ET ROBOCON 2014

MODELING STANDARDS

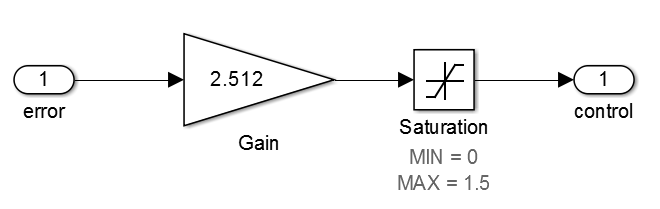
1. **GENERAL**
   1. **FILE ORGANIZATION**

Folders should all be lowercase (doesn’t really matter in Windows),   
and use underscores to separate words.

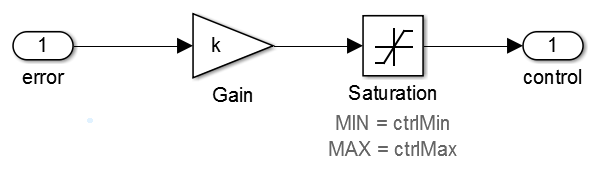
|  |  |
| --- | --- |
| **Folder** | **Description** |
| **code\_generation\_directory** | *INTERNAL:*  Derived files (MEX files, slprj folder) are placed here |
| **data\_dictionaries** | All **.sldd** files linked to your models should go here. |
| **data\_types**   * **enumerations** * **buses** * **…** | While data types should be inside the data dictionary, you can use this folder to create files containing enumerations, buses, etc. Then, import these files into the data dictionary for your final design. |
| **documents**   * **architect\_doc** * **media** * **…** | Contains System Requirements and Standards documents.  Will also contain all the files needed to create the documentation, presentations, etc. |
| **models**   * **host** * **physmod** * **NXT** * **…** | All Simulink models and libraries go here.  Add your team’s models to a subfolder. |
| **simulation\_cache\_directory** | *INTERNAL:*  Generated code is placed here |
| **templates** | Template data, models, or examples are located here. |
| **utilities**   * **+standards** * **shortcuts** * **…** | Project shortcuts, supporting files, test infrastructure, classes/packages, etc. are located here. |
| **work\_area** | Folder for making temporary files while you are working on the repository. |

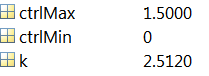
* 1. **DATA**

Avoid hard-coding parameters in models. Using descriptive parameter names can help you to remember the purpose of each parameter (i.e., to have a self-documenting model). More generally, it can make the generated code more tunable and traceable.

**BAD**  


**GOOD**

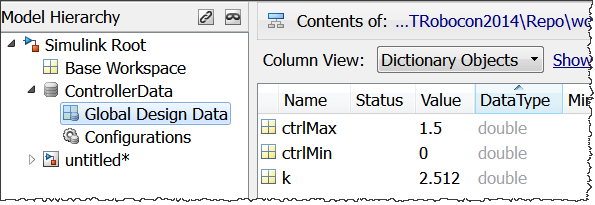




* + 1. **Data Dictionary**

Any data that isn’t hard-coded should exist in a Data Dictionary linked to the model. Data dictionaries replace the base workspace – this prevents you from accidentally clearing the base workspace or overwriting variables.

Data dictionary files (**.sldd**) should be named in **MixedCase**:



* + 1. **Parameter Variables**

All parameters should follow **camelCase**:   
no underscores, all words except the first are capitalized.

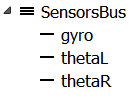
*EXCEPTION:*   
If your variable uses a matrix linked to an equation (e.g. **K**),   
or a starts with an acronym (e.g., **PWM**)

|  |  |
| --- | --- |
| **GOOD** | **BAD** |
| **gain1** | **Gain1 gain\_1  Gain\_1** |
| **controlGain** | **ControlGain controlgain  control\_gain** |
| **controlGainMatrix** | **Control\_gain\_matrix**  **Control\_GainMatrix** |

* + 1. **Buses**

All *bus data types* should follow **MixedCase**.

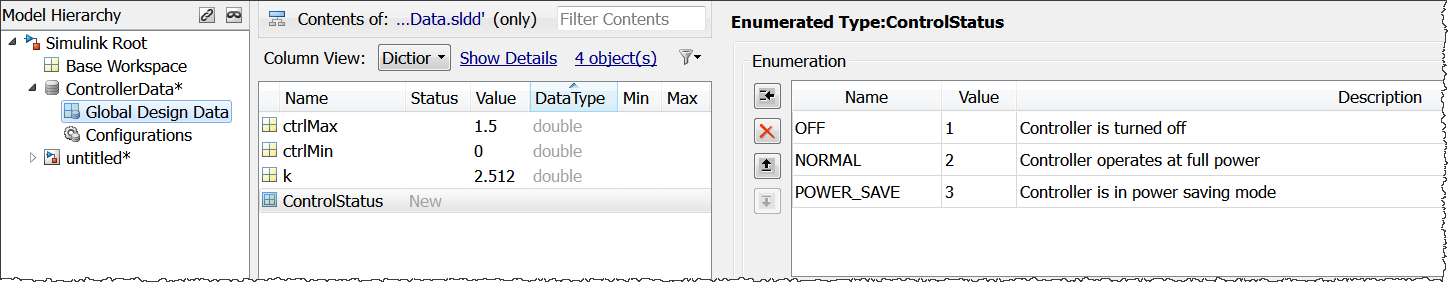
All *bus elements* should following **camelCase**.



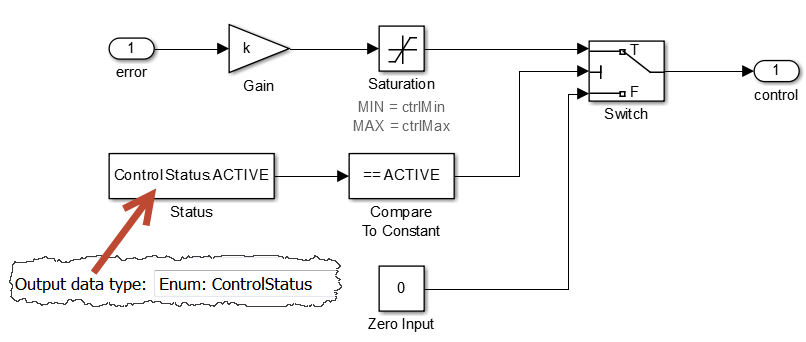
* + 1. **Enumerated Data Types**

All enumerations should be defined inside the data dictionary as shown below.

Enumeration data types should be in **MixedCase**.  
Enumeration values should be in **ALL\_CAPS\_WITH\_UNDERSCORES**.



Usage example of enumerations in Simulink:



* + 1. **Classes**

If you have any MATLAB classes or **Simulink.Variant** objects:

* use **MixedCase** for the class name
* use **camelCase** for properties and methods

**>> MyClass.property1 = 2;**

**>> MyClass.getProperty1();**

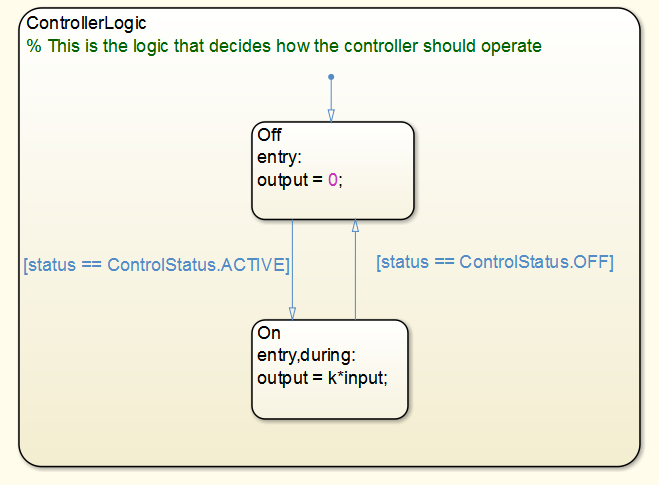
**>> SimVariant = Simulink.Variant('simMode==1');**

* 1. **STATEFLOW**
     1. **State naming and convention**

All states in your Stateflow chart should be named with **MixedCase**.

Actions should be in a separate line from state keywords   
(**entry**, **during**, etc.)

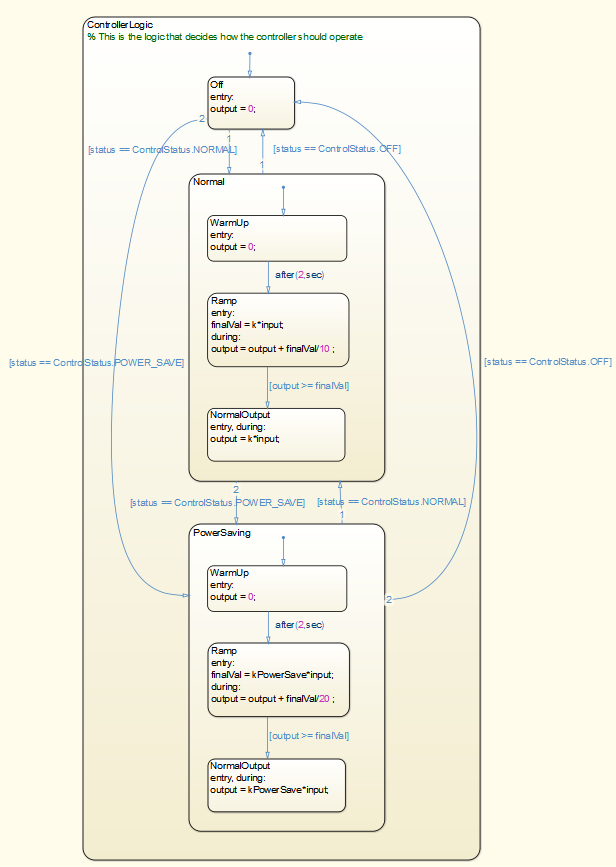
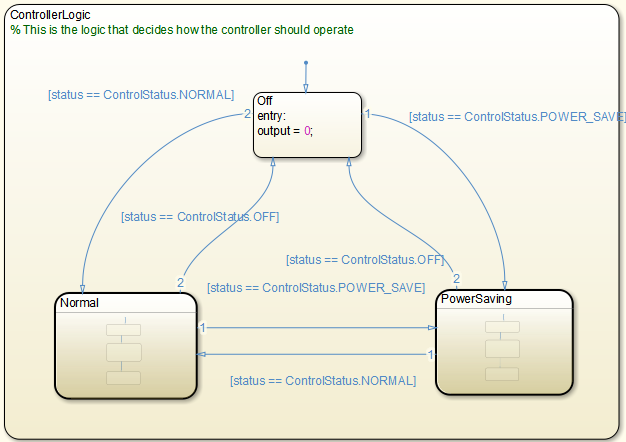
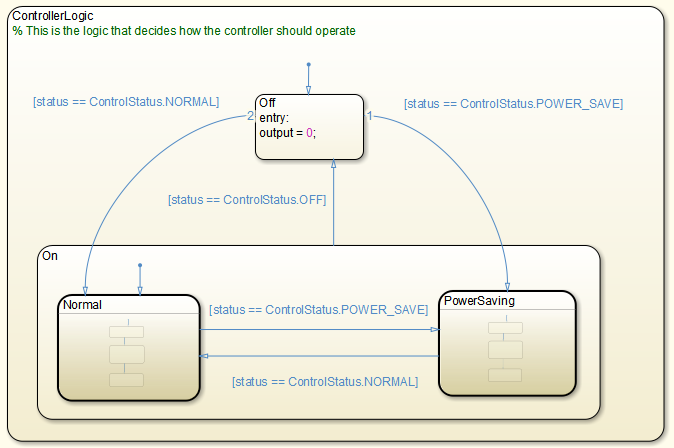
Example of state naming and enumeration usage in Stateflow:



* + 1. **Layout**

Make sure the chart logic flows from top to bottom. Left/right real estate should be used for decision branching.

Make sure your chart fits inside the model window at 100% zoom.   
If you have many states, and they share similar functionality, consider grouping them into subcharts and creating visual hierarchy.



**BETTER**

**BAD**

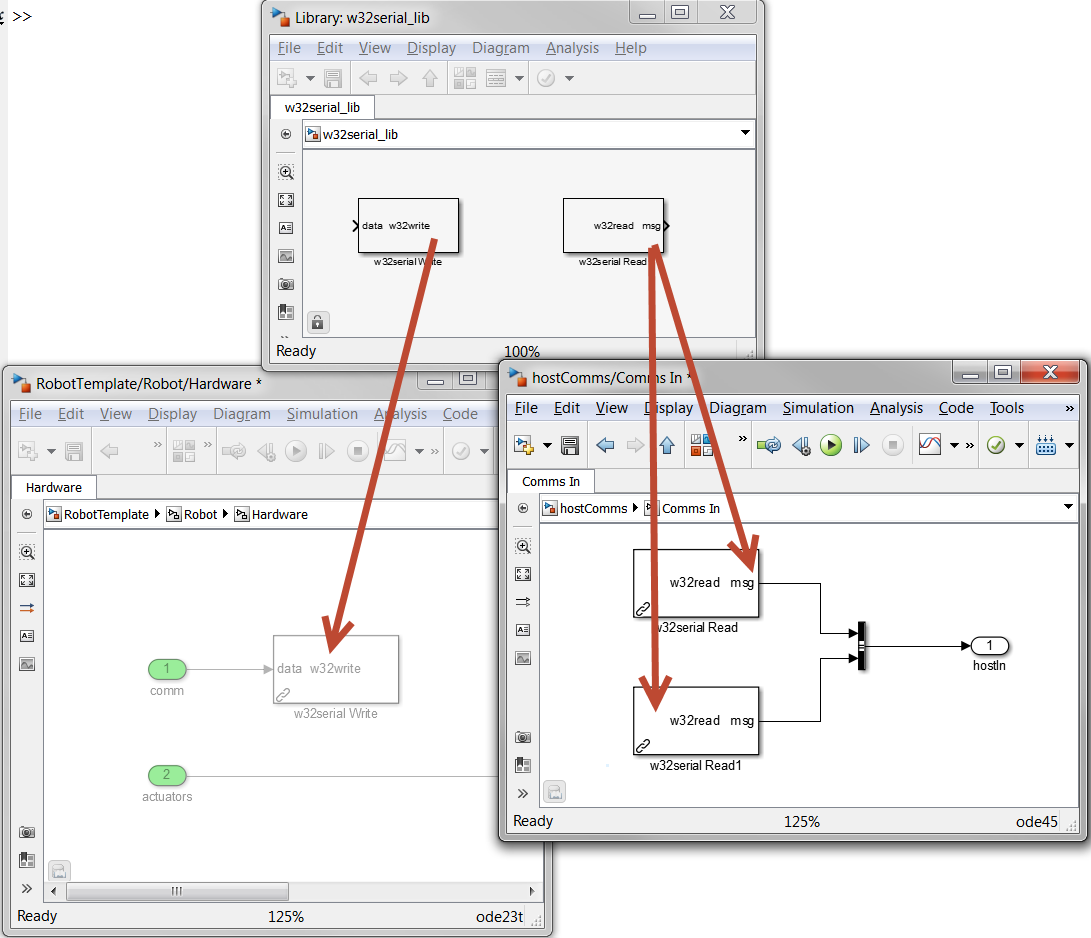
**GOOD**

* 1. **UTILITIES**
     1. **Libraries**

Any common utilities used in separate models should be placed in libraries.

What distinguishes a library from a model reference is that a library block does not have to work standalone – it is made solely to be plugged in as a utility.

Example: Serial communication blocks being used in host and robot hardware models.

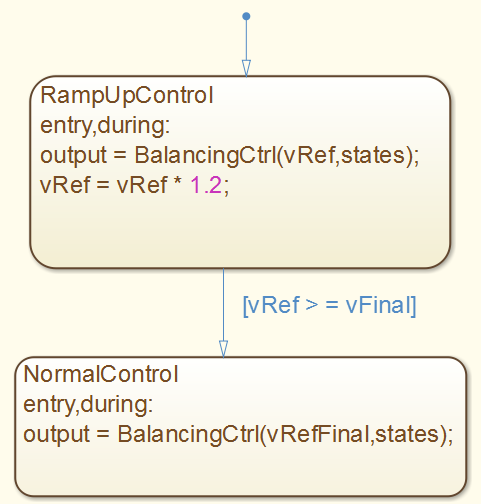
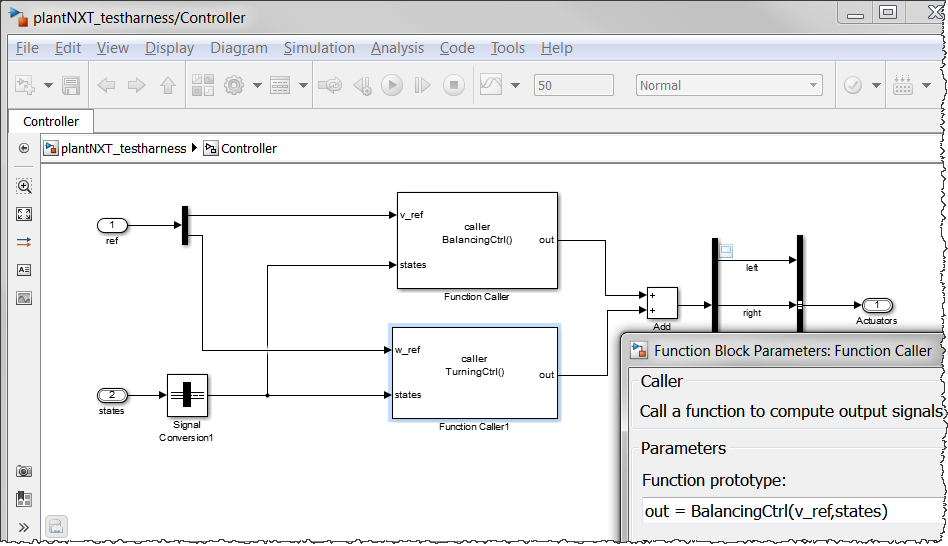


* + 1. **Simulink Functions**

If you plan on using the same utility in both Simulink and Stateflow environments, use Simulink Function blocks. These blocks can be called in Simulink (via Function Caller blocks) or in Stateflow (by just typing the function call).

You must place the Simulink Function block at the top (root) level of your model. If you plan on using the block in multiple models, you can store it in a library.

Example: Controller used in a robot model’s Stateflow controller, and in a plant model test harness.

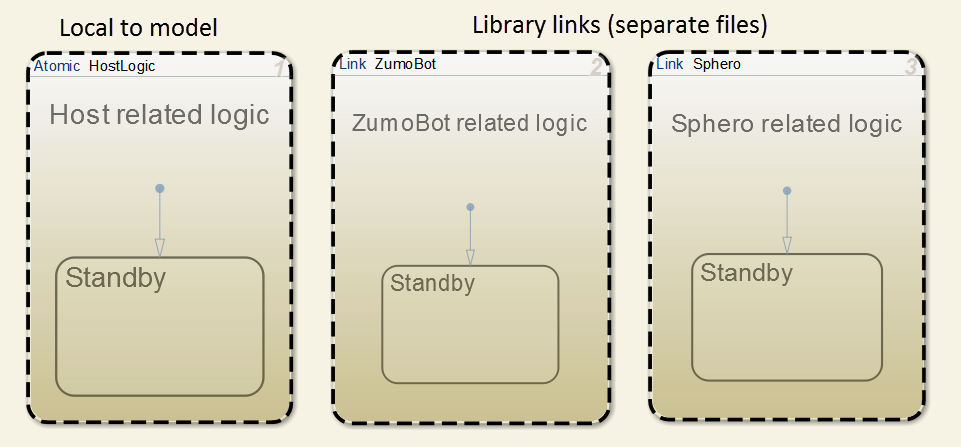
 

* + 1. **Atomic Subcharts**

When modeling in Stateflow, use atomic subcharts in libraries. This lets you:

* Create reusable Stateflow components
  + Many instances in one model
  + One or more instances in multiple models
* Create unit testable Stateflow components
* Work concurrently with other people on different elements of the chart

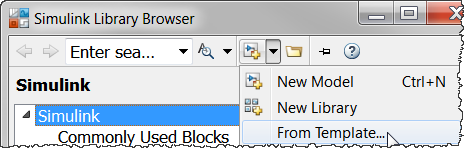
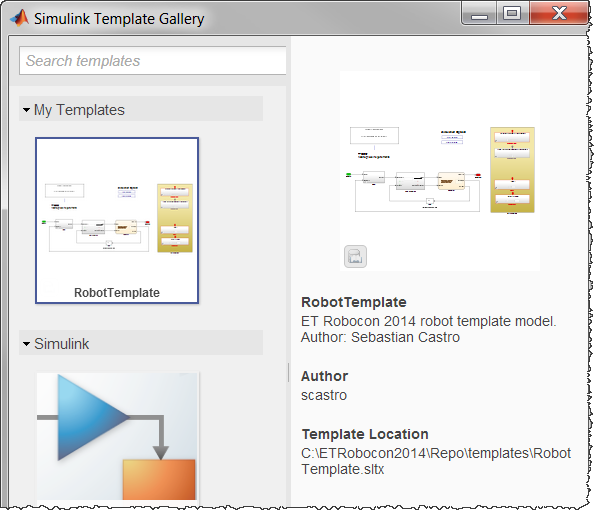
Example: The host model algorithm needs to have 3 parallel components for Sphero processing, Zumobot processing, and communications processing.



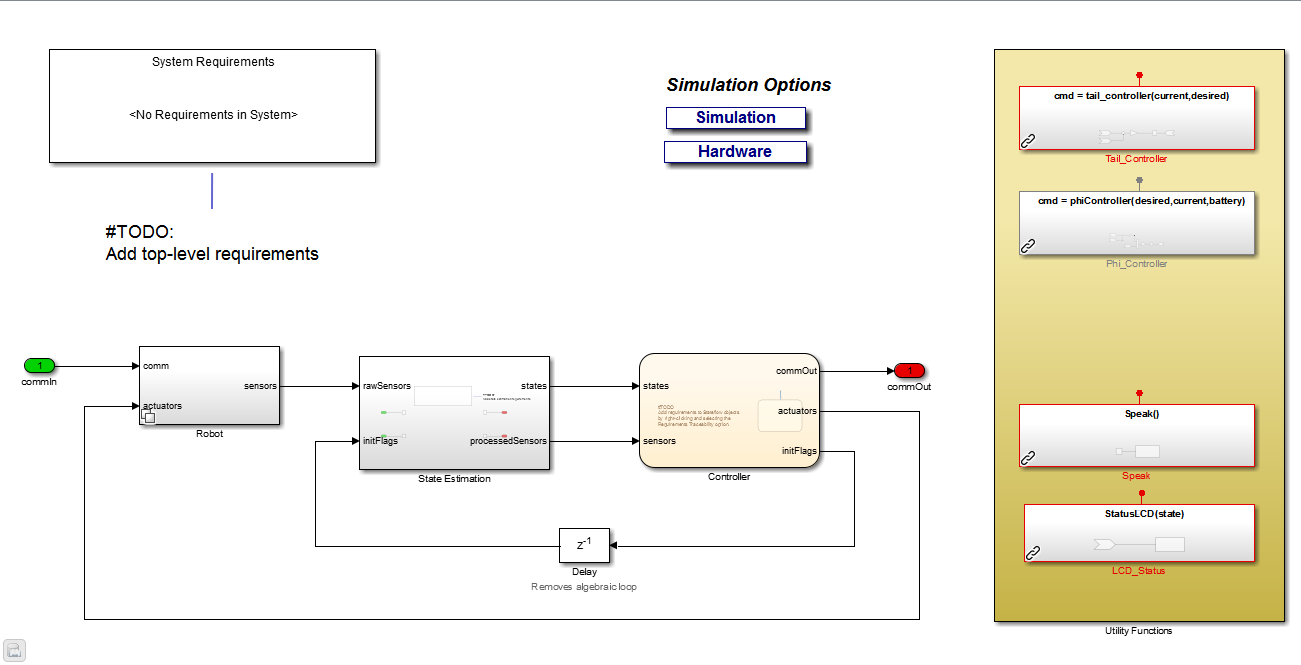
1. **ROBOT MODEL**

There will be one robot model for each different type of target hardware (NXT, Hydra, Zumobot, etc.). To ensure consistency, you can create your robot model from the **RobotTemplate.sltx** model template.

This will automatically create a data dictionary (<***modelname>*Data.sldd**) based on a template. However, you are expected to modify the model and data to suit your needs.

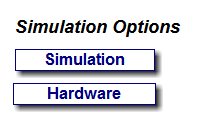
   
  
 

The model contains two root-level ports (1 Inport, 1 Outport). These blocks will serve as the Bluetooth communication interfaces when you connect the simulation plant variant to the host model. In hardware, you can ignore these I/O ports and use the appropriate driver blocks for communications.



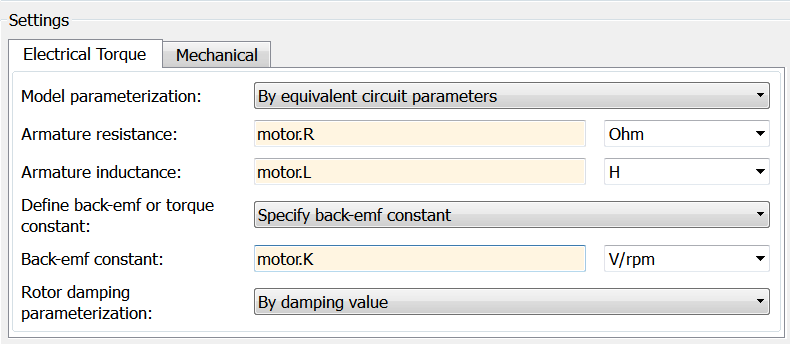
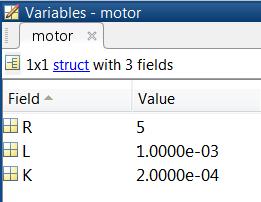
* 1. **PLANT**

You should have simulation and hardware variant subsystems for your plant model. By default, we provide two skeleton models whose variants can be overridden by the **simOption** variable, or by using the clickable annotations:



**NOTE:**  If you plan on reusing your plant models, we recommend that you place them in libraries and copy the library blocks into the Variant Subsystem block.

Try to group plant data into structures to reduce the number of variables in the data dictionary. For example, you can group variables by functionality:



* 1. **STATE ESTIMATION**

This block will convert the raw sensor data (type **SensorsBus**) to estimated state data (type **StatesBus**). You will then typically use the states for control.

The goal is to use the same State Estimation block regardless of which plant variant is present. Therefore, this block is local to the robot model and you should modify it there.

* 1. **CONTROLLER**

The controller is a Stateflow chart which:

* Will contain the bulk of your robot algorithm, if processing is done on the hardware)
* Will contain some basic functionality, if processing is done on the host. The bulk of the robot algorithm will then be associated with the host model Stateflow chart.

The provided interface specification is shown below:

INPUTS

|  |  |  |
| --- | --- | --- |
| **Input Name** | **Data Type** | **Description** |
| **states** | **StatesBus** | Estimated states used for control. |
| **sensors** | **SensorsBus** | Raw sensor values from hardware or simulation. |

OUTPUTS

|  |  |  |
| --- | --- | --- |
| **Output Name** | **Data Type** | **Description** |
| **control** | **ControlBus** | Control inputs to the robot. For the NXT, these are PWM values (0-100) for the left, right, and tail motors. |
| **initFlags** | **InitBus** | Typically zero; set values to 1 for a single time step to re-initialize state estimation integrators (encoder/gyro) to zero. |
| **commOut** | Numeric(depends on communication spec) | Used for Bluetooth Communications. |

EXECUTING TASKS

* + 1. TASK LIST  
       You should create a list of discrete tasks for your robot. Each discrete task can then have its own algorithm represented by an atomic subchart.  
         
       This list of tasks can be changed through a variable named **taskList**. This is an array of **TaskEnum** enumerations.  
         
       Example:  
         
       **taskList = [ TaskEnum.GYRO\_CALIB ...**

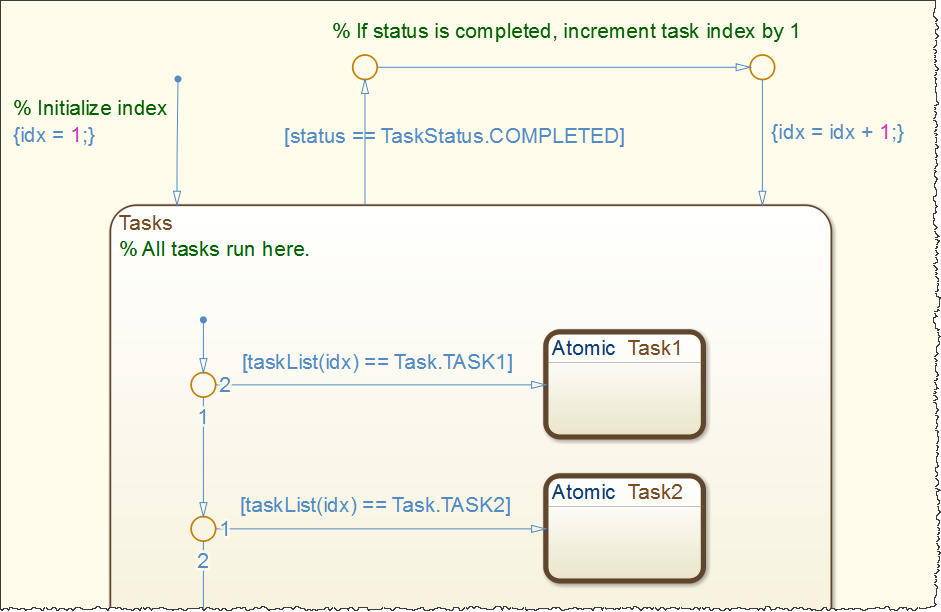
**TaskEnum.LINE\_TRACK ...**

**TaskEnum.PARKING ];**

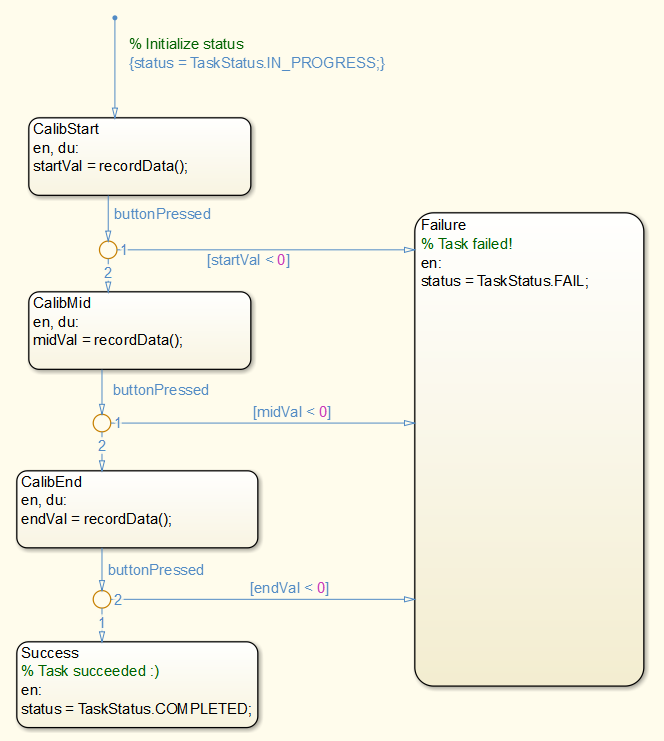
* + 1. STATUS REPORTING  
         
       Each task’s status should be assigned to a variable named **status**, of **TaskStatus** enumerated type.

As long as the task status value is **IN\_PROGRESS**, the state chart will remain in that task. When the task is completed, it must broadcast its status as **COMPLETED**. This enables it to move to the next task on the list.

* + 1. STATEFLOW REPRESENTATION  
         
       The **status** variable should be a Data Store Memory scoped variable to communicate data across atomic subchart boundaries.  
         
       Example Stateflow controller (and atomic subchart task):

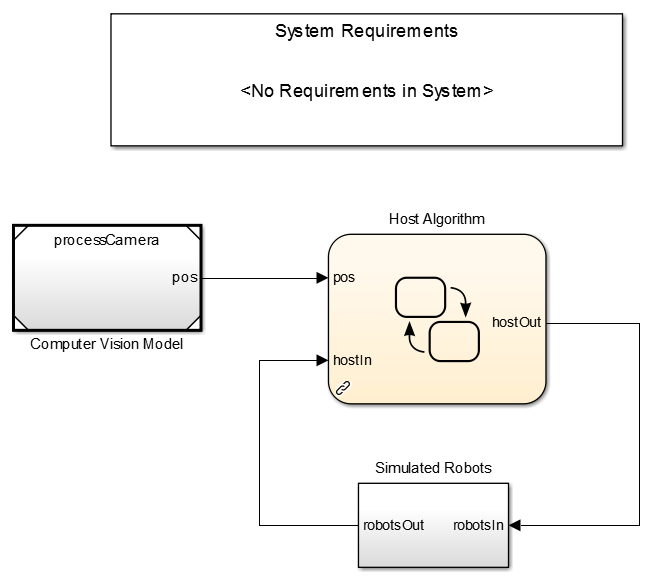


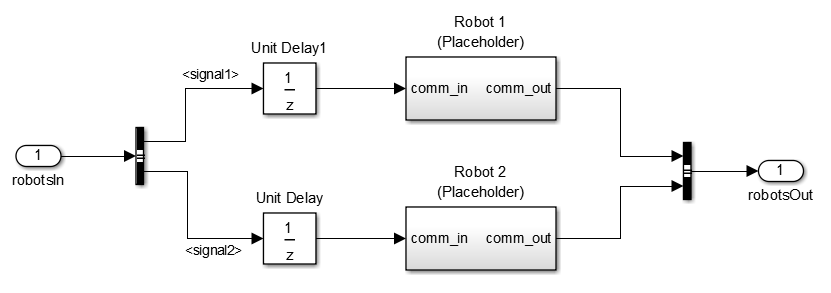




**Task1**

1. **HOST MODEL**
   1. **ARCHITECTURE**
      1. **SIMULATED VERSION**The simulated version is located in **models\host\hostSim.slx**.  
         Each robot model is connected in a closed-loop configuration, using model referencing. The robot plant variant subsystems are set to simulation variants. Communication values are passed in and out of the robot models via the root level I/O ports in each model.

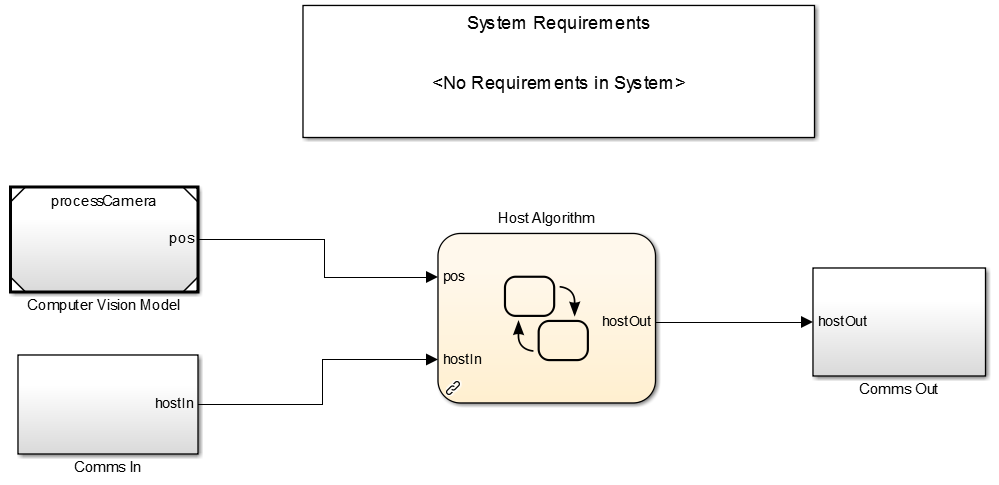




**Reference to robot model**

* + 1. **STANDALONE**

The standalone version is located in **models\host\hostComms.slx**.  
The robot models also run standalone, and communication is done via the Bluetooth driver blocks, as necessary.



* 1. **COMPUTER VISION**

The CV algorithm is its own model, and can be referenced from both the standalone and simulation host models. It should also be compatible for unit testing in a harness model.

For testing purposes, this model can switch between video file reading and live streaming.

INPUTS

*<none>*: The model itself streams the video data.

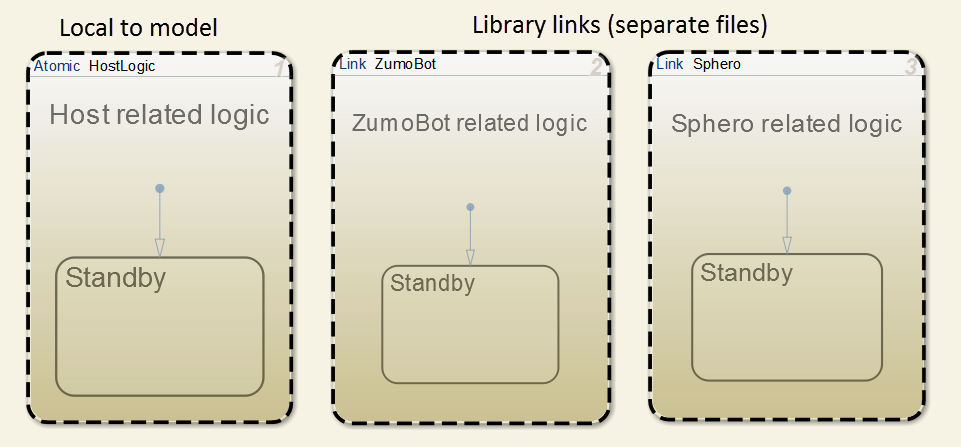
OUTPUTS

|  |  |  |
| --- | --- | --- |
| **Output Name** | **Data Type** | **Description** |
| **pos** | **PositionBus** | Contains positions for all robots being tracked in the computer vision algorithm. |

* 1. **HOST ALGORITHM**

The host algorithm is contained in a library and is used in both the standalone and simulated host model.

This algorithm contains parallel states for Sphero processing, Zumobot processing, and communications processing. Each parallel state will be an atomic subchart to facilitate concurrent development. The ZumoBot and Sphero atomic subcharts are library linked so their algorithms exist in a separate model.



INPUTS

|  |  |  |
| --- | --- | --- |
| **Input Name** | **Data Type** | **Description** |
| **pos** | **PositionBus** | Contains positions for all robots being tracked in the computer vision algorithm. |
| **hostIn** | **CommsBus** | Contains the numeric communication values from each robot, in whichever format is necessary given the communications spec. |

OUTPUTS

|  |  |  |
| --- | --- | --- |
| **Output Name** | **Data Type** | **Description** |
| **hostOut** | **CommsBus** | Contains the numeric communication values to send to each robot, in whichever format is necessary given the communications spec. |

ATOMIC SUBCHARTS

Data for atomic subcharts should be mapped as needed.