model for which the MATLAB app is generated, specified as a string.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: 'AppName', 'modelNameApp'

### AppName — Name of the app

modelName\_SLSimApp (default) | string

Name of the generated app, specified as the comma-separated pair consisting of 'AppName' and a string.

### Template — Template to use

SimAppTemplate (default) | string

Template to use to generate a MATLAB app. Specified as the comma-separated pair consisting of 'Template' and a string.

### OutputDir — Directory for creating build artifacts

pwd (default) | string

Directory for creating build artifacts, as the comma-separated pair consisting of 'OutputDir' and a string.

### InputMatFiles — MAT files that specify inputs

MAT file name

MAT files that specify inputs for the `Simulink.Simulation`. Input objects, specified as the comma-separated pair consisting of 'InputMatFiles' and a MAT file.

## See Also

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment` | **applicationCompiler**

### Topics

“Simulink Compiler Workflow Overview”

“Deploy an App Designer Simulation with Simulink Compiler”

“Generate, Modify and Deploy a MATLAB App for a Simulink Model”

### Introduced in R2020b

# simulink.compiler.getTunableVariables

Find names of all tunable variables

## Syntax

```
simulink.compiler.getTunableVariables(modelName)
```

## Description

`simulink.compiler.getTunableVariables(modelName)` returns a structure containing all the tunable variables in the model `modelName`, and their values.

Each leaf of a struct variable present in the model is given an entry in the output of `simulink.compiler.getTunableVariables` function.

## Examples

### Find Tunable Variables in a Model

This example shows how to use `simulink.compiler.getTunableVariables` to find the tunable variables in a model.

Open the model.

```
open_system('sldemo_suspn_3dof')
```

Find the tunable variables present in the model.

```
simulink.compiler.getTunableVariables('sldemo_suspn_3dof')
```

```
### Building the rapid accelerator target for model: sldemo_suspn_3dof
### Successfully built the rapid accelerator target for model: sldemo_suspn_3dof
```

Build Summary

Top model rapid accelerator targets built:

Model	Action	Rebuild Reason
sldemo_suspn_3dof	Code generated and compiled	Code generation information file does not exist.

```
1 of 1 models built (0 models already up to date)
Build duration: 0h 1m 1.066s
```

```
ans =
```

```
1×8 struct array with fields:
```

```
    QualifiedName
    Value
```

View a variable and its value.

```
ans(1)
```

```
ans =
```

```
struct with fields:
```

```
    QualifiedName: "Ixx"  
    Value: 1500
```

## Input Arguments

**modelName** — Name of the model

string

Name of model for which you want to find tunable parameters, specified as a string

## See Also

[simulink.compiler.configureForDeployment](#) | [simulink.compiler.genapp](#)

**Introduced in R2021b**

## **simulink.compiler.loadEnumTypes**

Configure model with enum types for deployment

### **Syntax**

```
simulink.compiler.loadEnumTypes('modelName')
```

### **Description**

`simulink.compiler.loadEnumTypes('modelName')` configures the deployment for models with enum types. Use the `simulink.compiler.loadEnumTypes` function only if the following is true:

- The `simulink.compiler.configureForDeployment` function is not used to configure the `Simulink.SimulationInput` object.
- Model in the deployed script or application uses enum types.
- The deployed scripts or applications refer to enum types before the execution of the `sim` command.

### **Input Arguments**

#### **modelName — Name of the model**

string

Name of the model, specified by a string, for which the enum types are loaded.

### **See Also**

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment`

### **Topics**

[“Simulink Compiler Workflow Overview”](#)

[“Create and Deploy a Script with Simulink Compiler”](#)

[“Deploy an App Designer Simulation with Simulink Compiler”](#)

[“Deploy Simulations with Tunable Parameters”](#)

### **Introduced in R2020a**

# simulink.compiler.modifyParameters

Tune block parameters at runtime via workspace variables

## Syntax

```
simulink.compiler.modifyParameters(modelName)
```

## Description

`simulink.compiler.modifyParameters(modelName)` tunes block parameters at simulation runtime via workspace variables. You can use `simulink.compiler.modifyParameters` to modify variables only during a running simulation.

You can use `simulink.compiler.modifyParameters` to tune any variables that are returned by the `simulink.compiler.getTunableVariables` function. `simulink.compiler.modifyParameters` is supported only for rapid accelerator and deployment workflows.

## Examples

### Use `simulink.compiler.modifyParameters` to Tune Block Parameters

This example shows you how to use the `simulink.compiler.modifyParameters` function to tune block parameters.

#### Open the Model

The example model `example_modify_parameters` references another model, `mRefUsingGlobalWksVars`. The top model gain block, the triggered subsystem and the gain block in the referenced model all use global workspace variables. The masked subsystem present in the model uses a global variable and a model workspace variable.

```
open_system("example_modify_parameters.slx");
```

#### Write a Function for Runtime Parameter Tuning

The following function sets the simulation mode to rapid and creates a `Simulink.SimulationInput` object. In this function, you can use `simulink.compiler.setPostStepFcn` API to set a callback which uses `simulink.compiler.modifyParameters` to tune block parameters

```
function runtimeParameterTuning()
simMode = 'rapid';
model = 'example_modify_parameters';
ref = 'exRefModelWorkspaceVars';
load_system(model);
closeModels = onCleanup(@(x) cellfun(@(x)close_system(x,0),{model,ref}));
set_param(model, "SimulationMode", simMode);
```

```
% Get simulation input object
```

```
simInput = Simulink.SimulationInput(model);  
% Set post-step callback function that tunes variables  
simInput = simulink.compiler.setPostStepFcn(simInput,@(time)postStepParameterTuner(time,model));  
  
out = sim(simInput);  
end
```

### Write a Function for the Post-Step Callback

The function `postStepParameterTuner` uses `simulink.compiler.modifyParameters` to modify the variables.

```
function postStepParameterTuner(time,model)  
% Callback which tunes parameters based on time  
if time==5.0  
    % Modify global variables used by top model gain block  
    newGlobalVars = [Simulink.Simulation.Variable('gNum',1.1),...  
                    Simulink.Simulation.Variable('gDen',0.5)];  
    simulink.compiler.modifyParameters(model,newGlobalVars);  
end  
if time==2.5  
    % Modify variables in reference model workspace  
    newRefWksVars = [Simulink.Simulation.Variable('gNum',1.2),...  
                    Simulink.Simulation.Variable('gDen',0.1)];  
    simulink.compiler.modifyParameters(model,newRefWksVars);  
end  
if time==4.5  
    % Modify variables used by mask dialog parameters  
    newMaskVars = [Simulink.Simulation.Variable('mGain',2.0),...  
                  Simulink.Simulation.Variable('Bias',-1,'Workspace',model)];  
    simulink.compiler.modifyParameters(model,newMaskVars);  
end  
end
```

## Input Arguments

### **modelName** — Name of the model

string

Name of model for which you want to find tunable parameters, specified as a string

## See Also

`simulink.compiler.configureForDeployment` | `simulink.compiler.genapp`

**Introduced in R2021b**



# simulink.compiler.setExternalInputsFcn

Set callback to specify data to each external root inport port block at the start of each simulation step

## Syntax

```
in = simulink.compiler.setExternalInputsFcn(in, @(id, time) getInput(id, time))
```

## Description

`in = simulink.compiler.setExternalInputsFcn(in, @(id, time) getInput(id, time))` function registers a callback that dynamically provides values for every external root input port block specified by `id` at the specified `time` at the root level of a model during simulation. The callback is required to return the value to be set at the inport block. To return the value from the callback, use the syntax, `returningValue = getInput(id, time)`.

## Examples

### Input Arguments

#### **in** – Simulation inputs

`Simulink.SimulationInput` object

Simulation inputs and changes to model for simulation, specified as a `Simulink.SimulationInput` object

Example: `in = Simulink.SimulationInput('vdp')`

#### **@(id, time) getInput(id, time)** – Function handle for callback

MATLAB function handle

Function handle for callback to provide values for each root inport block specified by `id` at simulation step time, `time`.

- `id` - A root inport block index, for which the callback is set, specified by a numerical value.
- `time` - Time for which the input to the root inport block is required, specified by a numeric value.

## See Also

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment`

## Topics

“Simulink Compiler Workflow Overview”

“Create and Deploy a Script with Simulink Compiler”

“Deploy an App Designer Simulation with Simulink Compiler”

“Deploy Simulations with Tunable Parameters”

**Introduced in R2020b**

# simulink.compiler.setExternalOutputsFcn

Set callback to read external root output block data after each simulation step

## Syntax

```
in = simulink.compiler.setExternalOutputsFcn(in, @(id, time, data)  
processOutput(id, time, data))
```

## Description

`in = simulink.compiler.setExternalOutputsFcn(in, @(id, time, data) processOutput(id, time, data))` function registers a callback to dynamically process the values for every output port at the root level of a model during simulation.

## Examples

### Deploy App with Live Simulation Results of Lorenz System

This example shows how to develop an app that uses callbacks for simulation inputs and outputs to view the simulation of a Simulink model of the Lorenz system. You can then deploy the app with Simulink® Compiler™.

### Open and Examine the Project File

This example uses a Simulink project that contains all the files required to run this example. The project contains a Simulink® model of the Lorenz system and an app, created in the MATLAB® App Designer that simulates the model with different input and output values. To learn more about how to create an app using the App Designer, see “Create and Run a Simple App Using App Designer”.

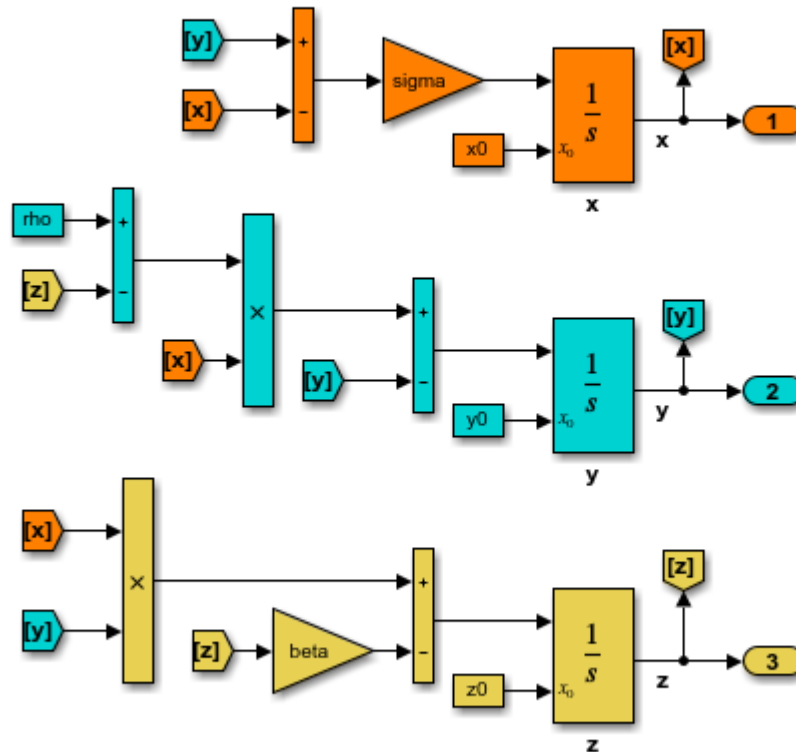
```
simulink.compiler.example.LorenzSystem
```

# 1 Functions

The screenshot shows the 'Project - LorenzSystem' window. At the top, there are tabs for 'PROJECT' and 'PROJECT SHORTCUTS'. Below these are buttons for 'New Shortcut' and 'Organize Groups', and project shortcuts for 'LorenzSystemApp' and 'LorenzSystemModel'. The main area is divided into 'Views' and 'Labels'. The 'Views' section shows a list of files with the following columns: Name, Status, and Classification.

Name	Status	Classification
AppStatus.m	✓	Design
LorenzAttractor.gif	✓	Other
LorenzEquations.svg	✓	Other
LorenzSystemApp.mlapp	✓	Design
LorenzSystemModel.gif	✓	Other
LorenzSystemModel.slx	✓	Design
onShutdown.m	✓	Design
onStartup.m	✓	Design

The 'Labels' section shows a 'Classification' label. At the bottom, there is a 'Details' section.



## App Details

Open the `LorenzSystemApp.mlapp` file. You can view the code written to create this app in the **Code View** section of App Designer. The essential part of building this app is the behavior of the **Simulate** button. It has the following salient parts: creating the `SimulationInput` object, configuring it for deployment, using simulation callbacks to read the output port data and plot the data at each time step. These three functions allow you to see the live results of the simulation in the deployed app.

### Create the `Simulink.SimulationInput` Object

In the function `createSimulationInput`, define an empty `Simulink.SimulationInput` object for the model. Use this `Simulink.SimulationInput` object to set simulation callbacks and variables for the model.

The simulation callback functions are used to register the callbacks. The `simulink.compiler.setPostStepFcn` function registers a callback that is invoked after every simulation step. The `simulink.compiler.setExternalOutputsFcn` registers a callback that dynamically processes the values for every output port at root level of a model during simulation.

Use the `setVariable` method of the `Simulink.SimulationInput` object to provide the parameter values to the app. Values for the simulation are obtained from the edit fields of the UI of the app. To deploy the app, use the `simulink.compiler.configureForDeployment` function. (Comment the line of code that calls `simulink.compiler.configureForDeployment` function for faster debugging.)

```
function simInp = createSimulationInput(app)
    % Create an empty SimulationInput object
    simInp = Simulink.SimulationInput('LorenzSystemModel');

    % Specify the simulation callbacks
    simInp = simulink.compiler.setPostStepFcn(simInp, @app.postStepFcn);
    simInp = simulink.compiler.setExternalOutputsFcn(simInp, @app.processOutputs);

    % Load the parameters values from the ui edit fields
    simInp = simInp.setVariable('rho', app.rhoUIC.Value);
    simInp = simInp.setVariable('beta', app.betaUIC.Value);
    simInp = simInp.setVariable('sigma', app.sigmaUIC.Value);
    simInp = simInp.setVariable('x0', app.x0UIC.Value);
    simInp = simInp.setVariable('y0', app.y0UIC.Value);
    simInp = simInp.setVariable('z0', app.z0UIC.Value);

    % Configure simInp for deployment
    % DEBUG TIP: Comment out the line below for
    % faster/easier debugging when running in MATLAB
    simInp = simulink.compiler.configureForDeployment(simInp);
end % createSimulationInput
```

### Simulation Callback Functions

The simulation callback functions register callbacks that allow you to read values from the output ports and to write values to the root input ports. These functions register callbacks at every simulation time step, which allows you to view live results of the simulation.

### The processOutputs Callback

The `simulink.compiler.setExternalOutputsFcn` line refers to the function `processOutputs`. The `processOutputs` callback function processes the values for every root output port block of model during simulation. The `processOutputs` function is called once per port and per the sample time of the port. When the `processOutputs` function is called, it reads the values for every root output block and caches those values. The `postStepFcn` obtains the cached values to update the plot.

```
function processOutputs(app, opIdx, ~, data)
    % Called during sim to process the external output port data,
    % will be called once per port per its sample hit.
    switch opIdx
        case 1
            app.txyzBuffer.x = data;
        case 2
            app.txyzBuffer.y = data;
        case 3
            app.txyzBuffer.z = data;
        otherwise
            error(['Invalid port index: ', num2str(opIdx)]);
    end
end
```

## The postStepFcn Callback

The `postStepFcn` callback function is invoked after every simulation step. The `time` argument is the time for the previous simulation step. The `postStepFcn` function obtains the cached output block values for every time and passes those values to the `updateTrace` function to plot the cached values at simulation time.

```
function postStepFcn(app, time)
    % Called during sim after each simulation time step
    app.updateSimStats(time);
    if app.status == AppStatus.Starting
        app.switchStatus(AppStatus.Running);
        app.simStats.WallClockTimeAfterFirstStep = tic;
    end
    if app.stopRequested
        app.switchStatus(AppStatus.Stopping);
        stopRequestedID = [mfilename('class'), ':StopRequested'];
        throw(MException(stopRequestedID, 'Stop requested'));
    end
    %-----
    app.txyzBuffer.t = time;
    x = [app.txyzBuffer.x];
    y = [app.txyzBuffer.y];
    z = [app.txyzBuffer.z];
    app.updateTrace(x, y, z);
    app.updateMarker('head', x, y, z);
    %-----
    drawnow limitrate;
end % postStepFcn
```

## Test in App Designer

Before deploying the application, ensure that the app runs in the App Designer. Click **Simulate** to verify that the application works by simulating the model for different values.

## Compile App for Deployment

You can use the App Designer to compile and deploy the app. You can also use the `deploytool` function. For more information on compiling and deploying with the App Designer, see [Develop Apps Using App Designer](#), [Web Apps and Application Compiler](#).

To compile the app in this example, use the `mcc` command followed by the app name.

```
mcc -m LorenzSystemApp
```

## Input Arguments

### in — Simulation inputs

`Simulink.SimulationInput` object

Simulation inputs and changes to model for simulation, specified as a `Simulink.SimulationInput` object

Example: `in = Simulink.SimulationInput('vdp')`

**@(id, time, data) processOutput(id, time, data) – Function handle for callback**  
MATLAB function handle

Function handle for callback to process outputs with the values, `data` for every root port block specified by `id` at simulation step time, `time`.

- `id` - A root output block index, for which the callback is set, specified by a numerical value.
- `time` - Time for which the input to the root output block is required, specified by a numeric value.
- `data` - Value for the root output block.

**See Also**

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment`

**Topics**

“Simulink Compiler Workflow Overview”

“Create and Deploy a Script with Simulink Compiler”

“Deploy an App Designer Simulation with Simulink Compiler”

“Deploy Simulations with Tunable Parameters”

**Introduced in R2020b**



# simulink.compiler.setPostStepFcn

Register a callback to run after each simulation step

## Syntax

```
in = simulink.compiler.setPostStepFcn(in, @(time) postStepFcn(time))
```

## Description

`in = simulink.compiler.setPostStepFcn(in, @(time) postStepFcn(time))` function registers a callback that is invoked after every simulation step.

## Examples

### Deploy App with Live Simulation Results of Lorenz System

This example shows how to develop an app that uses callbacks for simulation inputs and outputs to view the simulation of a Simulink model of the Lorenz system. You can then deploy the app with Simulink® Compiler™.

### Open and Examine the Project File

This example uses a Simulink project that contains all the files required to run this example. The project contains a Simulink® model of the Lorenz system and an app, created in the MATLAB® App Designer that simulates the model with different input and output values. To learn more about how to create an app using the App Designer, see “Create and Run a Simple App Using App Designer”.

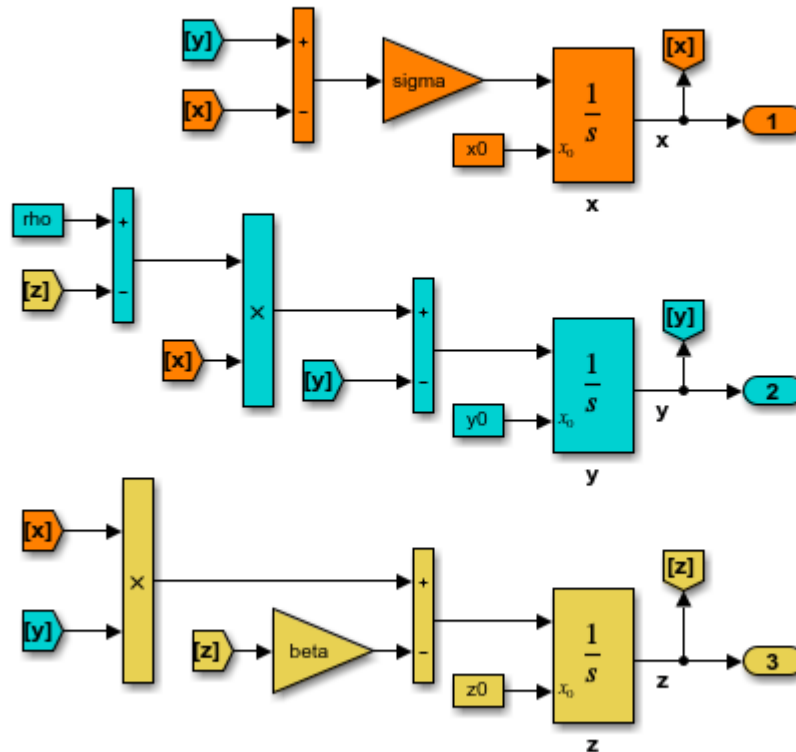
```
simulink.compiler.example.LorenzSystem
```

# 1 Functions

The screenshot shows a software interface for a project named "LorenzSystem". The interface is divided into several sections:

- PROJECT SHORTCUTS:** Contains "LorenzSystemApp" and "LorenzSystemModel".
- Views:** Includes "Files" and "Dependency Analyzer".
- Labels:** Includes "Classification".
- Table:** A table listing files with their status and classification.

Name	Status	Classification
AppStatus.m	✓	Design
LorenzAttractor.gif	✓	Other
LorenzEquations.svg	✓	Other
LorenzSystemApp.mlapp	✓	Design
LorenzSystemModel.gif	✓	Other
LorenzSystemModel.slx	✓	Design
onShutdown.m	✓	Design
onStartup.m	✓	Design



## App Details

Open the `LorenzSystemApp.mlapp` file. You can view the code written to create this app in the **Code View** section of App Designer. The essential part of building this app is the behavior of the **Simulate** button. It has the following salient parts: creating the `SimulationInput` object, configuring it for deployment, using simulation callbacks to read the output port data and plot the data at each time step. These three functions allow you to see the live results of the simulation in the deployed app.

### Create the `Simulink.SimulationInput` Object

In the function `createSimulationInput`, define an empty `Simulink.SimulationInput` object for the model. Use this `Simulink.SimulationInput` object to set simulation callbacks and variables for the model.

The simulation callback functions are used to register the callbacks. The `simulink.compiler.setPostStepFcn` function registers a callback that is invoked after every simulation step. The `simulink.compiler.setExternalOutputsFcn` registers a callback that dynamically processes the values for every output port at root level of a model during simulation.

Use the `setVariable` method of the `Simulink.SimulationInput` object to provide the parameter values to the app. Values for the simulation are obtained from the edit fields of the UI of the app. To deploy the app, use the `simulink.compiler.configureForDeployment` function. (Comment the line of code that calls `simulink.compiler.configureForDeployment` function for faster debugging.)

```
function simInp = createSimulationInput(app)
    % Create an empty SimulationInput object
    simInp = Simulink.SimulationInput('LorenzSystemModel');

    % Specify the simulation callbacks
    simInp = simulink.compiler.setPostStepFcn(simInp, @app.postStepFcn);
    simInp = simulink.compiler.setExternalOutputsFcn(simInp, @app.processOutputs);

    % Load the parameters values from the ui edit fields
    simInp = simInp.setVariable('rho', app.rhoUIC.Value);
    simInp = simInp.setVariable('beta', app.betaUIC.Value);
    simInp = simInp.setVariable('sigma', app.sigmaUIC.Value);
    simInp = simInp.setVariable('x0', app.x0UIC.Value);
    simInp = simInp.setVariable('y0', app.y0UIC.Value);
    simInp = simInp.setVariable('z0', app.z0UIC.Value);

    % Configure simInp for deployment
    % DEBUG TIP: Comment out the line below for
    % faster/easier debugging when running in MATLAB
    simInp = simulink.compiler.configureForDeployment(simInp);
end % createSimulationInput
```

### Simulation Callback Functions

The simulation callback functions register callbacks that allow you to read values from the output ports and to write values to the root input ports. These functions register callbacks at every simulation time step, which allows you to view live results of the simulation.

### The processOutputs Callback

The `simulink.compiler.setExternalOutputsFcn` line refers to the function `processOutputs`. The `processOutputs` callback function processes the values for every root output port block of model during simulation. The `processOutputs` function is called once per port and per the sample time of the port. When the `processOutputs` function is called, it reads the values for every root output block and caches those values. The `postStepFcn` obtains the cached values to update the plot.

```
function processOutputs(app, opIdx, ~, data)
    % Called during sim to process the external output port data,
    % will be called once per port per its sample hit.
    switch opIdx
        case 1
            app.txyzBuffer.x = data;
        case 2
            app.txyzBuffer.y = data;
        case 3
            app.txyzBuffer.z = data;
        otherwise
            error(['Invalid port index: ', num2str(opIdx)]);
    end
end
```

## The postStepFcn Callback

The `postStepFcn` callback function is invoked after every simulation step. The `time` argument is the time for the previous simulation step. The `postStepFcn` function obtains the cached output block values for every time and passes those values to the `updateTrace` function to plot the cached values at simulation time.

```
function postStepFcn(app, time)
    % Called during sim after each simulation time step
    app.updateSimStats(time);
    if app.status == AppStatus.Starting
        app.switchStatus(AppStatus.Running);
        app.simStats.WallClockTimeAfterFirstStep = tic;
    end
    if app.stopRequested
        app.switchStatus(AppStatus.Stopping);
        stopRequestedID = [mfilename('class'), ':StopRequested'];
        throw(MException(stopRequestedID, 'Stop requested'));
    end
    %-----
    app.txyzBuffer.t = time;
    x = [app.txyzBuffer.x];
    y = [app.txyzBuffer.y];
    z = [app.txyzBuffer.z];
    app.updateTrace(x, y, z);
    app.updateMarker('head', x, y, z);
    %-----
    drawnow limitrate;
end % postStepFcn
```

## Test in App Designer

Before deploying the application, ensure that the app runs in the App Designer. Click **Simulate** to verify that the application works by simulating the model for different values.

## Compile App for Deployment

You can use the App Designer to compile and deploy the app. You can also use the `deploytool` function. For more information on compiling and deploying with the App Designer, see [Develop Apps Using App Designer](#), [Web Apps and Application Compiler](#).

To compile the app in this example, use the `mcc` command followed by the app name.

```
mcc -m LorenzSystemApp
```

## Input Arguments

### in — Simulation inputs

`Simulink.SimulationInput` object

Simulation inputs and changes to model for simulation, specified as a `Simulink.SimulationInput` object

Example: `in = Simulink.SimulationInput('vdp')`

**@(time) postStepFcn(time) – Function handle for callback**

MATLAB function handle

Function handle for a callback that is invoked after every simulation step.

- `time` - Time for the previous simulation step, specified by a numeric value.

**See Also**

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment`

**Topics**

*"Simulink Compiler Workflow Overview"*

*"Create and Deploy a Script with Simulink Compiler"*

*"Deploy an App Designer Simulation with Simulink Compiler"*

*"Deploy Simulations with Tunable Parameters"*

**Introduced in R2020b**

# simulink.compiler.stopSimulation

Stop a long running simulation

## Syntax

```
simulink.compiler.stopSimulation('modelName')
```

## Description

`simulink.compiler.stopSimulation('modelName')` function enables you to stop a running simulation from a callback or a MATLAB app for the model specified .

## Input Arguments

**modelName** — Name of model

string

Name of model for which you want to stop the simulation, specified by a string.

## See Also

`sim` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `exportToFMU2CS` | `simulink.compiler.configureForDeployment` | **applicationCompiler**

## Topics

“Simulink Compiler Workflow Overview”

“Deploy an App Designer Simulation with Simulink Compiler”

“Generate, Modify and Deploy a MATLAB App for a Simulink Model”

**Introduced in R2020b**

