**SIL Simulink Setup Guide**

# Introduction

This guide is intended to guide Simulink developers in the implementation of the SIL for an existing model.

The first section details the steps needed to install the SIL and set it up for use by a model, at which point the will successfully generate the SIL-produced ECI interface header file. The second section documents the steps needed to implement the various application interfaces supported by the SIL.

## Format

In this guide, executable Matlab commands are placed on their own line and are preceeded by ‘>>’. As an example:

>> disp(‘Hello World’)

# setup

This section details the steps needed to setup a model to use the SIL.

## Installation

Make sure Simulink directory of this repo is on matlab path. Two methods are available:

Per-User installation:

* Pros:
  + Simple
  + One-time setup
* Cons:
  + Only one installation can be present on a computer at a time
  + Can be a problem if SIL is upgraded during project development… can’t use version that code was developed with

1. Place at fixed location on computer (ex, Documents/Matlab directory)
2. Run cfs\_add\_path() from matlab command line to permanently add the directory to the default Matlab path (ie path does not need to be added on Matlab restart)

Per-Project installation:

* Pros:
  + Can be CM’d with the project files
  + Can use multiple versions with different projects on same computer
* Cons:
  + Must be added to path each time Matlab is started/work on project begins

1. Place in directory structure for project
2. Make sure it gets added to path whenever project is initialized (and that it’s the only copy on the path)
   * Can be accomplished by init scripts for the particular project
   * Can be done per-user via startup.m (see docs)

## Model Setup

The SIL uses a custom TLC target to generate the extra header file which contains the interface information. In order to use this:

* Open model, Open model’s configuration parameters (Gear icon in ribbon)
* Select Code Generation tab (on left)
* Set the System Target File to “cfs\_ert.tlc” using the browse menu to the right
  + Note: If you can’t find the option in the browser, check that the Simulink directory is on the matlab path
  + You may also accomplish this by running the following command (while your model is open):

>> set\_param(bdroot,’SystemTargetFile’,’cfs\_ert.tlc’)

* Note that setting the cfs\_ert target will create a new configuration tab “CFS Target Settings” which containing settings to customize the code generation process.

## Generating Code

To generate code containing the new ECI interface header, use one of the following methods:

* Select Code > C/C++ Code > Build Model
* Click the ‘build’ button on the toolstrip
* Use the keyboard shortcut, ‘Ctrl-B’
* Use the following command (while the model is open):

>> rtwbuild(bdroot)

# Model Updates

The following section details the steps needed to specify the application interfaces for the SIL.

These instructions assume that the inputs and outputs of the model are defined using buses and that the parameter values are organized in structures in the base workspace.

If using a model reference hierarchy, note that the interfaces specified below must be implemented in the root of the model hierarchy. In most cases these interfaces will be ignored if implemented in child models.

## Setup input packets

## Define the Signal Object

cfsPackage.Signal is a thin wrapper of the Simulink.Signal object, which is an object which allows associating additional information with a signal for use in the code generation process. The SIL detects instance of the cfsPackage.Signal object used in the model and creates an interface to read that bus from the CFE software bus.

### In the model’s initialization script, create a cfsPackage.Signal object:

>> myMsg = cfsPackage.Signal();

### Next, specify the type of message, either ‘cfsTlmMessage’ or ‘cfsCmdMessage’:

>> myMsg.DataType = 'Bus: myBusType';

>> myMsg.CoderInfo.StorageClass = 'Custom';

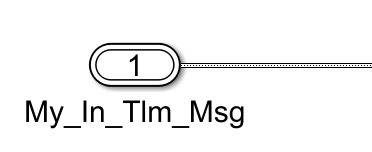
>> myMsg.CoderInfo.CustomStorageClass = 'cfsTlmMessage';

The type of message controls the handling of the message by the SIL (ie, what header is expected on the packet).

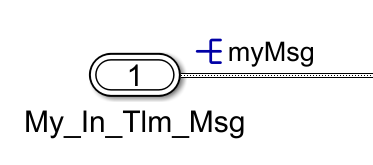
Run the initialization script and ensure that myMsg is defined in the Matlab workspace.

## Associate the Signal Object

The next step is to associate the signal object defined in the workspace with a signal in the model. In the model, right click on the bus signal exiting a root level outport and select ‘Properties’.



Enter the name of the signal previous defined (ie, ‘myMsg’) and then check the box labeled ‘Signal name must resolve to Simulink Signal object’ option. You should see a ‘pitchfork’ symbol appear next to the name of your signal, as shown below:



This bus is now configured as an input to the cFS application.

These changes have no effect on the behavior of the Simulink simulation. The simulation must ensure that the sample time/contents of the signal is appropriate for the data contained within the packet.

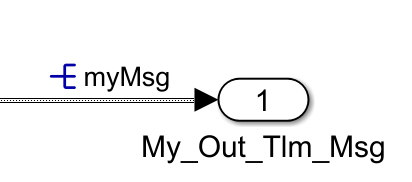
## NonVirtual Buses

All messages must be nonvirtual when they cross the app interface (ie, at the inport/outport). Nonvirtual buses can be identified by the line-dot-line signal shown above. Virtual buses are marked by a triple-line and don’t exist in the generated code (they’re a graphical grouping convenience tool), and therefore cannot be setup as input/output message.

## Setup output packets

This process is identical to setting up input packets with the exception that the cfsPackage.Signal object must be applied to a signal which is an input to a root level OutPort block.

Repeat the steps for defining a signal object in the base workspace (ensuring that it has a unique name) and repeat the steps to associate it with a signal in the model.



Note that there is no difference between and input packet and an output packet in the definition of the signal object… the SIL identifies a signal as either an input packet or output packet by if its connected to an Inport block (in which case it’s an input packet) or an Outport block (in which case it’s an output packet).

A signal object may not be used multiple times within a model… each input and output packet require its own cfsPackage.Signal object.

These changes have no effect on the behavior of the Simulink simulation.

## Conditional Message

The ‘CFS\_Conditional\_Msg’ block (in cfs\_library.slx) allows controlling when an output message will be issued. By default, output packets are always issued. The Conditional message block takes a Boolean input signal and will only issue the message when that signal is true.

In simulation, the Conditional message block will hold the value of the output bus when the flag is false, to simulate not issuing a packet.

Note that the Conditional Message Send block may not pass through any other blocks before terminating at the root level Outport.

## Setup parameter tables

cfsPackage.Parameter is a thin wrapper of the Simulink. Parameter object, which is an object which allows associating additional information with a signal for use in the code generation process. The SIL detects instance of the cfsPackage.Parameter object used in the model and creates a table interface for that structure in the generated code.

Define a bus definition matching the definition of the parameter structure. For this example, its assumed that the parameter structure is called myStruct and that the corresponding bus definition myStructBusName exists.

>> myParamTbl = cfsPackage.Parameter(myStruct);

>> myParamTbl.DataType = 'Bus: myParamTableBus';

>> myParamTbl.CoderInfo.StorageClass = 'Custom’;

>> myParamTbl.CoderInfo.CustomStorageClass = 'cfsParmTable’;

If desired, the name of the file the table is generated into can be controlled by setting the DefinitionFile attribute, as shown:

>> myParamTbl.CoderInfo.CustomAttributes.DefinitionFile = 'myParamTbl.c’;

This is generally necessary if using the elf2tbl tool.

Then, using myParamTbl.Gain in the code will yield the value of the parameter.

These changes have no effect on the Simulink simulation.

## Setup events

The CFS\_Event block included in cfs\_library allows issuing CFS events from within autogenerated code. By default block takes a single Boolean flag as input and will issue the event specified by its mask parameters when the signal evaluates to true. The block also has the ability to include values from signals in the generation of the event message.

The block mask requires the specification of an event ID, event type, event mask, and event message. The event ID must be a unique integer and will be used as a handle to register this event with CFE event services. The event type may be one of ‘Info’, ’Warning’, or ’Error’. The event mask allows the CFE to filter events which may be issued multiple times. The event message is a fprint-style format string which may contain ‘%f’ specifiers to include the value of data signals passed to the block. The format specifier will be replaced with the value of the signal during event generation.

To implement event generation, retrieve a ‘CFS\_Event’ block from cfs\_library.slx and place it in your model. Fill out the fields of the block mask and connect the flag and data signals.

This block has no effect in the Simulink simulation.

## Setup FDC

Retrieve a ‘CFS\_Status\_Flag’ block from cfs\_library.slx and place it in your model. Fill out the fields of the block mask and hook up a Boolean signal to its input.

This block has no effect in simulation.

## Setup time

Retrieve a ‘CFS\_Time’ block from cfs\_library.slx and place it in your model. In simulation mode, this block operates as a digital clock with the addition of the epoch specified in the block mask. The output of this block is a double containing simTime+Epoch.