

MathWorks
**AUTOMOTIVE
CONFERENCE
2018**

What's New for Automotive

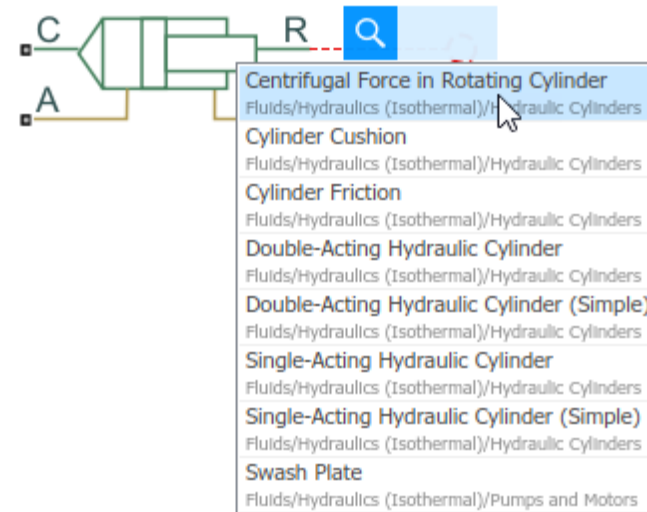


What's New for *Simulink* in R2018b

Predictive Quick Insert

Connect a recommended block to an existing block in your model, sorted by frequency of use

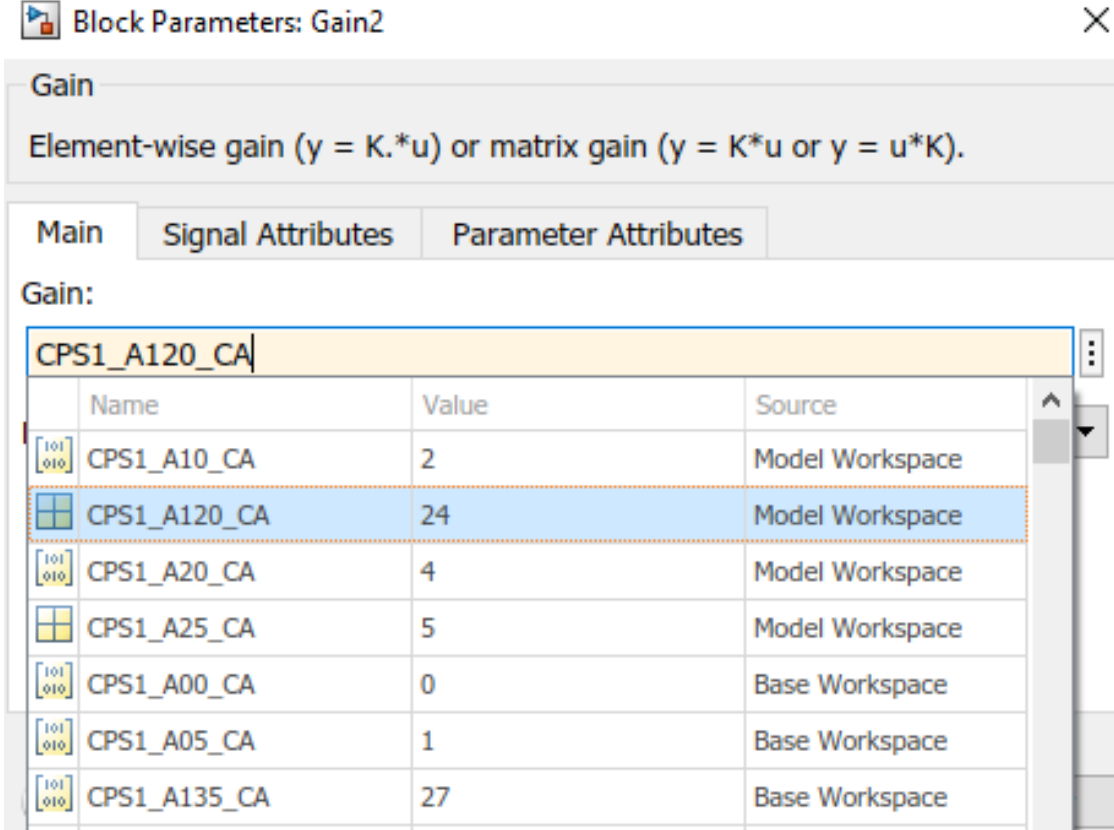
- Speed up model creation by limiting your search to only those blocks that can be connected to existing block ports
- Recommendations based on the example models included in your installation and can be customized with your own models
- Take advantage of the knowledge contained in your organization's existing working library of models



Block Parameter Autocomplete

Improve speed and accuracy of block parameter editing by selecting from suggested variable or function names as you type

- Displays variables from visible workspaces and dictionaries
- Provides additional information in a table to help with selecting a particular entry
- Use in both block dialogs and Property Inspector










Block Parameters: Gain2

Gain

Element-wise gain ($y = K.*u$) or matrix gain ($y = K*u$ or $y = u*K$).

Main Signal Attributes Parameter Attributes

Gain:

	Name	Value	Source
	CPS1_A10_CA	2	Model Workspace
	CPS1_A120_CA	24	Model Workspace
	CPS1_A20_CA	4	Model Workspace
	CPS1_A25_CA	5	Model Workspace
	CPS1_A00_CA	0	Base Workspace
	CPS1_A05_CA	1	Base Workspace
	CPS1_A135_CA	27	Base Workspace

Direct Editing on Icon

Edit block parameters and subsystem port labels directly in the Simulink Editor

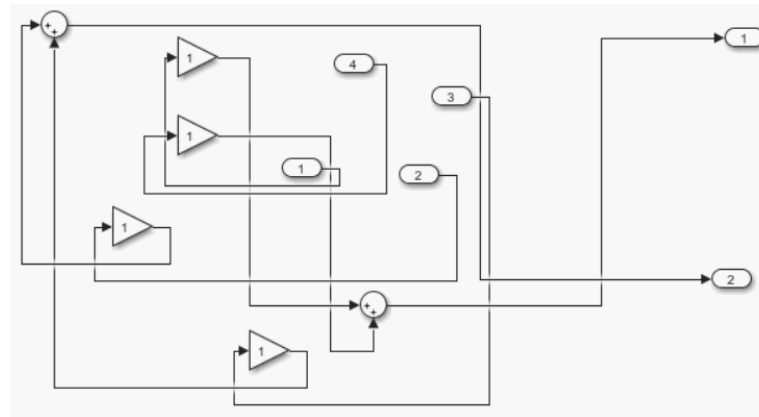
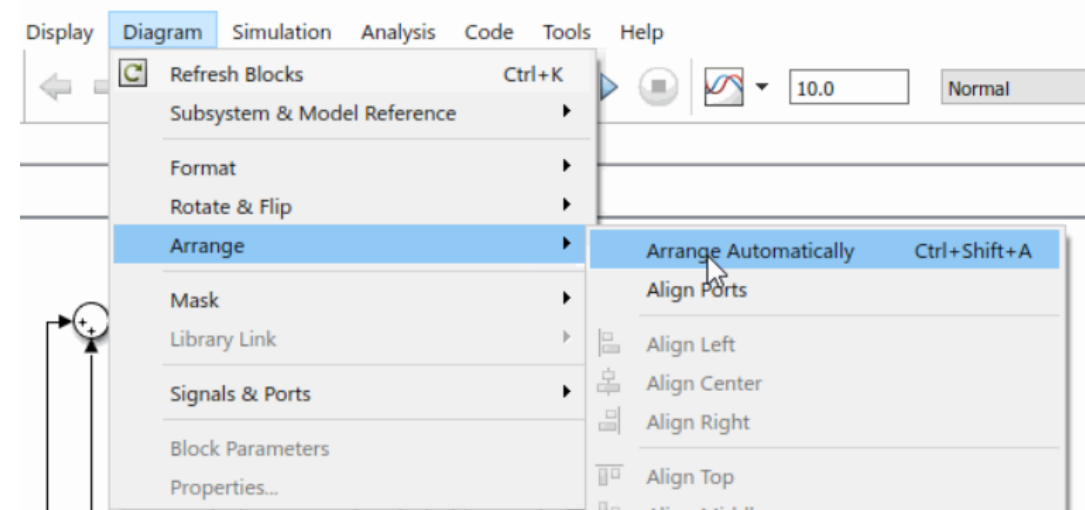
- Change a subset of block parameters directly on the block mask
- Works with the following blocks: Constant, From Spreadsheet, From Workspace, To Workspace, From File, To File



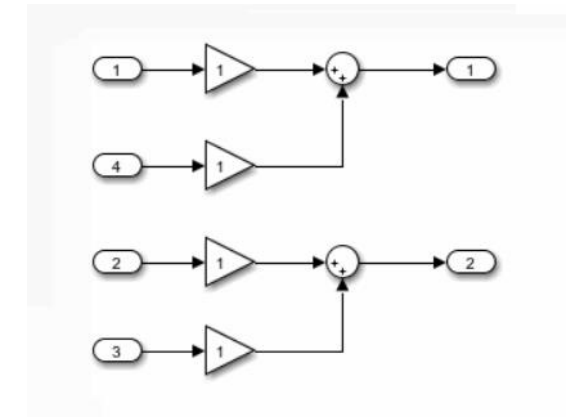
Arrange System

Automatically move and resize the blocks in your model to create cleaner looking diagrams

- Produces a left to right flow of signal information
- Resizes blocks such as gains so that variable names can be clearly seen



BEFORE



AFTER

Archiving runs in Simulation Data Inspector

Focus on your current simulation while retaining older runs

- Plots automatically update to show current run at start of simulation – similar to how scopes behave
- All runs available in a new archive region at the bottom
- Option to limit disk space used by configuring maximum number of runs to archive

The screenshot displays the Simulation Data Inspector (SDI) interface. The main window shows a plot of EngineRPM with a peak around 4000. The left sidebar contains a list of signals under 'Run 4: sldemo_autotrans : Coasting [Current]'. The 'Archive' dialog box is open, providing information about the archive feature and configuration options.

Archive

The archive is a place to store prior runs. When a model simulates, the prior run is automatically moved to the archive.

Auto-archive simulation runs

Number of runs to archive:

All runs

Last runs

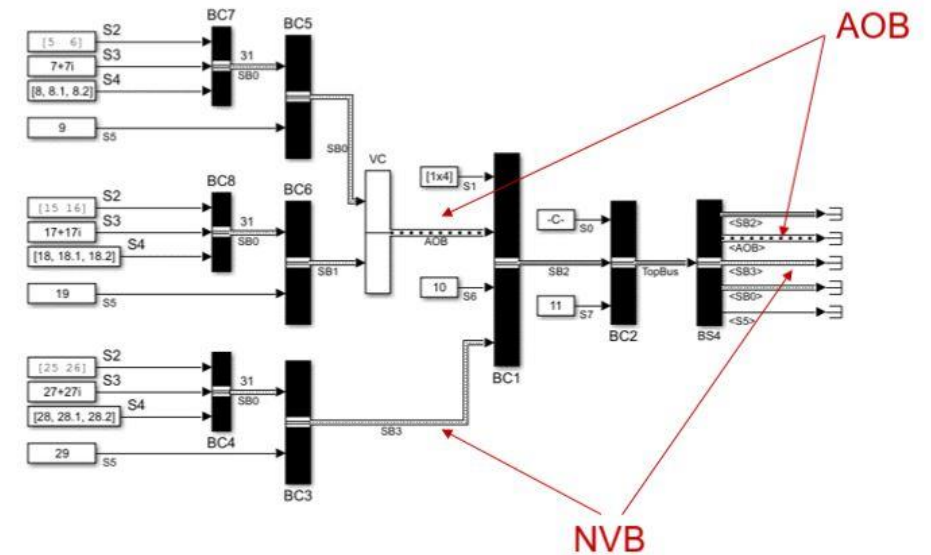
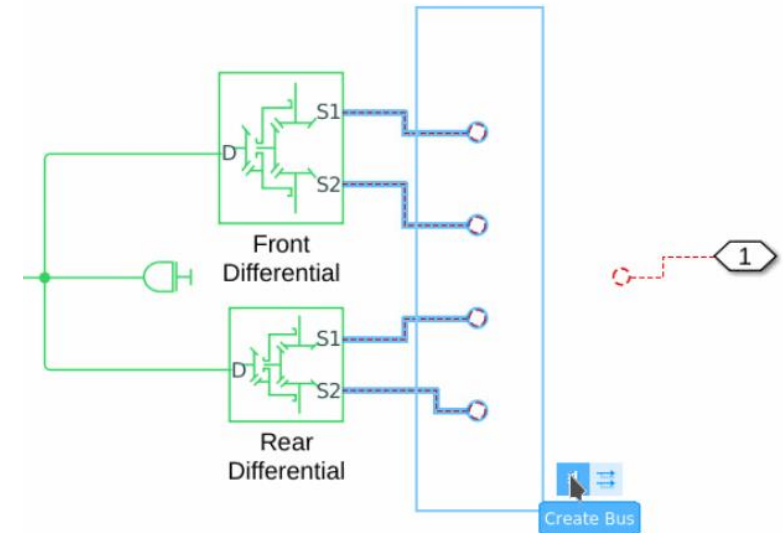
[Help with archiving](#)

The background interface shows the 'Inspect' tab with a search bar and a 'Compare' button. The signal list includes: EngineRPM (checked), Throttle, BrakeTorque, ShiftLogic:1, ImpellerTorque, OutputTorque, VehicleSpeed, and TransmissionRPM. The 'Archive' region at the bottom shows three previous runs: Run 1: sldemo_autotrans : Passing Maneuver, Run 2: sldemo_autotrans : Gradual Acceler..., and Run 3: sldemo_autotrans : Hard braking.

Buses

Create buses from Simscape physical connection lines and retain arrays of buses and nonvirtual buses within virtual buses

- Package multiple Simscape connection lines together into a bus
- Retain non-virtual buses within a virtual bus.
- No need to convert all to virtual or non-virtual buses, resulting in less data copies in generated code



Execution Domain Specification

Improve your solver performance and code by specifying the domains of subsystems within your model

- Improve simulation speed and code quality by reducing number of hybrid systems and solver switches in your model
- Set a subsystem to discrete quickly and enforce that it remain discrete
- Automate frame sizing, execution scheduling, and multi-threading acceleration for DSP subsystems with Dataflow
- Enforce discrete library blocks in Simulink to always have discrete rates

The screenshot displays the Simulink environment for a 'fuel_rate_control' subsystem. The main workspace shows a block diagram with the following components: 'sensors', 'control_logic' (containing 'es_l', 'es_o', 'O2_normal', and 'fuel_mode' blocks), 'airflow_calc' (containing 'O2_normal', 'fuel_mode', and 'fb_correction' blocks), and 'fuel_calc' (containing 'fuel_mode' and 'fb_correction' blocks). A 'validate_sample_time' block is also present. The 'Property Inspector' window is open on the right, showing the 'Domain' property set to 'Discrete'. A red circle highlights the 'Property Inspector' window title bar. A blue box at the bottom left points to a badge icon in the Simulink toolbar with the text 'Click badge to open Property Inspector'. Another blue box at the bottom right contains the text 'Use Property Inspector to config the execution domain'.

Batch Simulations

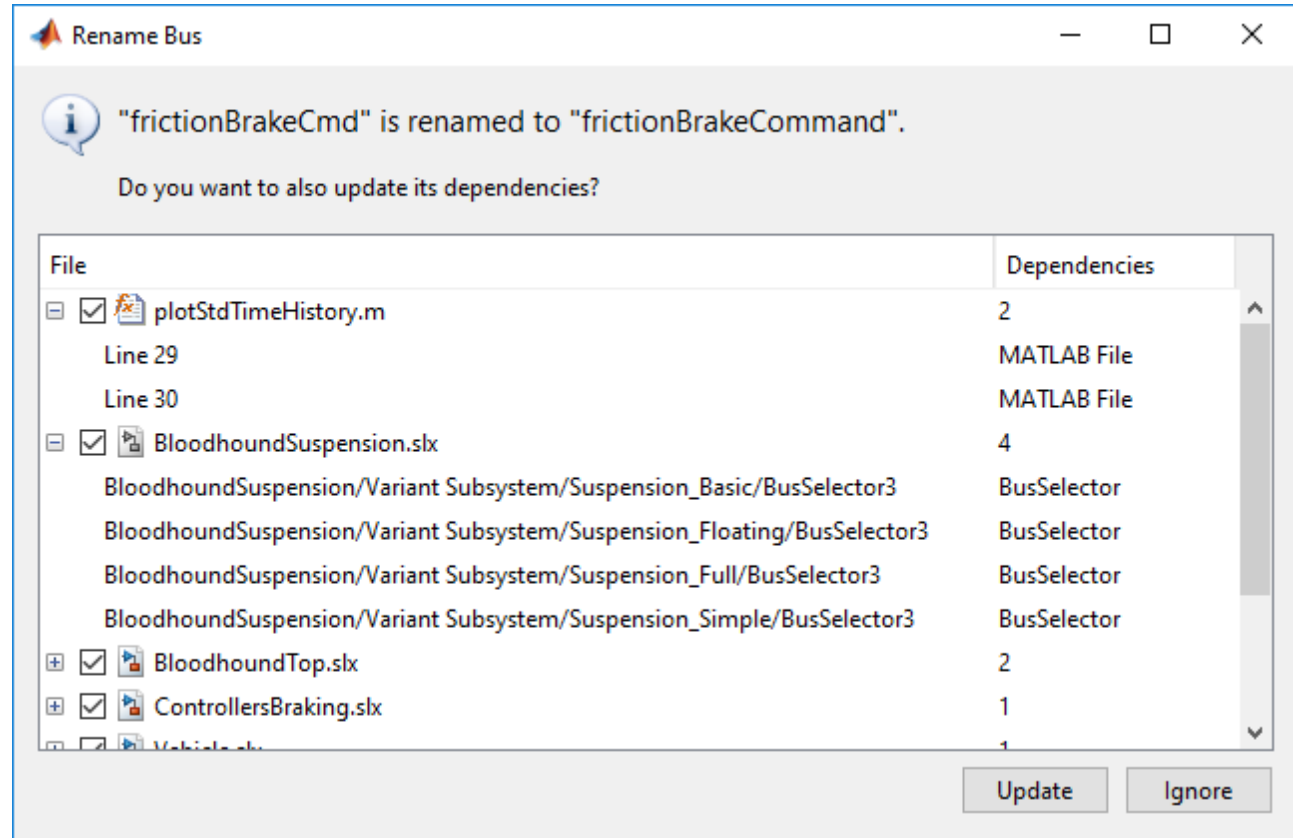
Offload execution of simulations to run in the background by using `batchsim`

- Run parallel simulations in batch mode
- Similar to `parsim`, `batchsim` automatically takes care of parallel configuration steps for you
- `batchsim` is interchangeable with `parsim`

Project-wide Bus Renaming

Automatically update all bus references across Simulink Project when you rename a bus or bus element

- Find all usages of a bus or bus element in a Simulink Project
- Automatically update those usages when renaming
- Prompts to update when renaming in the Bus Editor



Model Comparison

Use custom filters to simplify and focus model comparisons

- Create custom block and parameter filters for the comparison report
- Export and import filters to share them with colleagues

New Filter

Name: My Custom Filter

Rules:

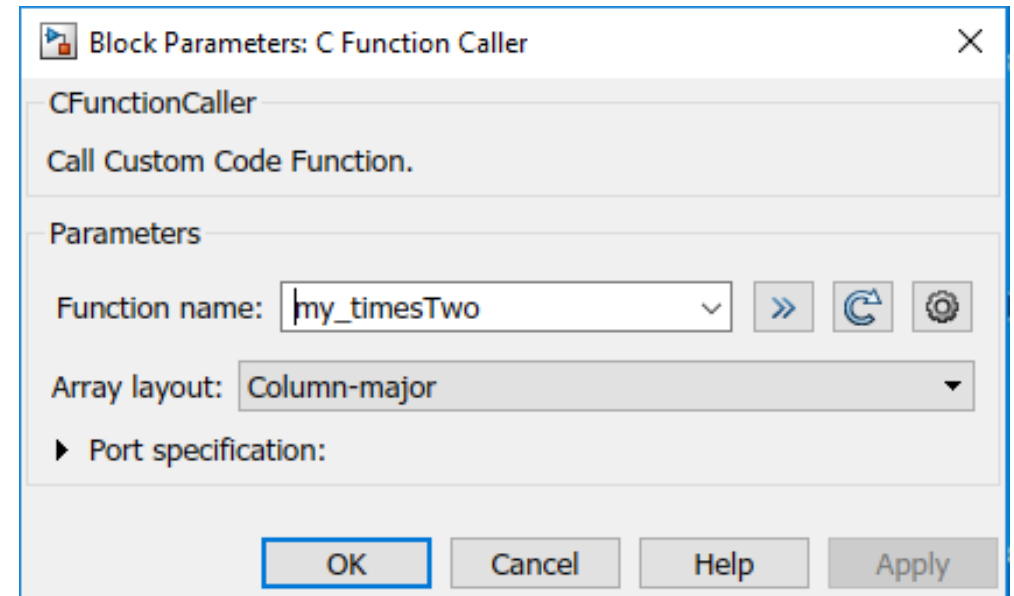
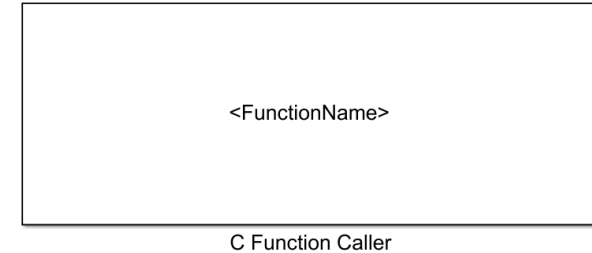
Add Rule Remove Rule

Select	Parameter Name	Parameter Value
Block	BlockType	Constant
	+ -	
Parameter	type	CONNECTIVE_JUNCTION

C Caller Block

Call external C functions directly from the model

- Simpler alternative to S-function Builder and Legacy Code Tool for calling a function written into C
- Changes in C function source file automatically captured when rebuilding model
- Supports simulation and code generation



Project References

Explorer the full project reference hierarchy and associated files directly from your Simulink Project

- View all references in a deep hierarchy
- View project and modified files for a selected reference
- Better workflow with new toolstrip

Simulink Project - Airframe Example

Toolstrip: New, Signal Multiplier, Open Project, Remove, signal gain manager, Feedback Form, Signal Multiplication, Rebuild S-function, How To Build, Terms And Conditions, Checkpoint Report, Update, Clear

Views: Files, References, Dependency Analysis

References: Signal Multiplier, plotutils, Wind Gust Library

Relative reference: ../refs/Signal Multiplier

Name	Status	SVN	Revision	Classification
.SimulinkProject			2	
demos	✓			
timesThreeDemo.slx	✓	+		Design
documentation	✓			
signalGainManager.m	✓	+		Design

Details

Name: Signal Multiplier

Description: This component project provides reusable functionality for amplifying signals.

Project root: LAB/projects/slexamples/airRef41/refs/Signal Multiplier

Change Detection:

Set a checkpoint to detect future changes made in this referenced project.

Checkpoint: **March 23, 2018 10:16:02 AM**

Custom Gauge Block

Create a gauge with a fully customizable appearance

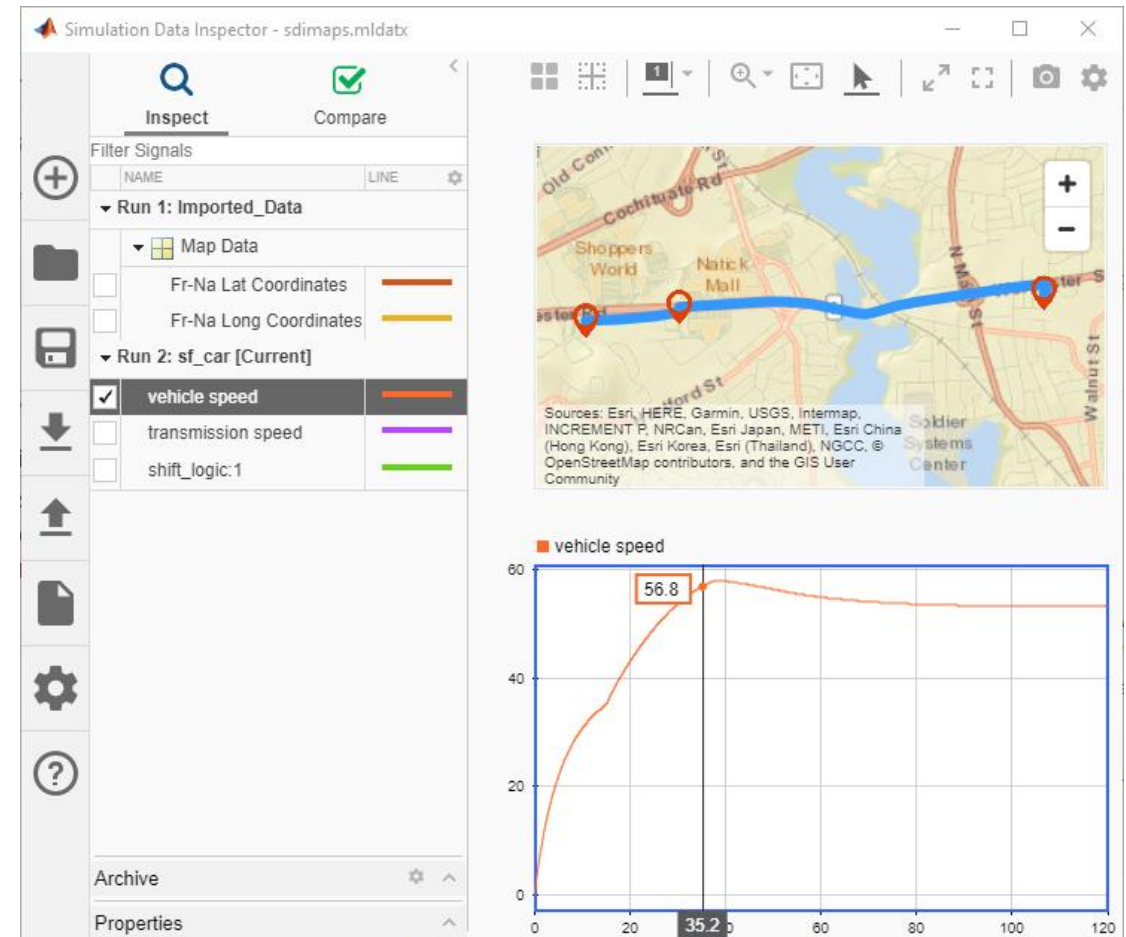
- New Dashboard block to create gauges with a customized appearance
- Choose the background image, needle image, arc, arc color, arc transparency



Maps in the Simulation Data Inspector

View synchronized map and signal data in the Simulation Data Inspector

- GPS latitude and longitude coordinates used to display location on a 2-D street map
- Position on map synchronized with simulation data
- Path history shown on map



What's New for *Simulink Requirements* in R2018b

External Rich Text Editor

Use text editing and formatting features of Microsoft Word to author and edit rich text requirements

- Spell-check requirements content
- Resize images
- Insert and edit equations
- Insert and edit tables
- Available for Description and Rationale

See: [Author and Edit Requirements Content by Using Microsoft Word](#)

The screenshot shows the MathWorks R2018b Properties window for a requirement. The 'Description' tab is active, and the 'Invoke Microsoft Word' button is highlighted with a red box. The text in the description reads: "The safety distance shall be given by the following equation:" followed by the equation $d_{safe} = S_{min} + T_{lead}V_{vehicle}$. Below the equation, it says "where MinSpace and LeadTime are defined as c".

The Microsoft Word window shows the same text and equation, with a table below it. The table has a header row and two empty rows below it.

Header		

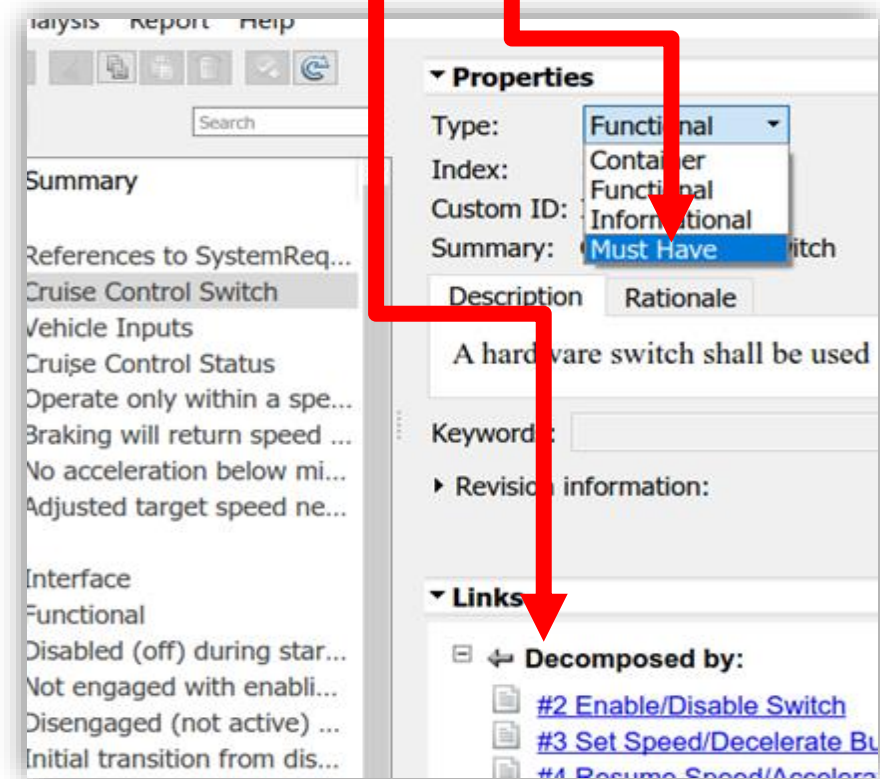
Customizable Types for Requirements and Links

R2018b

Define custom requirement and link types for improved classification and analysis

- Requirement type has been introduced for classifying requirements
 - Predefined types include Functional, Container, Informational
- Requirement and Link types are customizable through `sl_customization`.
- Roll-up status for implementation and verification statuses are computed based on specified requirements and links

```
function sl_customization(cm)
    cObj = cm.SimulinkRequirementsCustomizer;
    cObj.addCustomRequirementType('Must Have',slreq.custom.RequirementType.
        , 'Must have requirements');
    cObj.addCustomLinkType('Decompose',slreq.custom.LinkType.Implement,...
        'Decomposes','Decomposed by','Use this link for decomposing require
end
```

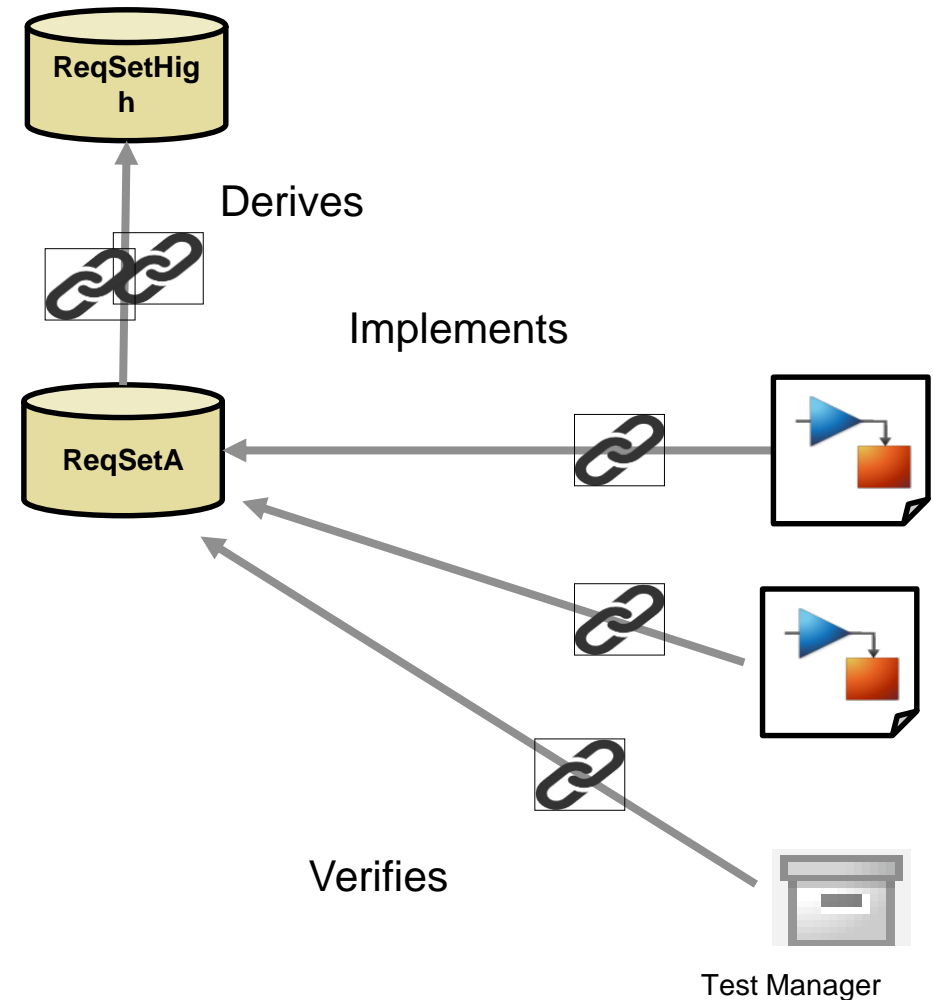


See [Define Custom Requirement and Link Types](#)

Incoming Link Resolution

Automatically load and display incoming links when opening artifacts stored in Simulink Projects

- All the link sets related to your requirements sets are loaded automatically when you load the source and destination artifacts
- Link information for any artifact is kept in memory as long as there is an artifact referring to this link information.
- Currently works for Simulink Projects only



DOORS Next Generation (DNG) Traceability with Configuration Management Enabled

R2018b

Create traceability between design and configuration-enabled DOORS Next Generation project

- Link with versions of artifacts in "streams" and "change sets".
- View incoming links from Simulink using dashboard in DNG.
- Batch redirect existing links to another stream or changeset in the project

The image displays a composite of three screenshots from the DOORS Next Generation (DNG) software interface. The top screenshot shows a 'DNG Project' configuration dialog box with dropdown menus for 'Project Area name' (set to 'Buick LaCrosse') and 'configuration stream or changeset' (set to 'changeset 3'). A red arrow points from this dialog to the 'changeset 3' dropdown in the main application window above it. The middle screenshot shows the 'Mini Dashboard' for a 'Buick LaCrosse' project, with a red circle highlighting a link labeled 'Query links from SL' under the '364 Available engines' artifact. The bottom screenshot shows an 'Overview' panel for artifact '364: <Available engines>', listing project details like 'Project: Buick LaCrosse' and 'Created On: Oct 17, 2016, 1:54:04 PM'. Two blue callout boxes are overlaid: 'Configuration selector in MATLAB' points to the configuration dialog, and 'Simulink dashboard in DNG' points to the dashboard screenshot.

See [Requirements Traceability with IBM Rational DOORS Next Generation](#)

Saving view settings of Requirements Editor

R2018b

Saving view settings of Requirements Editor and Requirements perspective

- Save Requirements Editor or Requirements Perspective setting across sessions
- Settings saved:
 - Displayed columns for requirements and links view, respectively.
 - Item Sort Order
 - Current view (Requirements View or Links View)
- Export and import settings from a MAT-File

The screenshot shows the Requirements Editor window with a menu bar (File, Edit, Display, Analysis, Report, Help) and a toolbar. The 'View: Requirements' dropdown is active. A search box is present in the top right. The main area displays a table of requirements with the following data:

Index	ID	Summary	Implemented	Verified	Revision
crs_req_func_spec					
1	#1	Driver Switch Request Handling	[Progress Bar]	[Progress Bar]	46
1.1	#2	Switch precedence	[Progress Bar]	[Progress Bar]	46
1.2	#3	Avoid repeating commands	[Progress Bar]	[Progress Bar]	46
1.3	#4	Long Switch recognition	[Progress Bar]	[Progress Bar]	55
1.4	#7	Cancel Switch Detection	[Progress Bar]	[Progress Bar]	55
1.5	#8	Set Switch Detection	[Progress Bar]	[Progress Bar]	64
1.6	#9	Enable Switch Detection	[Progress Bar]	[Progress Bar]	55
1.7	#10	Resume Switch Detection	[Progress Bar]	[Progress Bar]	55
1.8	#11	Increment Switch Detection	[Progress Bar]	[Progress Bar]	55
1.9	#15	Decrement Switch Detection	[Progress Bar]	[Progress Bar]	55
2	#19	Cruise Control Mode	[Progress Bar]	[Progress Bar]	52
3	#37	Calculate Target Speed and Thr...	[Progress Bar]	[Progress Bar]	41
4	#44	System Interface	[Progress Bar]	[Progress Bar]	60
5	#71	Justifications	[Progress Bar]	[Progress Bar]	1

See [View or Hide Columns in the Requirements Editor](#)

Other new features

R2018b

- Simulink Requirements API support for Justification
 - Use API to add justifications to complete Implementation or Verification rollup
- Export Requirement and Link Sets to Previous Version
 - Exchange Requirement and Link Sets with other users running other versions
- Synchronization for non-unique Custom ID's
 - More flexibility to work with external requirements
- USDM Excel Import to Simulink Requirements
 - Support importing from Excel following USDM format
 - Allows for multiple rows per item, and unique IDs are not always in the same column.

What's New for *Simulink Check* in R2018b

Metrics Dashboard Customization: Add Metric Thresholds

Define and visualize thresholds to increase actionability of metric data

- Categorize metric data into 3 categories - Compliant, Warning, and Non-Compliant
- Aggregated Project status on the Metrics Dashboard
- Find issues in the model through threshold indications with metric details

The screenshot displays the MathWorks Metrics Dashboard interface. The top window shows a summary for 'HevIpsReferenc...' with metrics: 1.5K Blocks, 10 Models, 10 Files, 0 MATLAB LOC, and 62 Stateflow LOC. A gauge indicates 64.4% High Integrity. Below, a bar chart shows 894 High Integrity Model Advisor issues, and a counter shows 0 Code Analyzer Warnings.

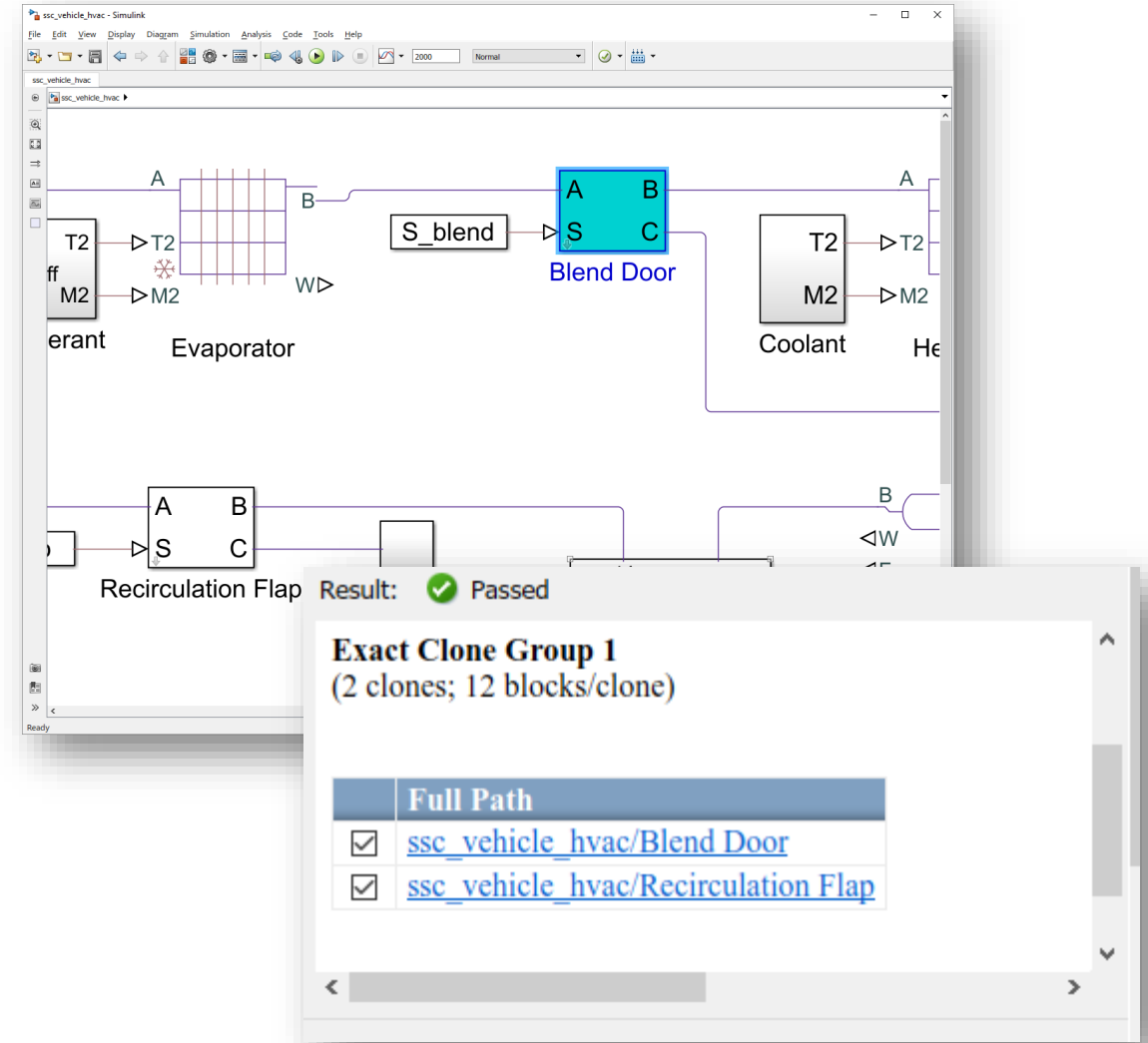
The bottom window provides a detailed view of 'Model Advisor standards issues for High Integrity'. The table below lists the issues:

Type	Component	Path	Qty	Issues	Issues (incl. Descendants)
Model	MotDynamic		1	3	3
Subsystem	Drivetrain Output	HevIpsReferenceApplication/...er Car/Drivetrain Output	1	3	3
Subsystem	Torque Limit	HevIpsPowertrainController/... Management/Torque Limit	1	3	3
Subsystem	Powertrain Control Module (PCM)	HevIpsReferenceApplication/...ain Control Module (PCM)	1	3	420
Subsystem	OutLim	HevIpsPowertrainController/...ontrol.GenControl/OutLim	1	3	3
Subsystem	Mech To Elec Power Estimate	HevIpsPowertrainController/...h To Elec Power Estimate	1	2	2
Subsystem	Mech to Elec Power Estimate	HevIpsPowertrainController/...h to Elec Power Estimate	1	2	2
Subsystem	Differential and Compliance	DrivetrainHevIps/Differential and Compliance	1	2	7
Subsystem	Torque Blending	HevIpsPowertrainController/...OpMode3/Torque Blending	1	1	1
Subsystem	Motor_Control.Mot_HEV.MotTrqHEV	HevIpsPowertrainController/...ontrol.Mot_HEV.MotTrqHEV	1	1	1
Subsystem	Operating_Modes.Stationary.Charging.ChargePowerCalc	HevIpsPowertrainController/...Charging.ChargePowerCalc	1	1	1
Subsystem	Electric Plant Output	HevIpsReferenceApplication/...ar/Electric Plant Output	1	1	1
Subsystem	Torque Limit	HevIpsPowertrainController/... Management/Torque Limit	1	1	1
Subsystem	Engine Control Module (ECM)	HevIpsReferenceApplication/...ine Control Module (ECM)	1	0	56
Subsystem	Motor	HevIpsReferenceApplication/...Car/Electric Plant/Motor	1	0	10
Subsystem	Generator	HevIpsReferenceApplication/...Electric Plant/Generator	1	0	10

Simscape Support with Clone Detection

Improve maintainability and reuse of Simscape Models

- Detect duplicate modeling patterns or clones in Simscape Models
- Refactor to replace these clones with links to reusable library blocks

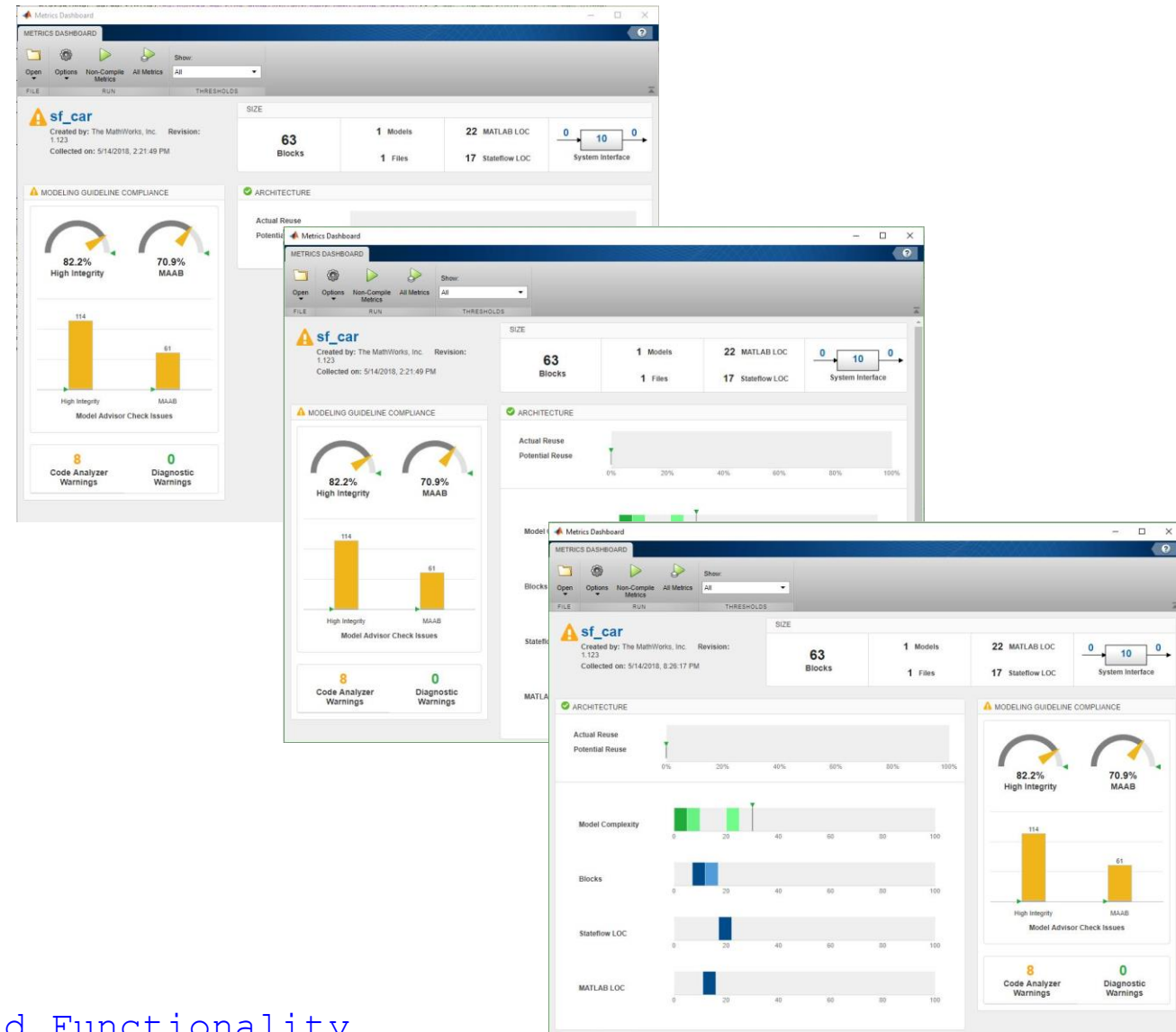


Simulink Metrics Dashboard - Layout Customization

R2018b

Update layout of dashboard by adding, removing or customizing metrics

- Reuse shipping graphical widgets & visualizations
- Distribute custom layout across organization
- Create multiple views depending on project phase



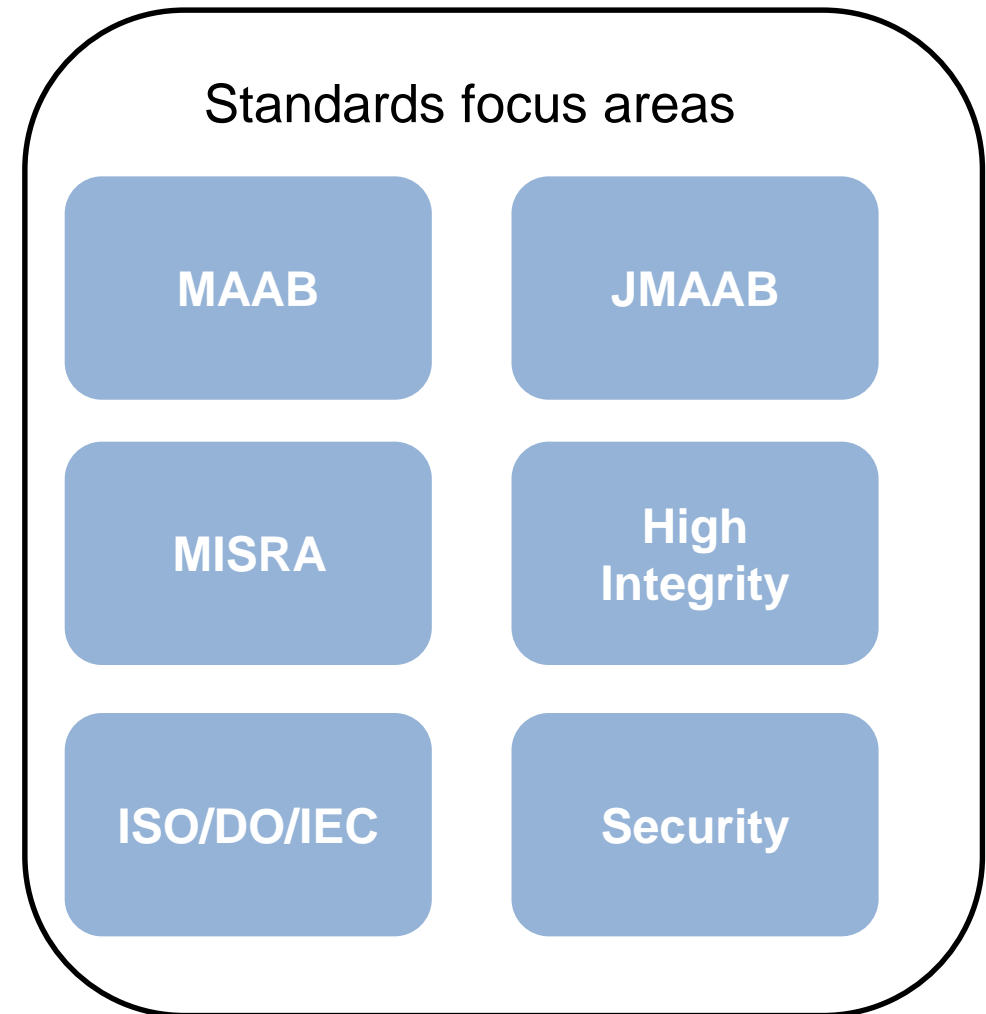
See [Customize Metrics Dashboard Layout and Functionality](#)

Additional Checks for High-Integrity Systems Modeling and MAAB 3.0

Automate checking for MAAB 3.0 and High-Integrity Systems Modeling guidelines

- 18 new High Integrity Systems Modeling checks!
 - 3 Stateflow Guidelines
 - 9 Simulink Guidelines
 - 6 MATLAB Function Block related guidelines
- New MAAB 3.0 check for number of function calls in MATLAB Function blocks
- Existing DO-178C/DO-331, and EN 50128, IEC 61508, IEC 62304, and ISO 26262 check IDs are renamed for better usability and consistency

» Documentation: [Model Advisor Checks for MAAB Guidelines](#)



JMAAB 4.01 Support

Automate checking of models to JMAAB (Japan MATLAB Automotive Advisory Board) 4.01 modeling style guidelines

- 47 New checks are added for JMAAB 4.01!
 - 23 Simulink Guidelines
 - 24 Stateflow Guidelines

Modeling Standards for JMAAB

- ✓ Naming Conventions
- ✓ Model Architecture
- ✓ Model Configuration Options
- ✓ Simulink
- ✓ Stateflow
- ✓ MATLAB Functions

Web Browser - Model Advisor Report for 'vdp'

Model Advisor Report for 'vdp' x +

Location: file:///C:/Users/purban/MATLAB/Apps/slpij/modeladvisor/vdp/report_1372.html

Filter checks

- ✓ Passed
- ✗ Failed
- ⚠ Warning
- ☐ Not Run

Keywords

Navigation

- Modeling Standards for JMAAB
- 1 Naming Conventions
- 2 Model Architecture
- 3 Model Configuration Options
- 4 Simulink
- 5 Stateflow
- 6 MATLAB Functions

View

- 🔍 Scroll to top
- Hide check details

Modeling Standards for JMAAB

- 1 Naming Conventions 13 Passed 0 Failed 2 Warning 0 Not Run

⚠ Check file names

Identify file names with incorrect characters or formatting.

Warning

The following files have incorrect file names:

File Name	Incorrect Character or Format
064566 Cruise Control Project Overview Template HLF Cruise.slsreqx	File name contains incorrect characters. Correct characters are a-z, A-Z, 0-9, and underscore (_).
064566 Cruise Control Project Overview Template HLF Cruise.slsreqx	File name starts with a number.
Key Word Report for - clone refactor .pdf	File name contains incorrect characters. Correct characters are a-z, A-Z, 0-9, and underscore (_).
Key Word Report for - clone refactor model transformer .pdf	File name contains incorrect characters. Correct characters are a-z, A-Z, 0-9, and underscore (_).

See: [Model Checks for Japan MATLAB Automotive Advisory Board \(JMAAB\) Guideline Compliance](#)

New JMAAB Checks (Slide 1 of 2)

R2018b

Model Advisor Check	Addresses Guideline
Check block orientation	mathworks.jmaab.jc_0110
Check usable characters for signal names and bus names	mathworks.jmaab.jc_0222
Check usable characters for parameter names	mathworks.jmaab.jc_0232
Check length of model file name	mathworks.jmaab.jc_0241
Check length of folder name at every level of model path	mathworks.jmaab.jc_0242
Check length of subsystem names	mathworks.jmaab.jc_0243
Check length of Inport and Outport names	mathworks.jmaab.jc_0244
Check length of signal and bus names	mathworks.jmaab.jc_0245
Check length of parameter names	mathworks.jmaab.jc_0246
Check length of block names	mathworks.jmaab.jc_0247
Check if blocks are shaded in the model	mathworks.jmaab.jc_0604
Check operator order of Product blocks	mathworks.jmaab.jc_0610
Check icon shape of Logical Operator blocks	mathworks.jmaab.jc_0621
Check for parentheses in Fcn block expressions	mathworks.jmaab.jc_0622
Check usage of Memory and Unit Delay blocks	mathworks.jmaab.jc_0623
Check usage of Lookup Tables	mathworks.jmaab.jc_0626
Check usage of the Saturation blocks	mathworks.jmaab.jc_0628
Check Signed Integer Division Rounding mode	mathworks.jmaab.jc_0642
Check type setting by data objects	mathworks.jmaab.jc_0644
Check if tunable block parameters are defined as named constants	mathworks.jmaab.jc_0645
Check prohibited comparison operation of logical type signals	mathworks.jmaab.jc_0655
Check default/else case in Switch Case blocks and If blocks	mathworks.jmaab.jc_0656
Check usage of Merge block	mathworks.jmaab.jc_0659

See Release Notes for additional changes in checks

New JMAAB Checks (Slide 2 of 2)

R2018b

Model Advisor Check	Addresses Guideline
Check for unused data in Stateflow Charts	mathworks.jmaab.jc_0700
Check first index of arrays in Stateflow	mathworks.jmaab.jc_0701
Check execution timing for default transition path	mathworks.jmaab.jc_0712
Check for parallel Stateflow state used for grouping	mathworks.jmaab.jc_0721
Check scope of data in parallel states	mathworks.jmaab.jc_0722
Check uniqueness of State names	mathworks.jmaab.jc_0730
Check usage of State names	mathworks.jmaab.jc_0731
Check uniqueness of Stateflow State and Data names	mathworks.jmaab.jc_0732
Check repetition of Action types	mathworks.jmaab.jc_0734
Check if each action in state label ends with a semicolon	mathworks.jmaab.jc_0735
Check indentation of Stateflow blocks	mathworks.jmaab.jc_0736
Check uniform spaces before and after operators	mathworks.jmaab.jc_0737
Check comments in state actions	mathworks.jmaab.jc_0738
Check updates to variables used in state transition conditions	mathworks.jmaab.jc_0741
Check boolean operations in condition labels	mathworks.jmaab.jc_0742
Check condition actions in Stateflow transitions	mathworks.jmaab.jc_0743
Check for unexpected backtracking in state transitions	mathworks.jmaab.jc_0751
Check usage of parentheses in Stateflow transitions	mathworks.jmaab.jc_0752
Check condition actions and transition actions in Stateflow	mathworks.jmaab.jc_0753
Check prohibited use of operation expressions in array indices	mathworks.jmaab.jc_0756
Check starting point of internal transition in Stateflow	mathworks.jmaab.jc_0760
Check prohibited combination of state action and flow chart	mathworks.jmaab.jc_0762
Check usage of internal transitions in Stateflow states	mathworks.jmaab.jc_0763
Check usage of transition conditions in Stateflow transitions	mathworks.jmaab.jc_0772

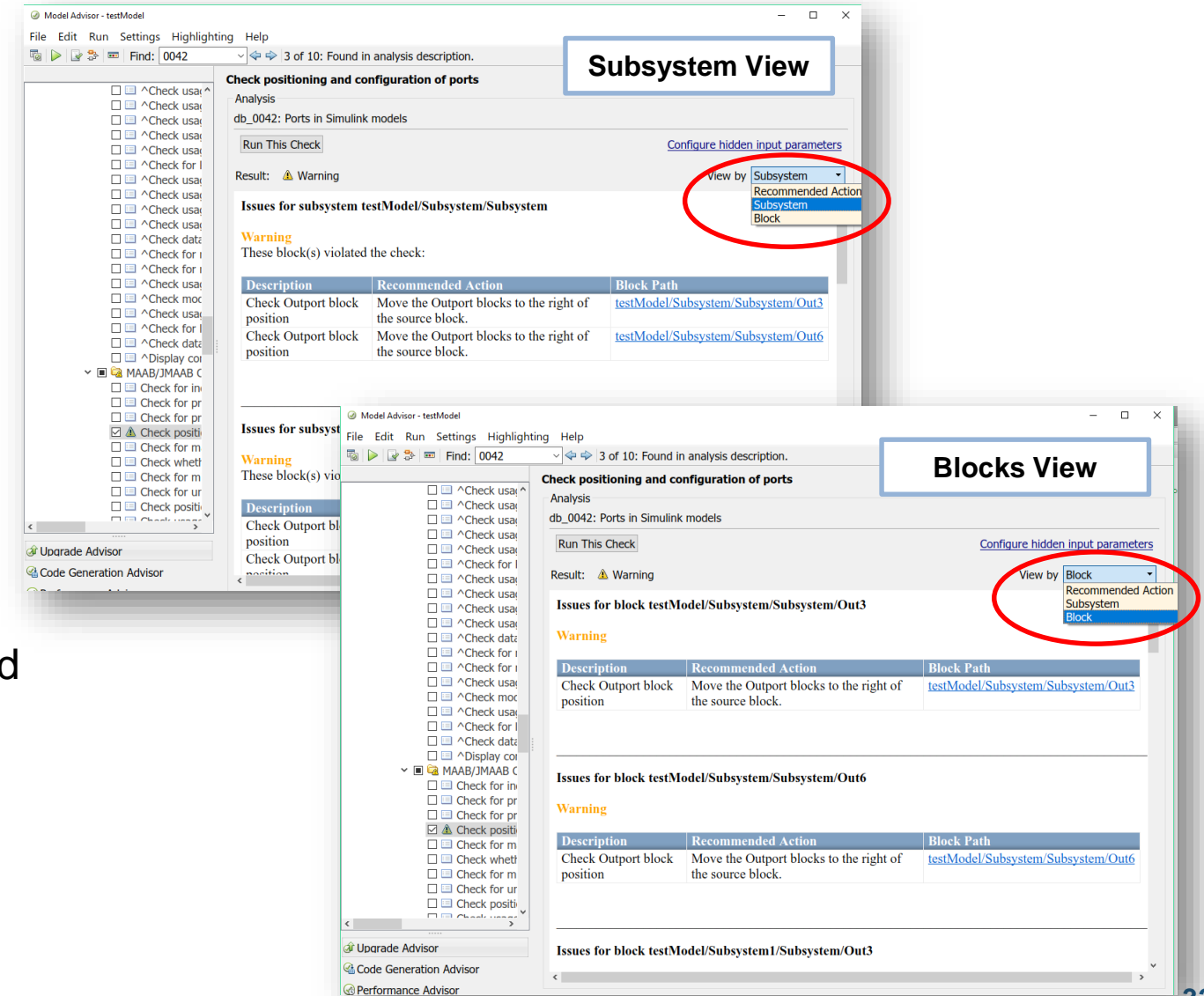
See Release Notes for additional changes in checks

New Check Authoring Style with automated reporting capabilities

R2018b

View and manage check results by subsystem or block

- Subsystem view organizes the flagged model by subsystem
- Block view organizes the flagged model components by block
- Reports provide a recommended action and hyperlinks to model component



New High-Integrity Systems Modeling Checks

R2018b

Model Advisor Check	Addresses Guideline
Check usage of standardized MATLAB function headers	mathworks.hism.himl_0001
Check if/elseif/else patterns in MATLAB Function blocks	mathworks.hism.himl_0006
Check switch statements in MATLAB Function blocks	mathworks.hism.himl_0007
Check usage of relational operators in MATLAB Function blocks	mathworks.hism.himl_0008
Check usage of equality operators in MATLAB Function blocks	mathworks.hism.himl_0009
Check usage of logical operators and functions in MATLAB Function blocks	mathworks.hism.himl_0010
Check for inappropriate use of transition paths	mathworks.hism.hisf_0014
Check naming of ports in Stateflow charts	mathworks.hism.hisf_0016
Check scoping of Stateflow data objects	mathworks.hism.hisf_0017
Check usage of conditionally executed subsystems	mathworks.hism.hisl_0012
Check usage of Merge blocks	mathworks.hism.hisl_0015
Check usage of Bitwise Operator block	mathworks.hism.hisl_0019

See Release Notes for additional changes in checks

Other New Features

R2018a

- High Integrity Systems Modeling Checks: Use the additional conditions to check the configuration parameters
- Modifications to existing Model Advisor checks that you use to verify compliance with MISRA C:2012 and Secure Coding standards
- Mnemonic Support: Use keyboard shortcuts with Metrics Dashboard

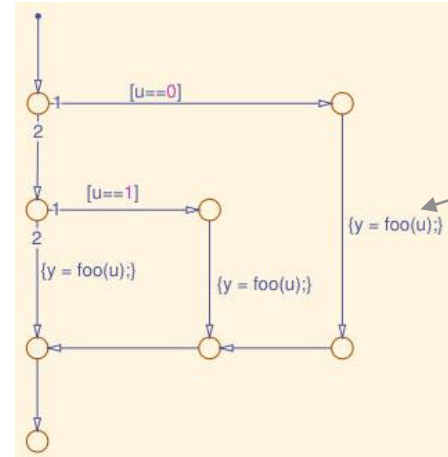
What's New for *Simulink Coverage* in R2018b

Stateflow/eML Custom Code Coverage

R2018b

Enable code coverage collection on Stateflow/eML custom code call

- Works with both mex and JIT-ed Stateflow machine
- Seamless integration with the Model Coverage API



```
#include "hfoo.h"

int foo(int u1) {
    switch(u1) {
        case 0:
            break;
        case 1:
            break;
        default:
            break;
    }
    return u1;
}
```

	Complexity	Decision
TOTAL COVERAGE		29%
1. . . . Model(s)	4	25%
1-1. mSFCov	4	25%
2. . . . Custom Code File(s)	3	33%
2-1. hfoo.c	3	33%

C Caller Block Code Coverage

R2018b

Enable code coverage collection on the new C Caller block

- Works with both mex and JIT based execution
- Seamless integration with the Model Coverage API

The diagram illustrates the integration between a Simulink block and its corresponding C code. A Simulink block labeled 'C Function Caller' has an input port 'u1' and an output port 'out'. The block is connected to a code editor showing the implementation of the 'foo' function. The code is as follows:

```
#include "hfoo.h"

int foo(int u1) {
    switch(u1) {
        case 0:
            break;
        case 1:
            break;
        default:
            break;
    }
    return u1;
}
```

The code editor also shows the 'Block Parameters: C Function Caller' dialog box, which is used to configure the block. The dialog box has the following parameters:

- Function name: foo
- Array layout: Column-major
- Port specification:

Arg name	Port name	Scope	Index	Type	Size
u1	u1	Input	1	int32	1
out	out	Output	1	int32	1

Coverage Report by Model

Top Model: mCFcn

	Complexity	Decision	Execution
TOTAL COVERAGE		33%	71%
1... Model(s)	0	--	100%
1-1... mCFcn	0	--	100%
2... Custom Code File(s)	3	33%	67%
2-1... hfoo.c	3	33%	67%

Parallel Simulation: Collect Coverage When Using pars **R2018b**

- Set SimulationInput:

```
in = Simulink.SimulationInput(gcs)
in.setModelParameter('RecordCoverage', 'on')
in.setModelParameter('CovSaveSingleToWorkspaceVar', 'on');
```

- Get coverage data from SimulationOutput:

```
>>simOut = parsim(in)
ans =
  Simulink.SimulationOutput:
    yout: [1x1 Simulink.SimulationData.Dataset]
    covdata: [1x1 cvdata]
    SimulationMetadata: [1x1 Simulink.SimulationMetadata]
    ErrorMessage: [0x0 char]
```



Coverage Visualization inside Simulink Editor

R2018b

Hover over an item to see its coverage

Click

Coverage Details

Saturate block "[Saturation](#)"

[Justify or Exclude](#)

Parent: [/slvndemo_cv_small_controller](#)

Uncovered ▶

Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	75% (3/4) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

input > lower limit	50%
false	0/6

Simulink Code Coverage Filter API

R2018b

Extend the existing Simulink Coverage Filter API for code coverage

- Filtering choices for coverage justification or exclusion for:
 - File
 - Function
 - Decision
 - Condition
- Support for the CFunctionCaller block, SF/eML custom code and SIL/PIL

```
% Create a filter for a particular function
```

```
>> s = slcoverage.CodeSelector(slcoverage.CodeSelectorType.Function, 'counterbus.c', 'counterbusFcn');
```

```
% Create a filter for a particular decision
```

```
>> s = slcoverage.CodeSelector(slcoverage.CodeSelectorType.Decision, 'counterbus.c', 'counterbusFcn', 'inputGElower', 2);
```

```
% Create all the possible code filters for Software-in-the-Loop based coverage collection
```

```
>> selectors = slcoverage.Selector.allSelectors(model, 'SimulationMode', 'sil');
```


Automatic Code Filter Rules Creation From Code Prover Results

R2018b

Automatically generate justification rules for dead-logic using Polyspace Code Prover results

- Allow to automatically justify code detected as not reachable by Code Prover
- Apply to:
 - Function that are never called
 - Expression that are not reachable or always evaluate to true or false
- Require a BF license for reading the results

1) Launch CP

2) Generate Results

4) Auto Import Filter rules

3) Select Results

ps_results.pscp

Coverage Data

Name	Type	Mode	Justification
0	by C/C++ decis...	Justified	Polyspace Cod...
0	by C/C++ decis...	Justified	Polyspace Cod...
bar	by C/C++ funct...	Justified	Polyspace Cod...

The decision expression "0" is justified
(in file "counterbus.c", function "counterbusFcn")

Filename: /mathworks/devel/sandbox/eroy/work/slcoverage/filter/foo*

Save filter
Load filter

Make justification filter rules for dead logic (using Polyspace Code Prover results)

What's New for *Simulink Coverage* in R2018b

Microsoft Excel®-Based Testing

R2018b

Define test cases in Excel spreadsheets using templates

- Generate a template test spreadsheet from any model or harness (SUT)
- Specify test case input data, output data, tolerances, and parameters using the template
- Fill data into spreadsheet and import into Simulink Test as a test case

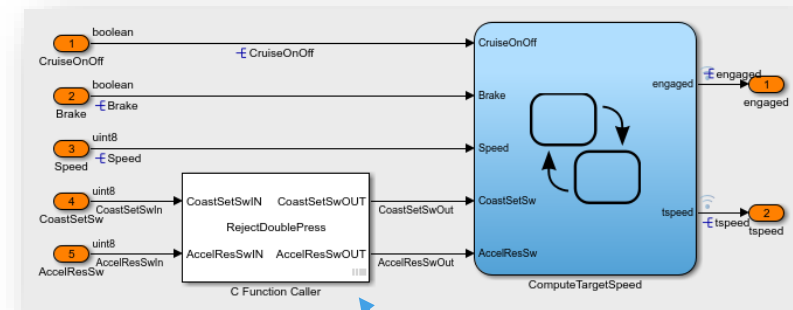
time	Magnitude	Angle	Parameter:	Value:	time	point.pixels_x	point.pixels_y
						AbsTol: 0.001	AbsTol: 0.001
						RelTol: 0.1	RelTol: 0.1
						BlockPath: coordinate_transform/Screen coordinates	BlockPath: coordin
						Source: Input	Source: Output
0	0	0	Xscale	1	0	0	0
10	0	0	Yscale	1	0.2	0	0
					0.4	0	0
					0.6	0	0
					0.8	0	0

C Caller Block Support

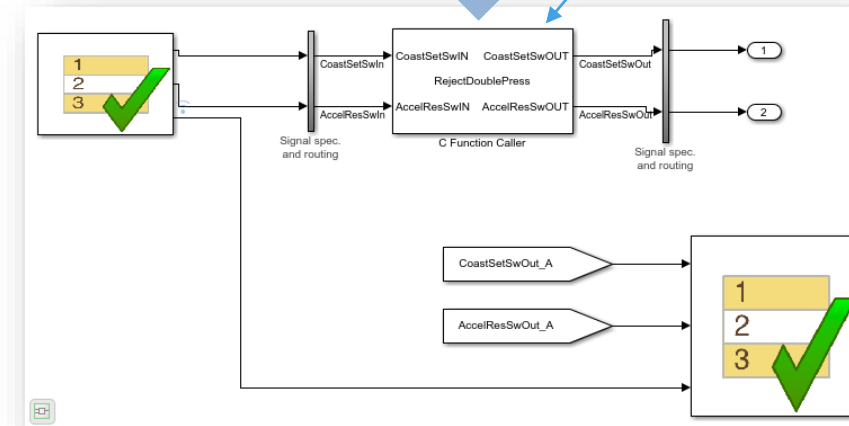
R2018b

Verify model and hand code together

- C Caller block allows you to call a C function directly from a model
- Test the C function by creating a test harness for the C Caller block
- Author, manage and execute tests of the C function with Simulink Test



Handwritten C Code



Test Harness

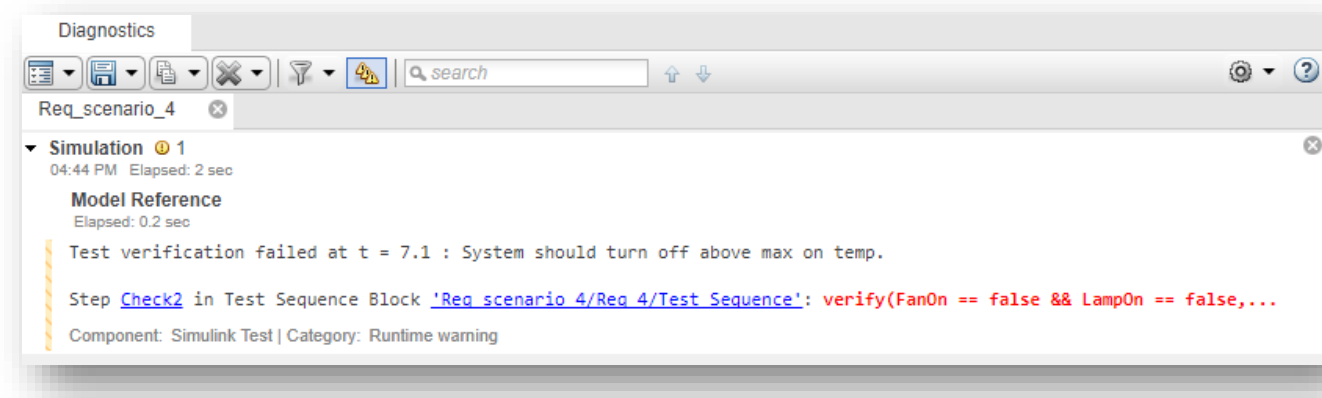
» Example: [Code Verification with Simulink Test](#)

SystemVerilog Assertions

R2018b

Assess model behavior in your HDL test environment

- Include verify statements in Test Sequence blocks and Test Assessment blocks
- Generate DPI component with HDL Verifier™ mapping to the verify statements
- Use the SystemVerilog DPI component in your HDL test environment.



Simulation emits a warning if the verify assessment fails

Lane-Following Algorithm Testing Example

Learn how to apply requirements-based testing for an automotive lane-following system

- Author tests in Simulink Test to verify safe operation for each requirement
- Use the model verification blocks to verify the algorithm
- View test results linked to high level test requirements in Simulink Requirements

Test Description	Host car	Lead car	Third car
Target Discrimination Test	initial velocity = 30m/s HWT = 2.2sec (HW = 66m) v_set = 30m/s	constant accel 24m/s ² → 27m/s @ 2m/s ² V _{end} = 27m/s (97.2kph)	24m/s

See [Testing a Lane Following Controller](#)

Additional Features

R2018b

- Test harness support for string data types
- Testing with MATLAB Unit Test
 - Specify the location to save the HTML model coverage report
 - Produce model coverage results in a format compatible with continuous integration systems such as Jenkins™. See [Tests for Continuous Integration](#)
 - Publish Test Manager results in the MATLAB® Test Report,
- Unified Scheduling Options
 - Generate scheduler for modeling styles including export function models and rate-based models
- Ability to overwrite simulation output results

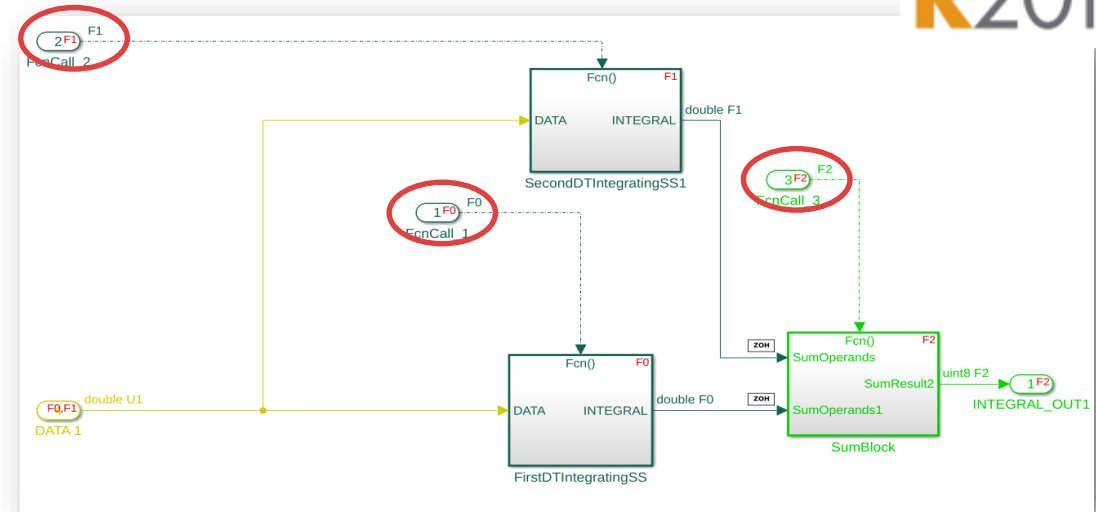
What's New for *Simulink Design Verifier* in R2018b

Export function models

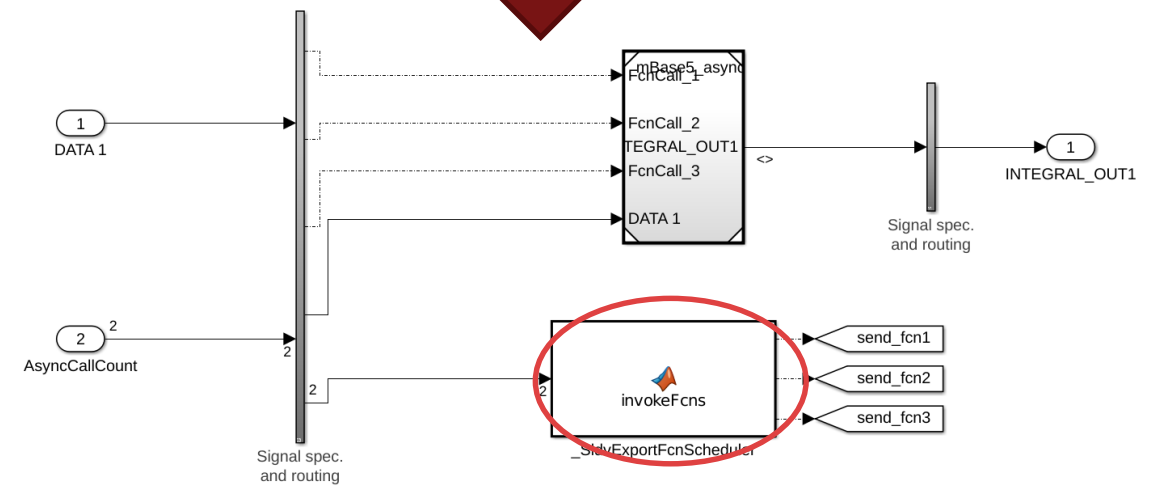
Simulink Design Verifier supports export function models

- Automatically create schedulers for analysis
 - Periodic export-function
 - Asynchronous export-functions

- Leverage harness creation from SL Test



Generate



SLDV support for Stateflow custom code

R2018b

Allow using SLDV on models containing Stateflow custom code

- Supports test generation and property proving
- Test generation includes C/C++ test coverage objectives
- To enable:


```
set_param(<model>, 'SimParseCustomCode', 'on', ...
                'SimAnalyzeCustomCode', 'on')
```

Parse custom code symbols
 Enable custom code analysis

Insert custom C code in generated:

Enable support for code coverage and Simulink Design Verifier.

Source file: _____ Source file: _____

Header file
 Initialize function
 Terminate function

```
{
  counterbus_struct = inbus;
  counterbus_struct.inputsignal = inbus.inputsignal;

  counterbusFcn(&counterbus_struct, u2, &outbus, &y2);
}
```

```
#include "hCounterBus.h"

void counterbusFcn(COUNTERBUS *u1, int u2, COUNTERBUS *y1, int *y2)
{
  int limit;
  int inputGELower;

  limit = u1->inputsignal.input + u2;
  inputGELower = (limit >= u1->limits.lower_saturation_limit);

  if ((u1->limits.upper_saturation_limit >= limit) && inputGELower) {
    *y2 = limit;
  } else {
    if (inputGELower) {
      limit = u1->limits.upper_saturation_limit;
    } else {
      limit = u1->limits.lower_saturation_limit;
    }
    *y2 = limit;
  }

  y1->inputsignal.input = *y2;
  y1->limits = u1->limits;
}
```

[Back to summary](#)
mSFBusDemo
[All Objectives](#) [Observability](#)

Condition:

- condition limit >= u1->limits.lower_saturation_limit T (file **SATISFIED** - [View test case](#) hCounterBus.c, function counterbusFcn, line 10)
- condition limit >= u1->limits.lower_saturation_limit F (file **SATISFIED** - [View test case](#) hCounterBus.c, function counterbusFcn, line 10)
- condition u1->limits.upper_saturation_limit >= limit T (file **SATISFIED** - [View test case](#) hCounterBus.c, function counterbusFcn, line 12)
- condition u1->limits.upper_saturation_limit >= limit F (file **SATISFIED** - [View test case](#) hCounterBus.c, function counterbusFcn, line 12)
- condition inputGELower T (file hCounterBus.c, function counterbusFcn, line 12) **SATISFIED** - [View test case](#)
- condition inputGELower F (file hCounterBus.c, function counterbusFcn, line 12) **SATISFIED** - [View test case](#)

Decision:

SLDV support for the C Caller Block

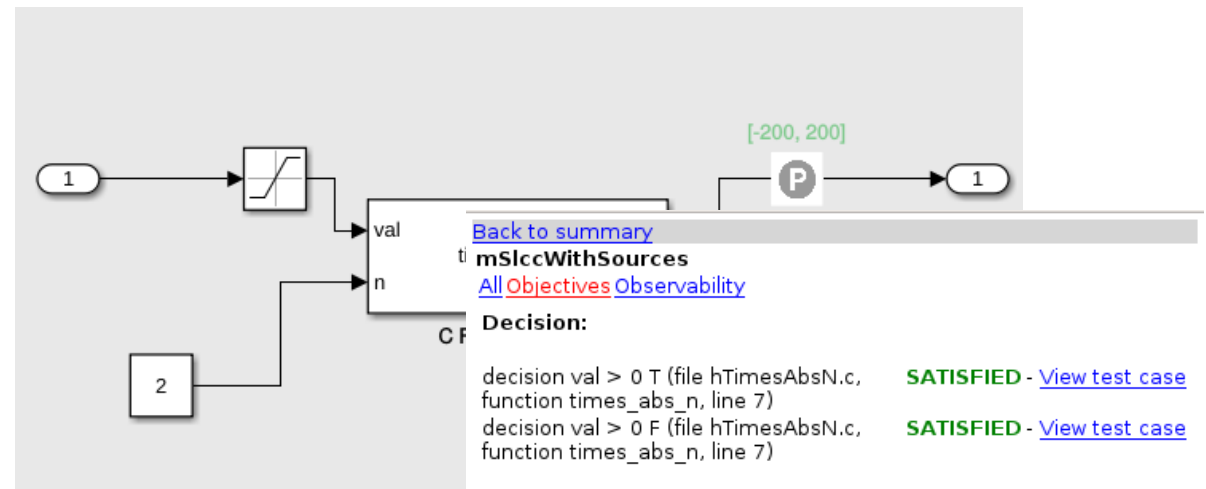
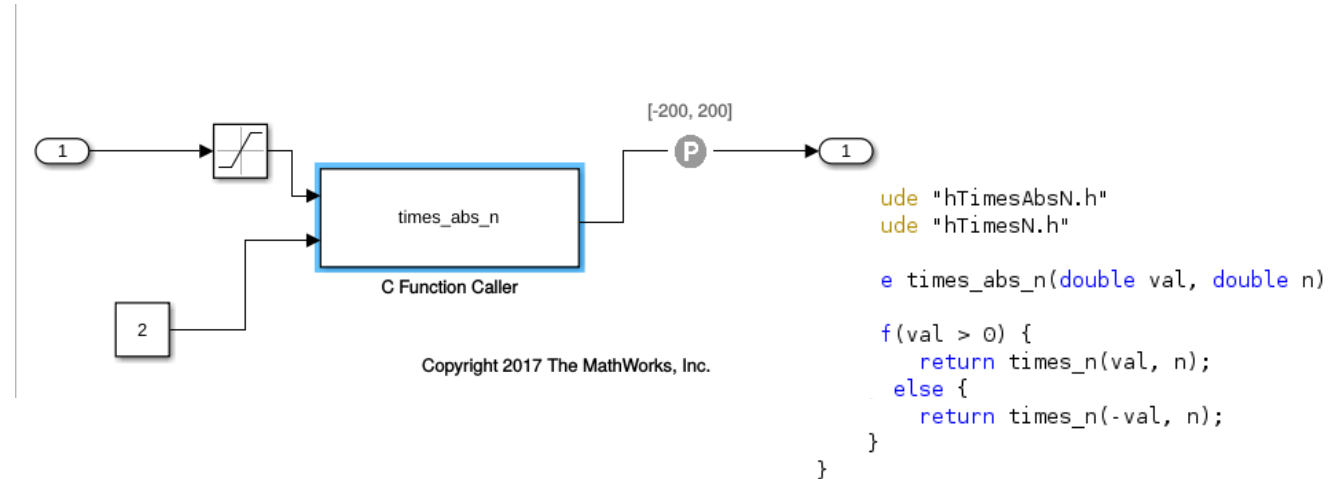
R2018b

Allow using SLDV on models containing C Caller Blocks

- Supports test generation and property proving
- Test generation includes C/C++ test coverage objectives

To enable:

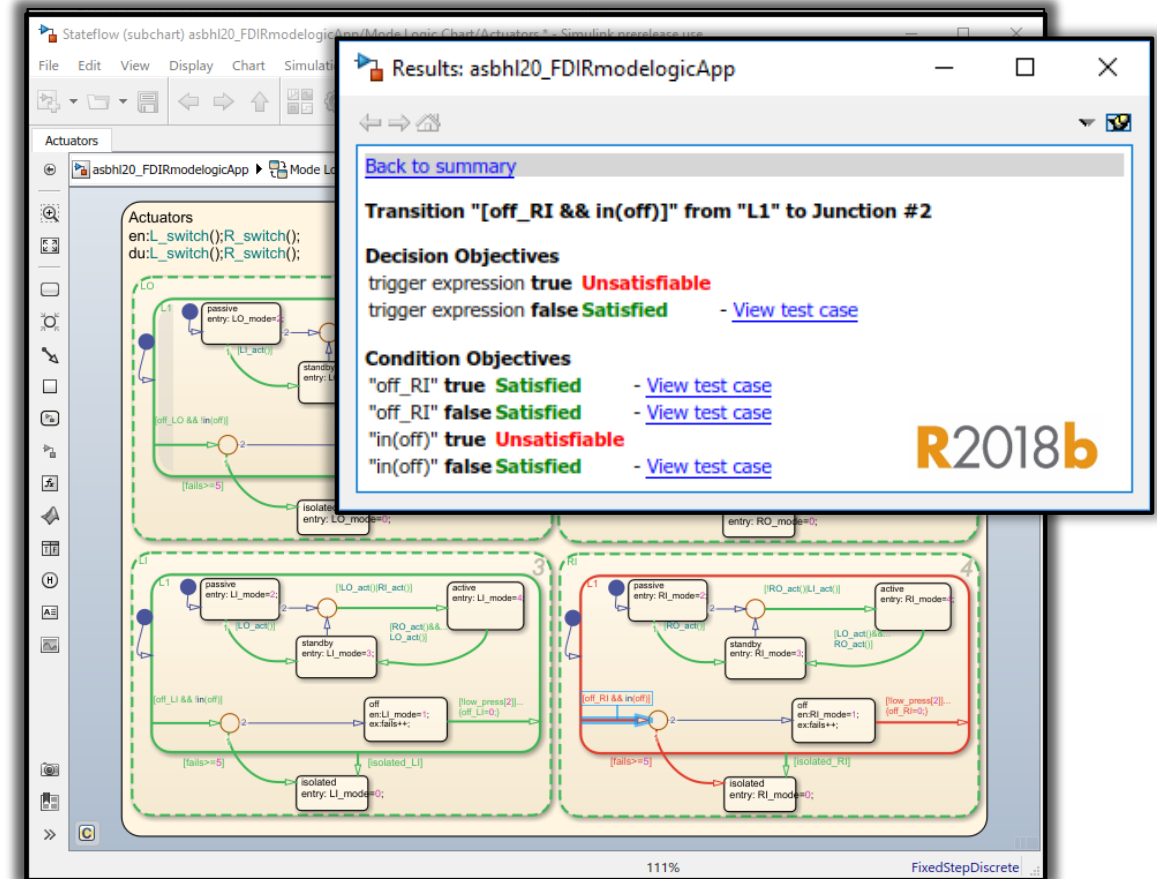
```
set_param(<model>, 'SimParseCustomCode', 'on', ...
           'SimAnalyzeCustomCode', 'on')
```



Improved Precision for Floating-point Analysis

Reduce impact of rational approximation on analysis results

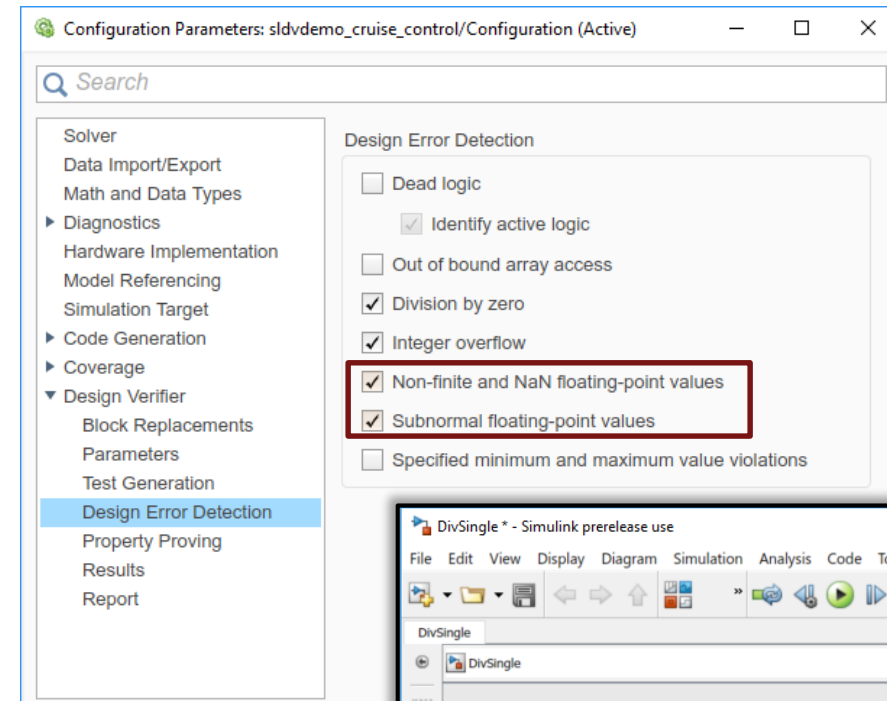
- Reduces occurrences of analysis results impacted by rational approximation, such as
 - unsatisfiable under approximation \rightarrow unsatisfiable
 - undecided with testcase \rightarrow satisfied
 - undecided due to nonlinearity \rightarrow satisfied
- Analysis may take more time and memory due to the bit-precise refinement phase
- R2018a** precision floating point only support in



Floating point Design Errors

Detect computation of +/-Infinity, NaN and subnormal values

- New design error checks for floating point computations
- Analysis may take more time and memory due to the refinement phase



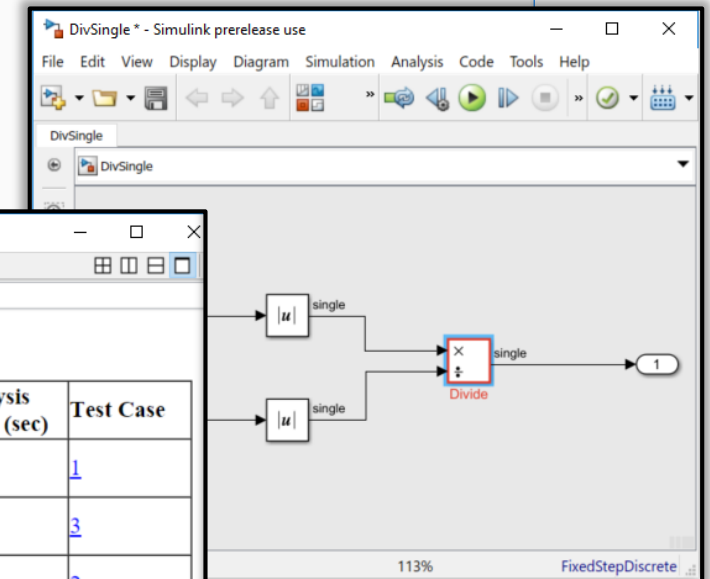
Web Browser - Simulink Design Verifier Report

Simulink Design Verifier Report

Location: file:///C:/Work/sldv_output/DivSingle/DivSingle_report.html

Objectives Falsified - Needs Simulation

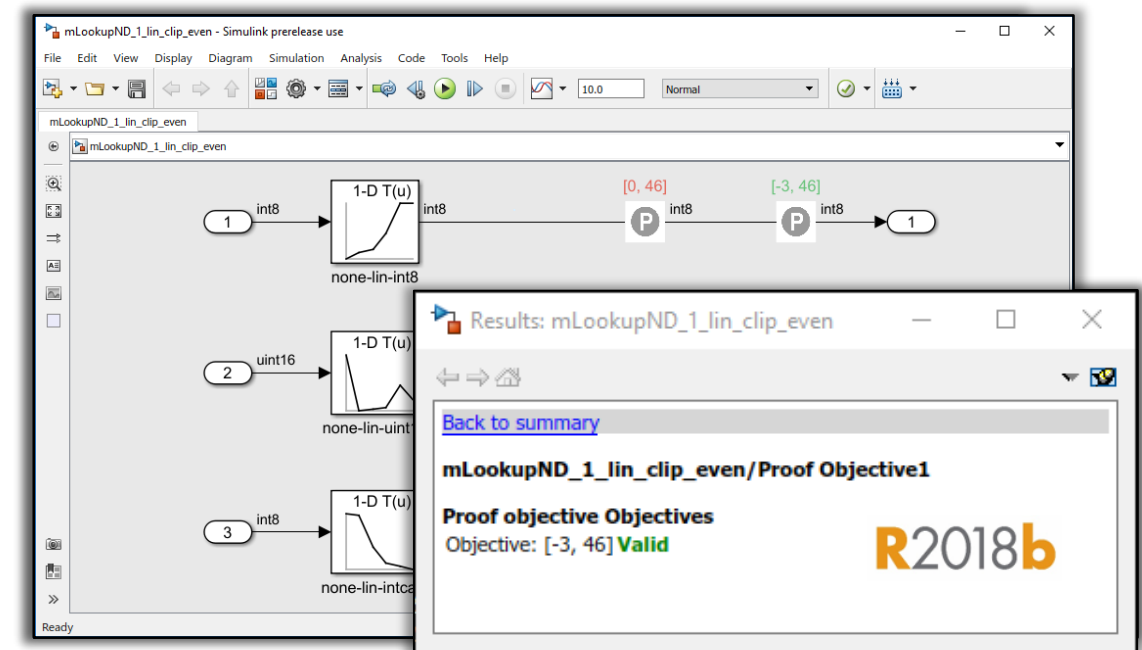
#	Type	Model Item	Description	Analysis Time (sec)	Test Case
3	Floating-point error	Divide	+/-Infinity	65	1
4	Floating-point error	Divide	NaN	219	3
5	Floating-point error	Divide	Subnormal value	142	2



Better Precision for Lookup Table analysis

Improve precision for lookup tables in all analysis modes

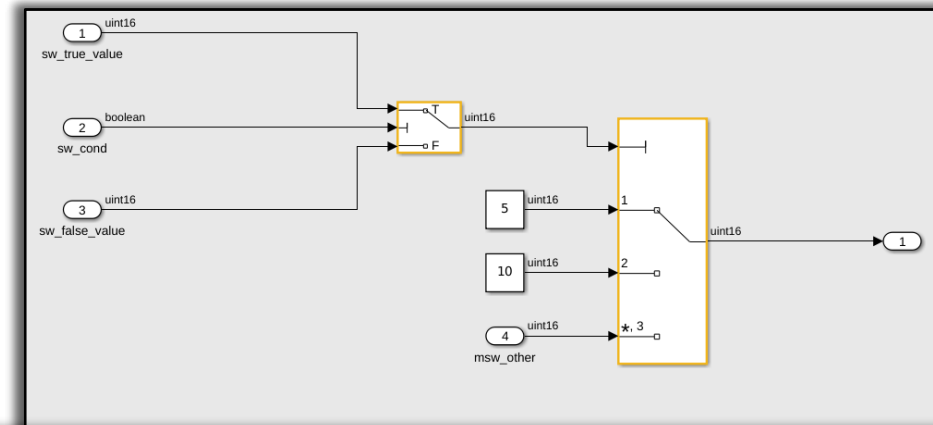
- Better precision for derived ranges
- Better precision for proven results by reducing impact of approximations
 - unsatisfiable under approximation → unsatisfiable
 - valid under approximation → valid



Avoid Run-Time Diagnostic Errors from Multiport Switch

Generate test cases that do not trigger multiport switch error diagnostic

- Generated test cases can be simulated without runtime errors
- Help achieve better coverage from the generated test suite



Back to summary

multiport_test_input_error/Multiport Switch R2018a

Decision Objectives

integer input value = 1 (output is from input port 1)	Satisfied	- View test case
integer input value = 2 (output is from input port 2)	Satisfied	- View test case
integer input value = *,3 (output is from input port 3)	Undecided due to runtime error	- View test case

Diagnostics

multiport_test_input_error_harness

Simulation 1
03:05 PM Elapsed: 4 sec

An error occurred while running the simulation and the simulation was terminated

Caused by:

- The control value '0' for 'multiport_test_input_error_harness/Test Unit (copied from multiport_test_input_error)/Multiport Switch' does not correspond to any data port indices. To suppress this message and use the default port, you can change the settings of 'Diagnostic for default case' to 'None'

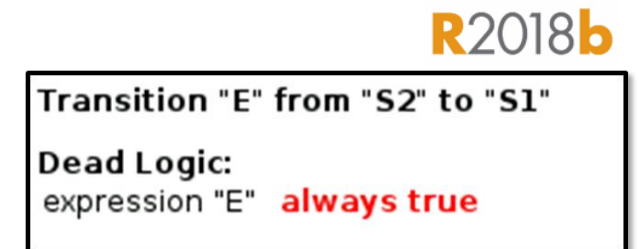
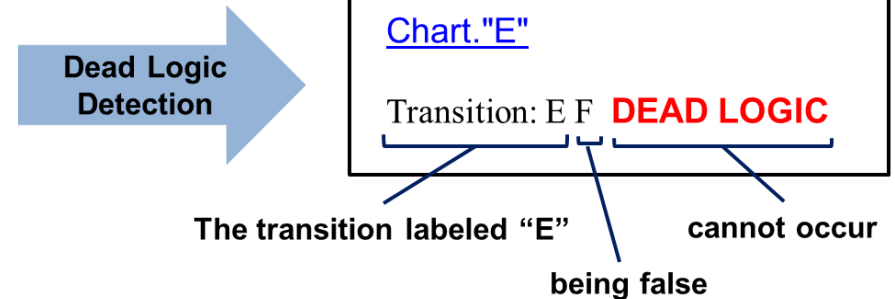
Component: Simulink | Category: Block error

Redesigned Status Reporting

R2018b

Dead logic report simplification

- Simplify dead logic reporting making it intuitive for user to understand its effect on the design
- Indicate types like “State” and “Transition” as part of item label and omit from descriptions of coverage outcomes

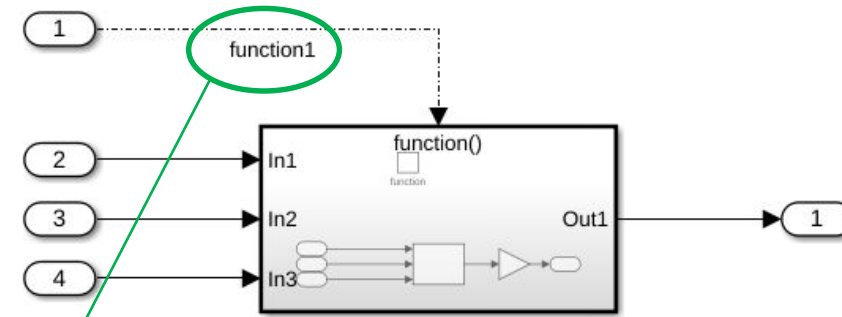


Enhanced Automatic Test Case Generation for Embedded Coder Generated Code

R2018b

Continue enhancing automatic test case generation for SIL code with Simulink Design Verifier

- Leverage SLDV support for export function model with root function-call inputs for SIL-based test generation
- Support for Row-Major array layout



```
13-May-2018 07:46:52
Preprocessing model...done
Checking compatibility for test generation: model 'mExportFunction'
Checking compatibility for test generation target: Code Generated as Top Model!
Compiling model...done
Checking compatibility...done
```

```
13-May-2018 07:47:35
'mExportFunction_SldvScheduler_replacement1' is compatible for test generation with
Simulink Design Verifier.
```

File Contents/Complexity	Test 1							
	Decision	Statement	Function	Function call	Function call	Function call	Function call	Function call
1. mExportFunction.c	4	100%	100%	100%	100%	100%	100%	100%
2. function1	2	100%	100%	100%	100%	100%	--	--
3. mExportFunction_initialize	1	--	100%	100%	100%	100%	100%	100%
4. mExportFunction_terminate	1	--	100%	100%	100%	100%	--	--

What's New for *Supporting AUTOSAR* in R2018b

AUTOSAR Run-Time Calibration

Configure Model Signals and States with AUTOSAR “perspective”

- Map internal signals and states to `ArTypedPerInstanceMemory` and `StaticMemory` for run-time calibration
- You can also configure its name, mapping, and other AUTOSAR properties, in the Property Inspector

The screenshot shows a Simulink model of a counter system and its configuration in the Property Inspector. The model includes blocks for incrementing, summing, comparing against a limit, switching, and amplifying signals. The Property Inspector shows the configuration for the `RelOpt` signal.

Named or Test Pointed Signals

Blocks with configurable state

Autosar properties for ArTypedPIM or StaticMemory

NAME	VALUE
Source	RelOpt
Name	equal_to_count
Code	
Mapped To	StaticMemory
Path	rtwdemo_autosar_counter
ShortName	equalCnt
Volatile	true
AdditionalNativeTypeQua...	
SwAddrMethod	
Calibration attributes	
SwCalibrationAccess	ReadOnly
DisplayFormat	

Source	Name
RelOpt	equal_to_count
Sum	sum_out
Switch	switch_out

Source	Name
X	X

AUTOSAR Run-Time Calibration

Configure Model Parameters with AUTOSAR “perspective”

- Map model workspace parameters to AUTOSAR component internal SharedParameters and ConstantMemory for run-time calibration
- Additional AUTOSAR properties can be set using the property inspector

The screenshot displays the MATLAB/Simulink environment for a model named 'rtwdemo_autosar_counter'. The main workspace shows a block diagram with components like 'INC', 'sum_out', 'LIMIT', 'RESET', and 'Amplifier'. The 'Code Mappings - AUTOSAR' window is open, showing a table of parameter mappings:

Source	Mapped To
INC	SharedParameter
K	ConstantMemory
LIMIT	Auto
RESET	Auto

The 'Property Inspector' on the right shows the configuration for parameter 'K':

NAME	VALUE
Source	K
Code	
Mapped To	ConstantMemory
Const	true
Volatile	false
Calibration attributes	
SwCalibrationAccess	ReadWrite
DisplayFormat	

Specify C type qualifiers for AUTOSAR static and constant memory

Customize generated code with C type qualifiers to control compiler optimizations

- Use AUTOSAR AdditionalNativeTypeQualifier to specify C-type qualifiers for AUTOSAR constant and static memory
- Import and export type qualifiers from and to ARXML
- Generate code containing type qualifiers

The screenshot displays the MATLAB/Simulink environment. The main window shows a Simulink model titled "Intake Airflow Estimation and Closed-Loop Correction". The model includes blocks for "Throttle Transient", "Pumping Constant", "Feedforward Control", and "Feedback Control".

The "Code Mappings - AUTOSAR" window is open, showing a table of mappings:

Source	Name	Mapped To	Path
Logic1	enable_integration	ArTypedPerInstanceMemory	...Closed-Loop Correction
Product1	e1	StaticMemory	...Closed-Loop Correction
Relational Operator1	normal_operation	Auto	StaticMemory ...Closed-Loop Correction
Sum1	e0	Auto	...Closed-Loop Correction
Throttle Transient	myTestpoint	ArTypedPerInstanceMemory	...Closed-Loop Correction

The "Property Inspector" window on the right shows the properties for the signal "Product1":

NAME	VALUE
Source	Product1
Name	e1
Code	
Mapped To	StaticMemory
Path	mFuelRateControllerLibVer
ShortName	signalSM
Volatile	false
AdditionalNativeTy...	test_qualifier
SwAddrMethod	
Calibration attributes	

```
/* Static Memory for Internal Data */
test_qualifier float32 signalSM;
```

```
<ADDITIONAL-NATIVE-TYPE-QUALIFIER>test_qualifier</ADDITIONAL-NATIVE-TYPE-QUALIFIER>
```

AUTOSAR Memory Sections

Author, edit and map AUTOSAR SwAddrMethods

- Use SwAddrMethods to control memory placement of AUTOSAR functions and data
- Add SwAddrMethods in AUTOSAR Dictionary
- Map SwAddrMethods to Runnables, Runnable Data, Signals, States and Parameters
- Generate C and ARXML with memory section information

The screenshot displays the AUTOSAR Dictionary tool interface. The left pane shows the 'AUTOSAR' tree structure with 'SwAddrMethods' selected. The right pane shows a table with columns 'Name' and 'SectionType', listing 'Code' and 'Var'. The main workspace shows a block diagram titled 'Modeling Multiple AUTOSAR Runnable Entities' with three 'Runnable' blocks and their associated subsystems and ports. The bottom pane shows 'Code Mappings - AUTOSAR' with tabs for 'Inports', 'Outports', 'Entry-Point Functions', and 'Data Transfers'. The 'Entry-Point Functions' tab is active, showing a table of mappings.

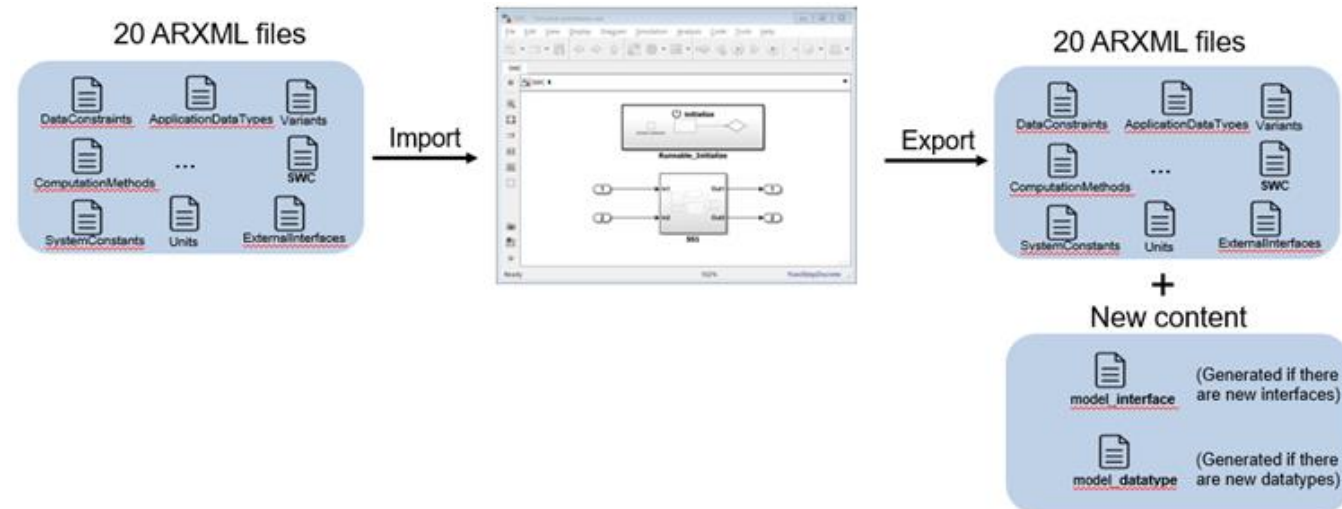
Source	Runnable
Exported Function:Runnable1	Runnable1
Exported Function:Runnable2	Runnable2
Exported Function:Runnable3	Runnable3

The Property Inspector on the right shows the configuration for the selected 'SwAddrMethod', with 'Code' selected for the 'SwAddrMethod' property.

Improved Round-trip of ARXML file content and structure

Preserve imported ARXML file structure for export

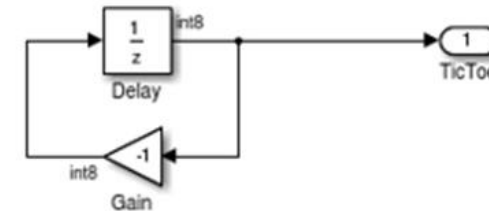
- Preserve ARXML file structure on import (physical level interoperability)
- Preserve AUTOSAR element UUIDs
 - If an imported element does not have a UUID, none is created
- New elements exported to separate file(s)
- New data types or interfaces exported to separate ARXML files



Generate code for models uses AUTOSAR platform types

Generated code uses AUTOSAR platform types

- Use AUTOSAR platform types e.g. `sint8` instead of corresponding Simulink code generation types, e.g. `int8_T`
- No need to configure Simulink replacement types in the model



```
void Runnable1(void)
{
  int8_T rtb_Gain_p;
  int8_T Delay_n;

  rtb_Gain_p = (int8_T)
    -rtDWork.Delay_DSTATE_a;
  ...
}
```

Simulink code generation types

```
void Runnable1(void)
{
  sint8 rtb_Gain_p;
  sint8 Delay_n;

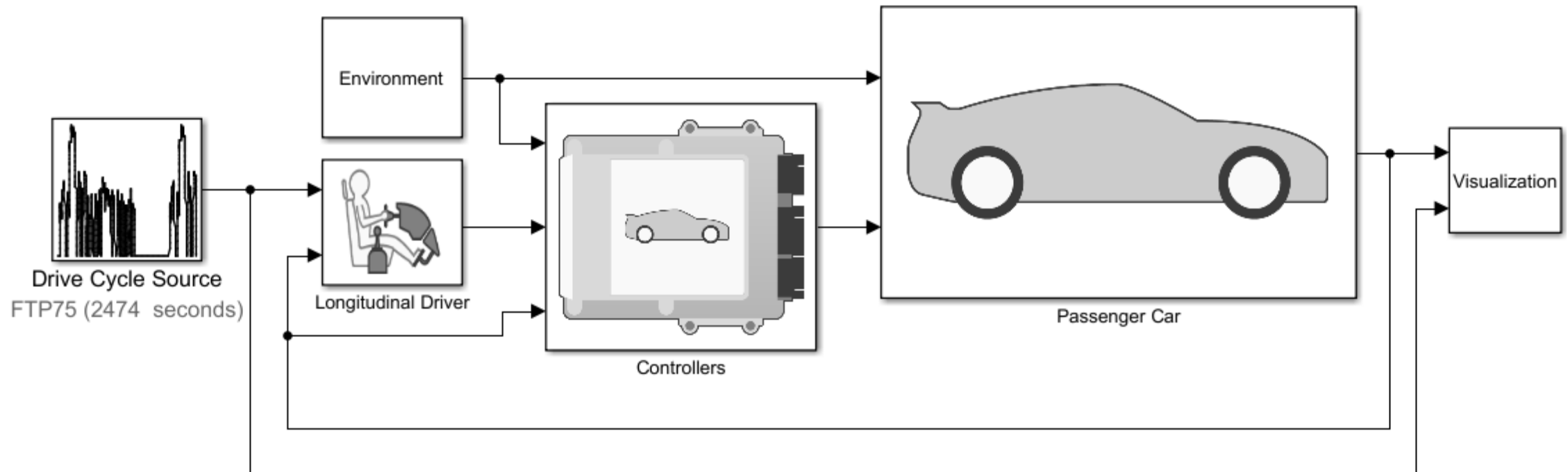
  rtb_Gain_p = (sint8)
    -rtDWork.Delay_DSTATE_a;
  ...
}
```

AUTOSAR platform types

Powertrain Blockset

Powertrain Blockset

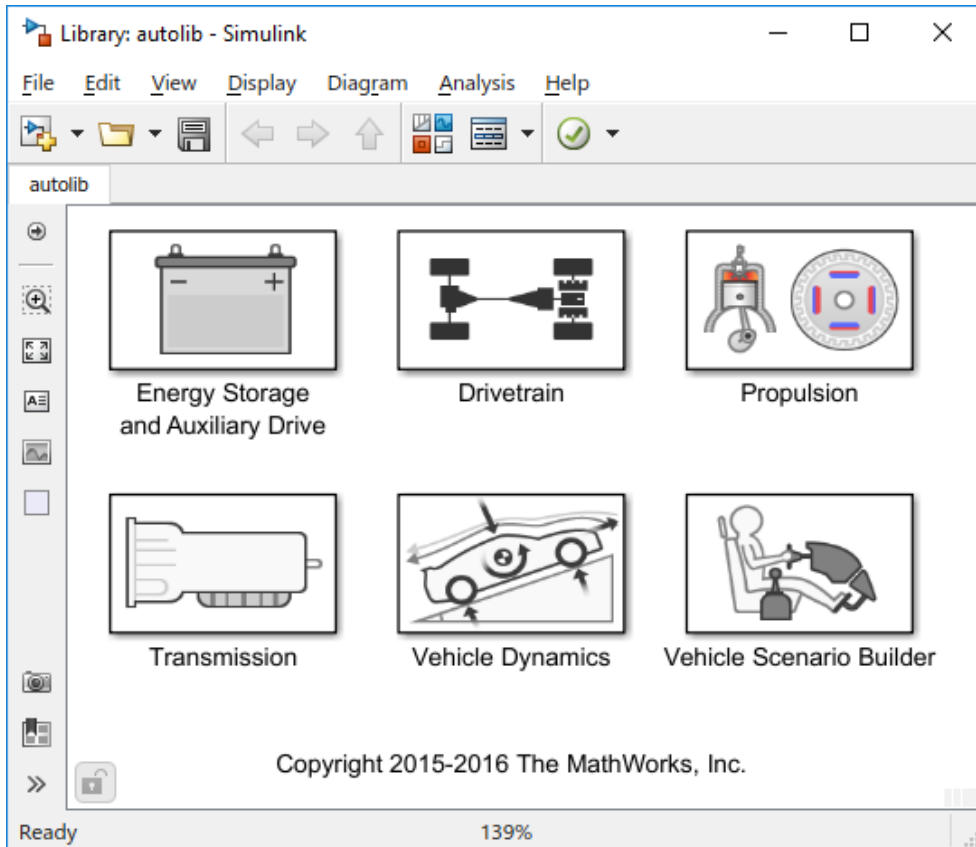
- Goals:
 - Provide starting point for engineers to build **good plant / controller models**
 - Provide **open** and documented models
 - Provide very **fast**-running models that work with popular HIL systems



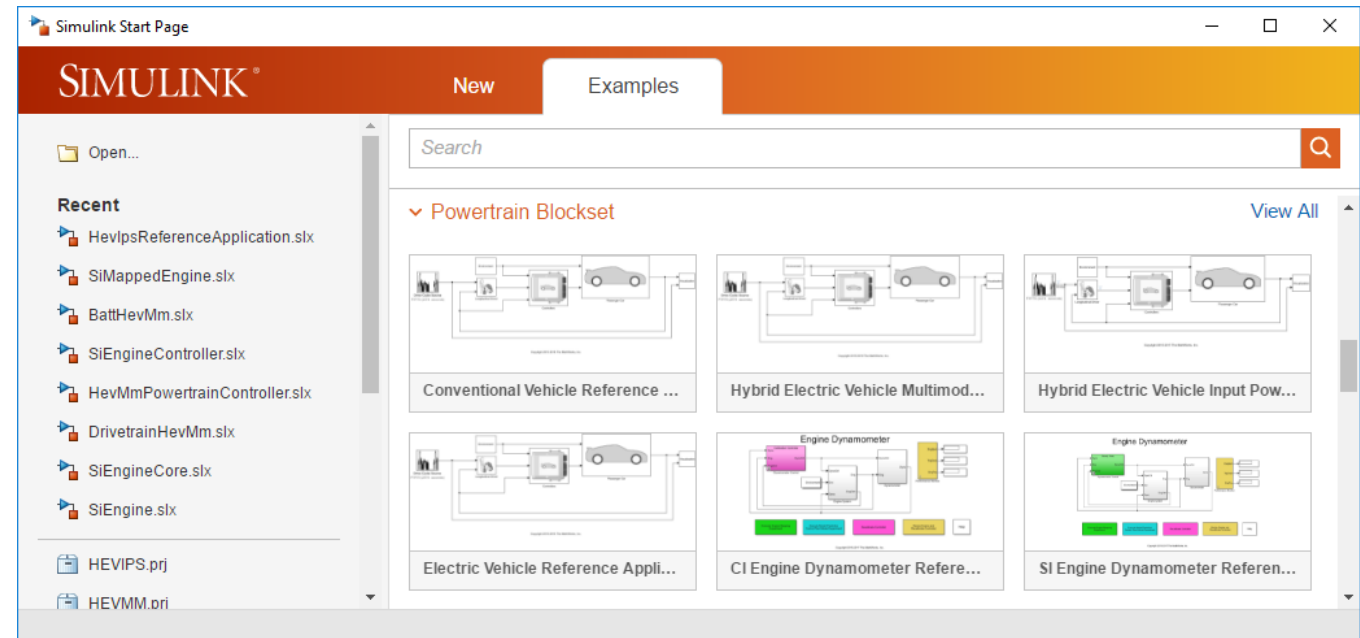
Lower the barrier to entry for Model-Based Design

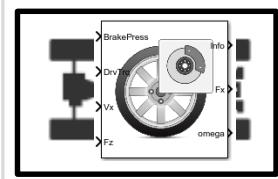
Powertrain Blockset Features

Library of blocks

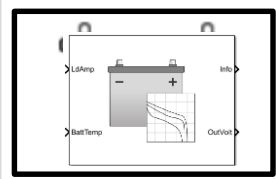


Pre-built reference applications

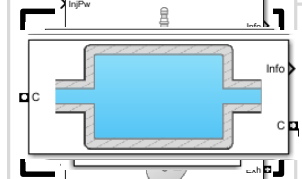




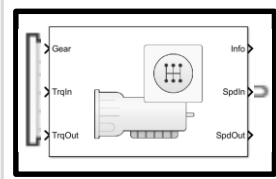
Drivetrain



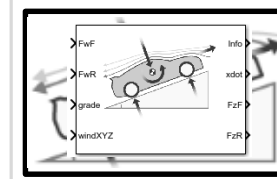
Energy Storage and Auxiliary Drive



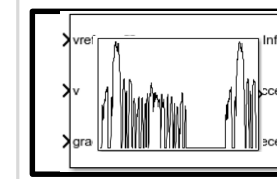
Propulsion



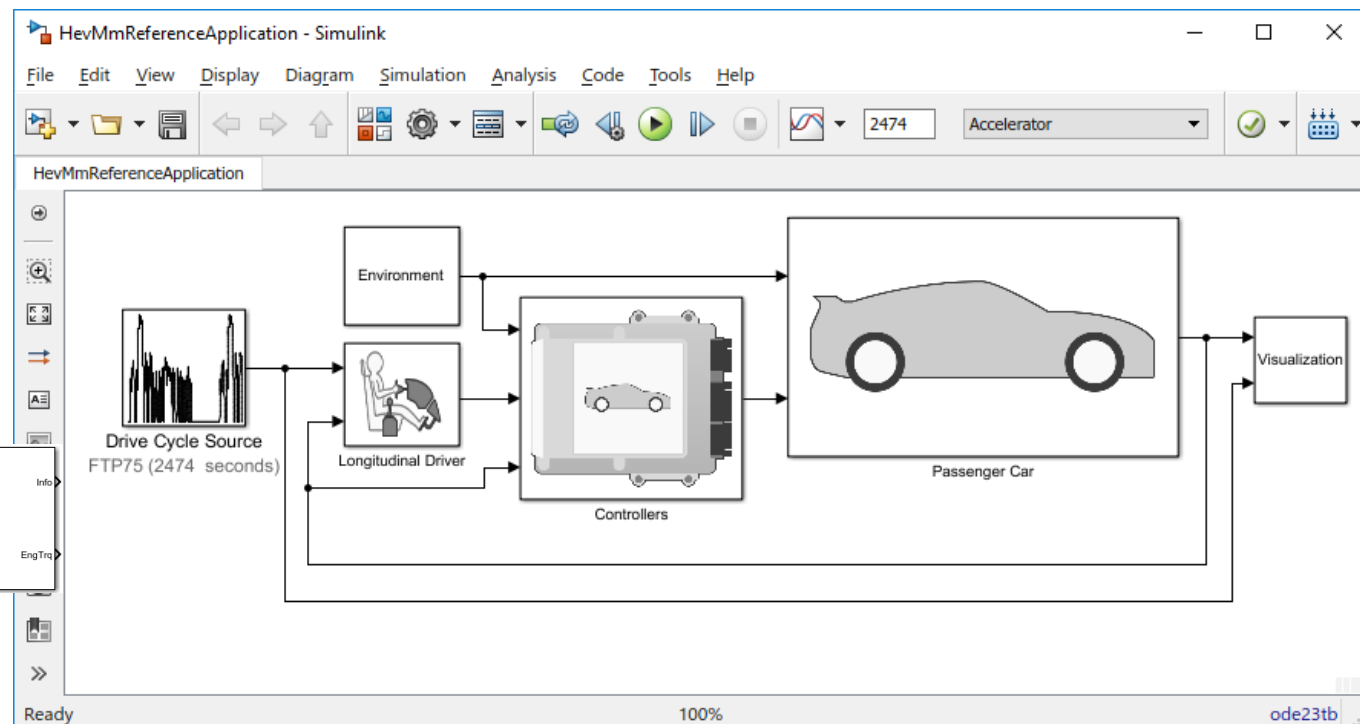
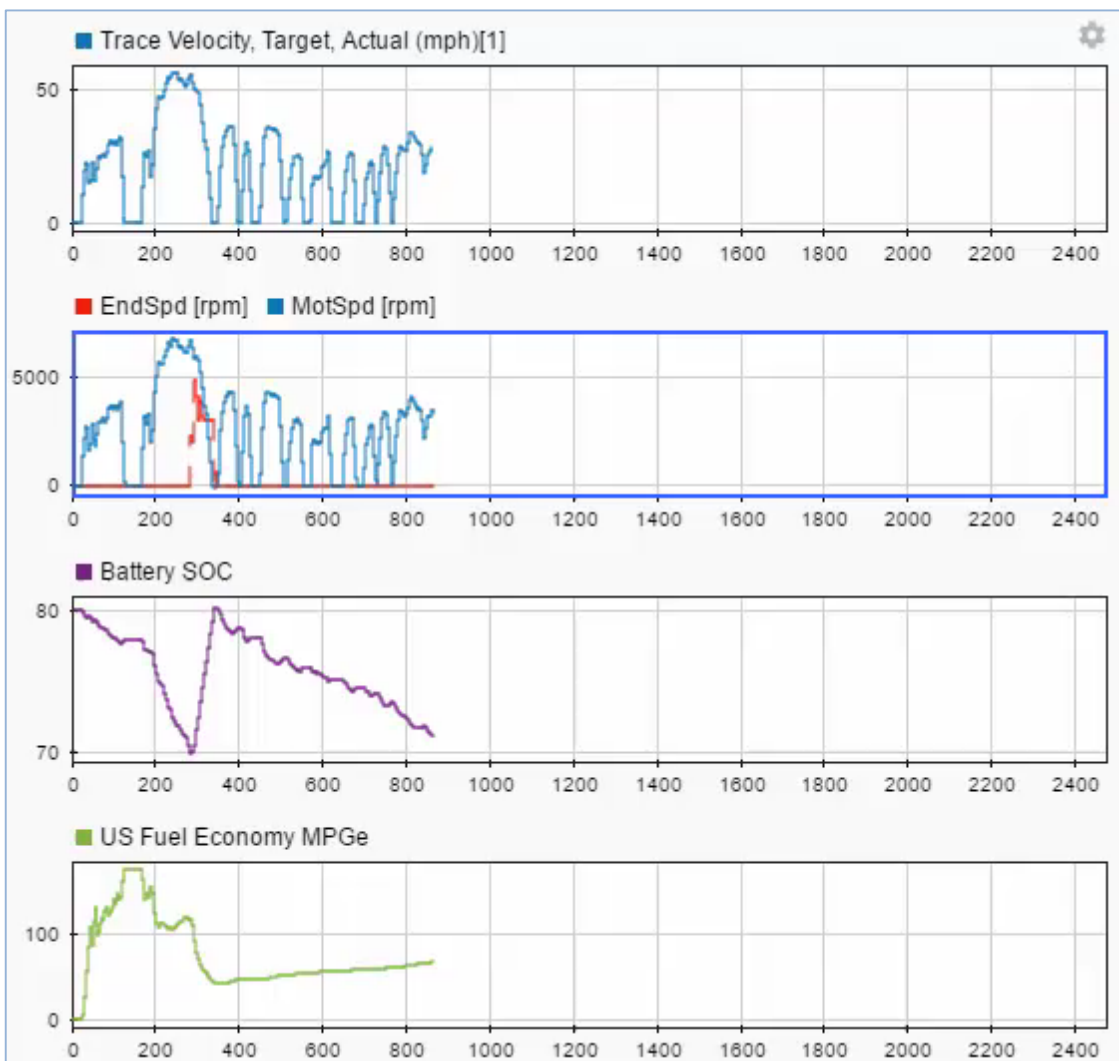
Transmission



Vehicle Dynamics



Vehicle Scenario Builder



Reference Applications

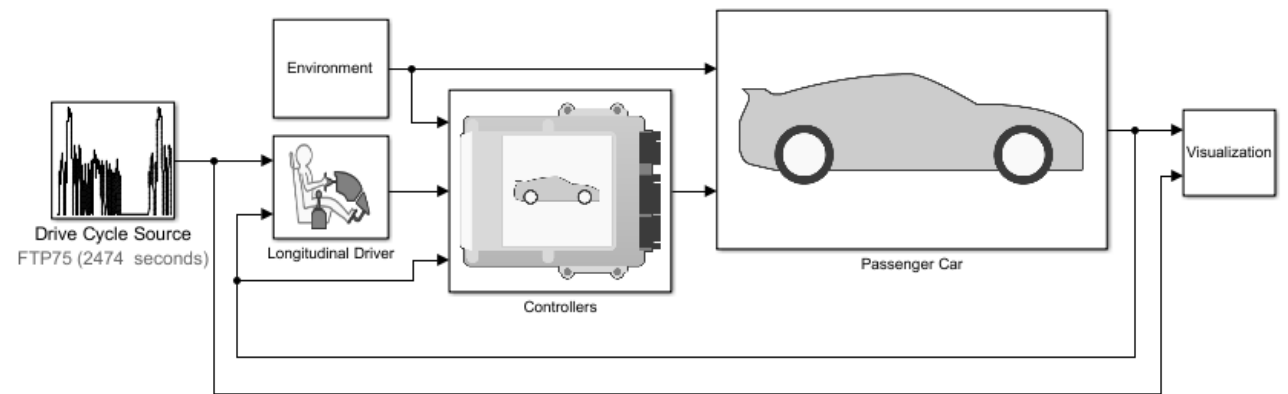
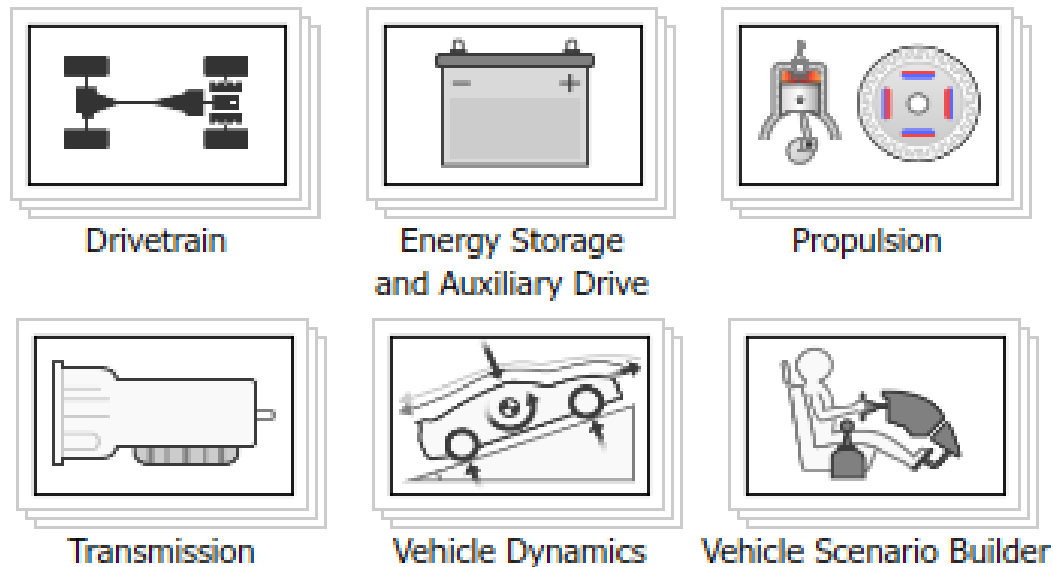
- Full vehicle models (conventional, EV, multi-mode HEV, input power-split HEV)
- Virtual engine dynamometers (compression ignition, spark ignition)

The screenshot displays the Simulink Start Page interface. The title bar reads "Simulink Start Page". The main header features the "SIMULINK" logo and two tabs: "New" and "Examples". A search bar is located below the tabs. On the left side, there is a sidebar with an "Open..." folder icon, a "Recent" list of files, and project files "HEVIPS.prj" and "HEVMM.ori". The main content area is titled "Powertrain Blockset" and contains a grid of six reference application thumbnails. Each thumbnail shows a block diagram of a vehicle or engine system. The thumbnails are labeled as follows:

- Conventional Vehicle Reference ...
- Hybrid Electric Vehicle Multimod...
- Hybrid Electric Vehicle Input Pow...
- Electric Vehicle Reference Appli...
- CI Engine Dynamometer Referen...
- SI Engine Dynamometer Referen...

Powertrain Blockset Value Proposition

- **Open** and documented library of component and subsystem models
- Prebuilt vehicle models that you can parameterize and **customize**
- **Fast**-running models that are ready for HIL deployment



Vehicle Dynamics Blockset

Vehicle Dynamics Blockset

New product (R2018a)

- Model and simulate vehicle dynamics in a virtual 3D environment
- Use Vehicle Dynamics Blockset for:
 - Ride & handling: characterize vehicle performance under standard driving maneuvers
 - Chassis controls: design and test chassis control systems
 - ADAS / AD: create virtual 3D test ground for ADAS and automated driving features



Ride & handling

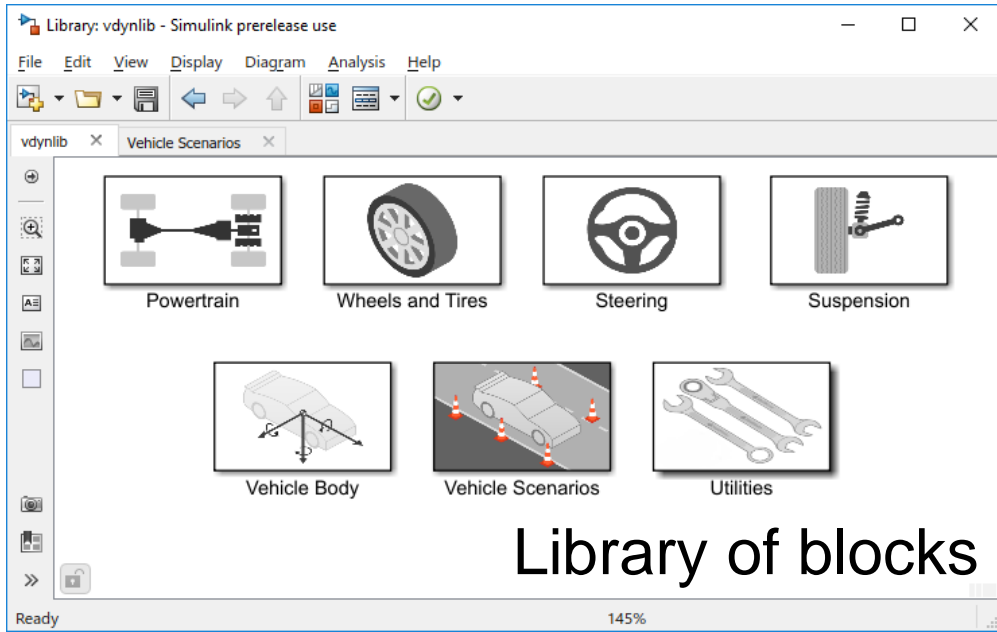


Chassis controls

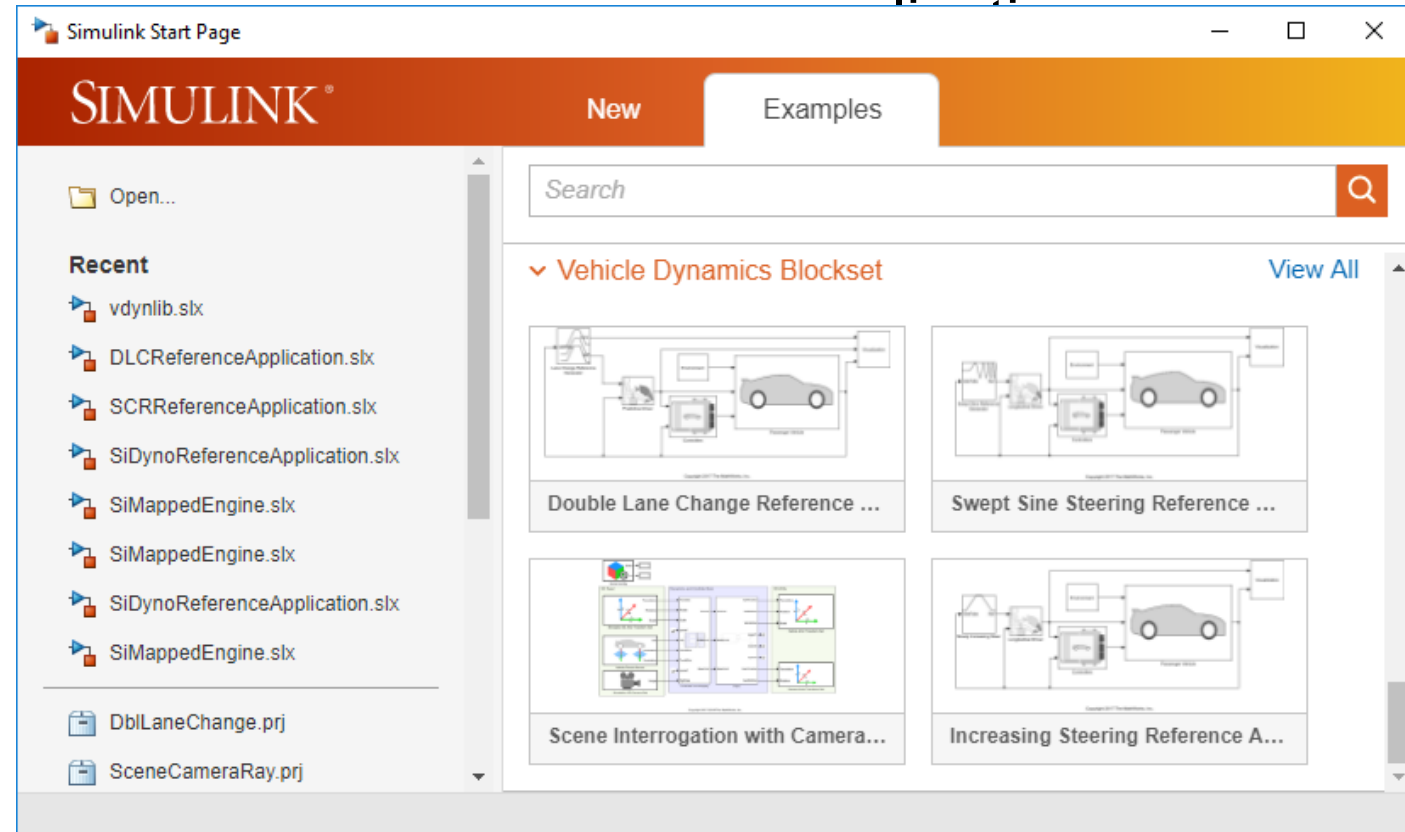


ADAS / AD

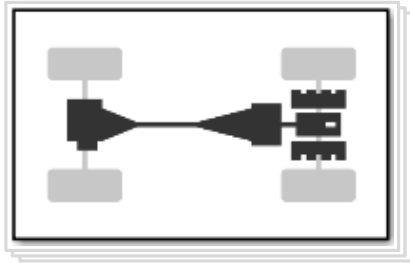
Vehicle Dynamics Blockset Features



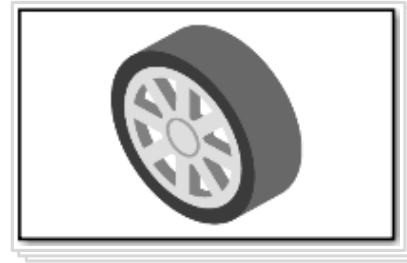
Pre-built reference



Block Library



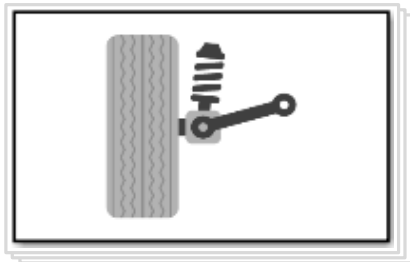
Powertrain



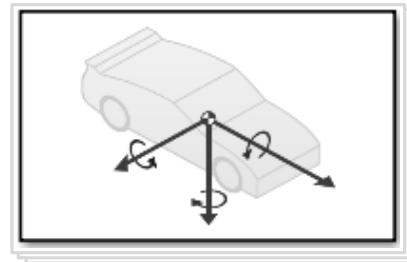
Wheels and Tires



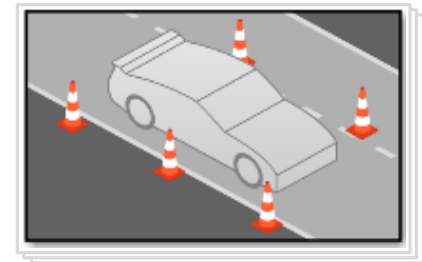
Steering



Suspension



Vehicle Body



Vehicle Scenarios

Game Engine Co-Simulation

Simulink

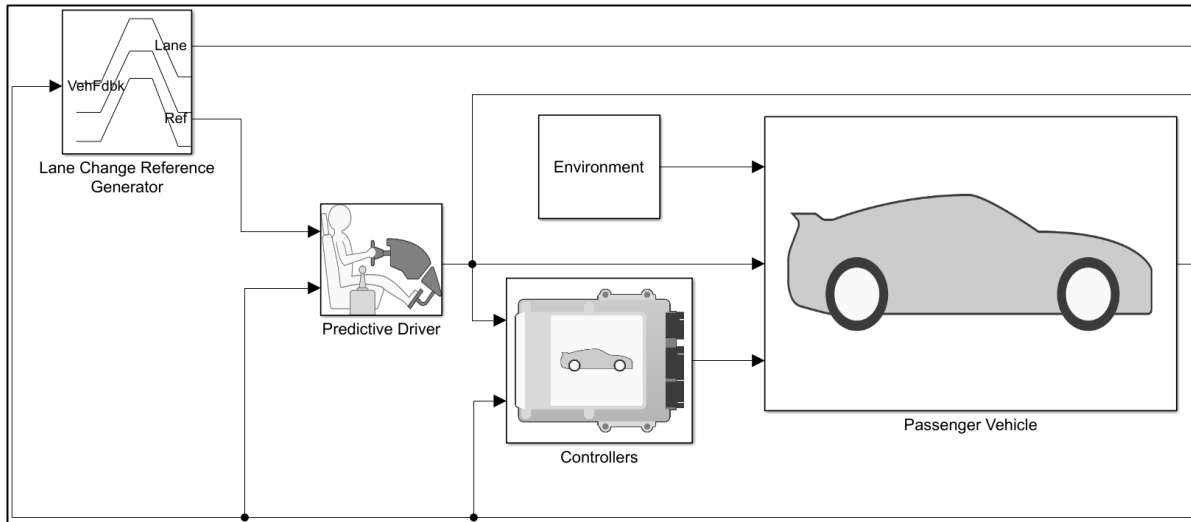
- Physics of vehicle
- Initialization of game engine camera

vehicle / camera location

Unreal Engine

- Rendering / lighting
- Physics of non-Simulink objects
- Collision detection

camera image, ground height, ...

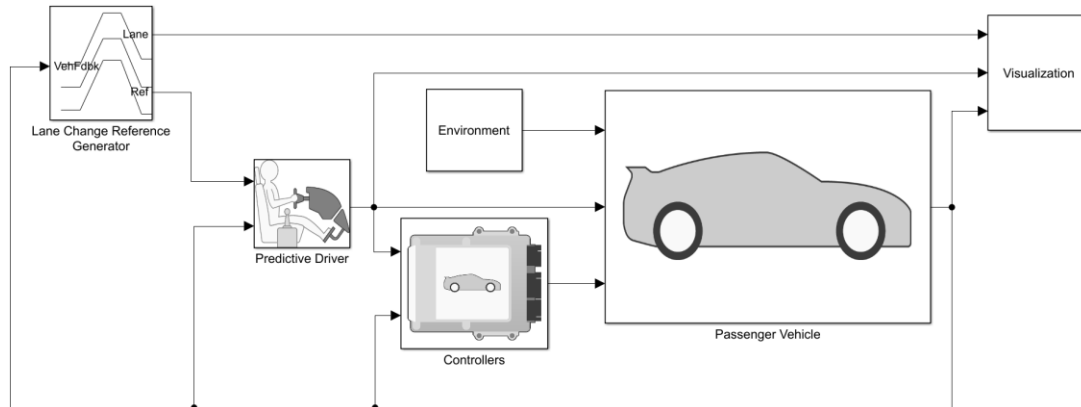


Reference Applications

Vehicle Maneuvers

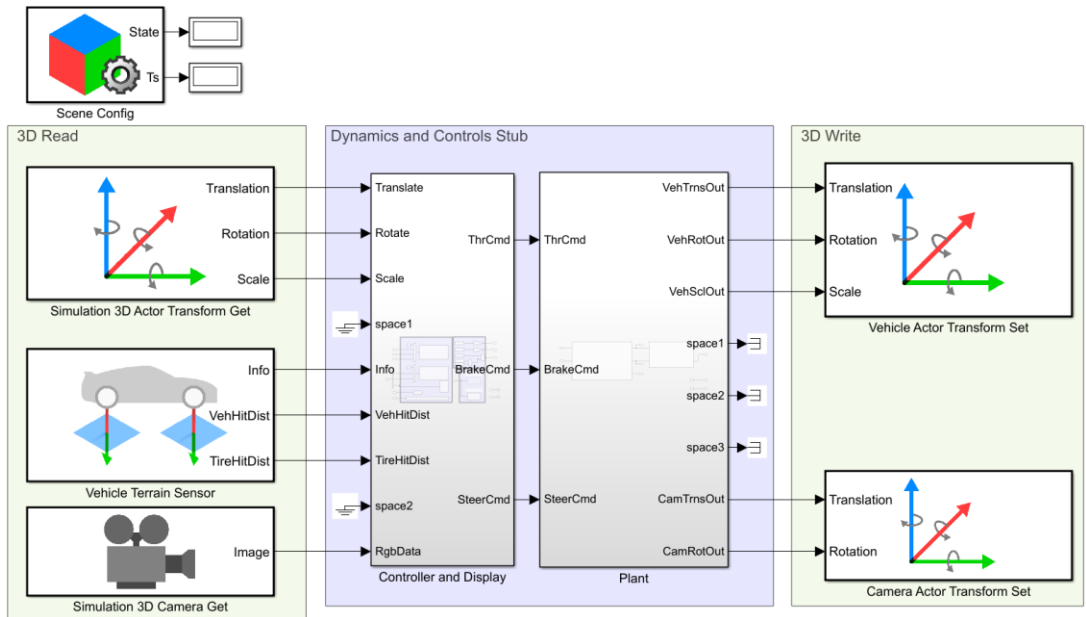
Analyze ride and handling on driving maneuvers such as:

- Double-lane change
- Swept sine steering
- Slowly increasing steering



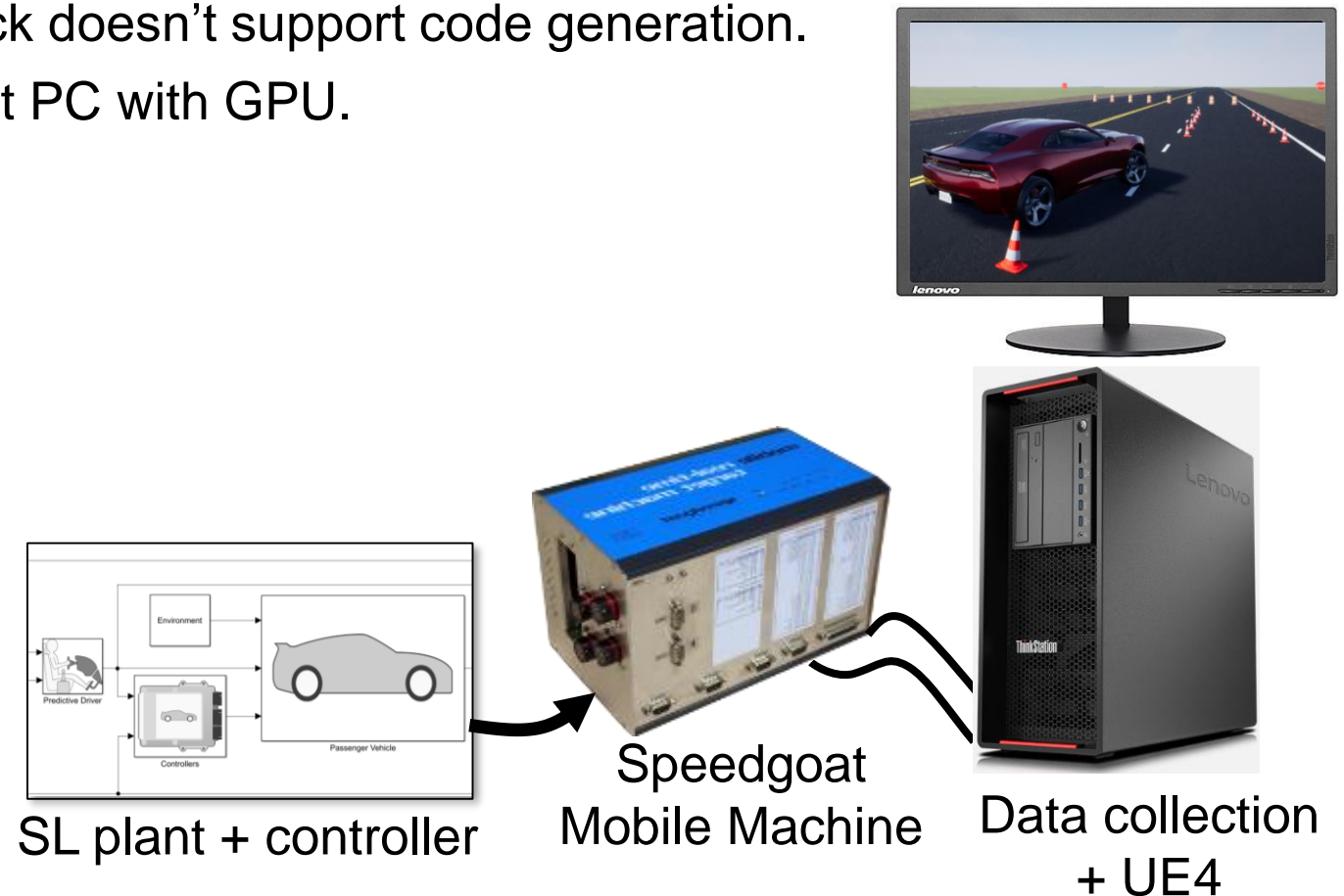
Scene Interrogation

Configure the interface to the 3D environment



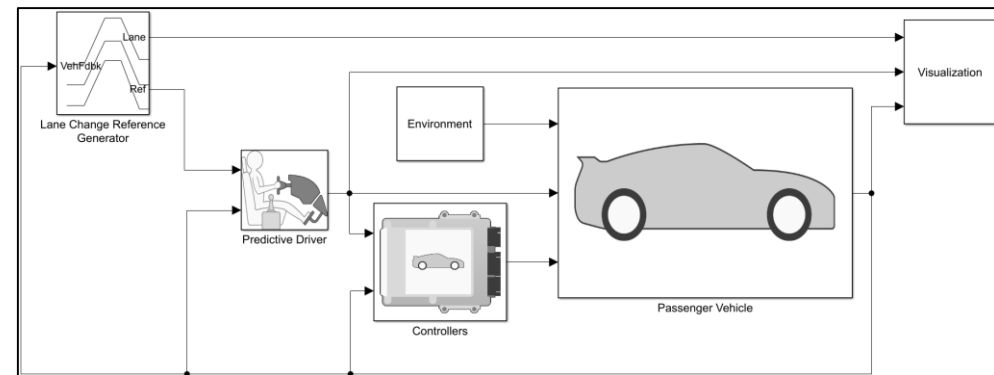
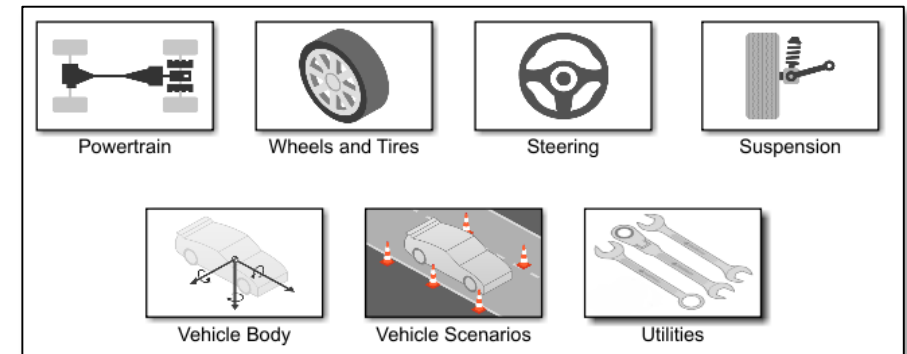
HIL Testing with UE

- Can you perform HIL testing with Unreal Engine running?
 - Yes!, but UE visualization block doesn't support code generation.
 - Unreal Engine can run on host PC with GPU.



Summary

- Vehicle Dynamics Blockset provides:
 - **Open** and documented library of component and subsystem models
 - Pre-built vehicle models that you can parameterize and **customize**
 - **Fast**-running models that are ready for HIL deployment
 - Interface to **Unreal Engine**



What's New for *Embedded Coder* in R2018b

Defining Code Generation Behavior Without Packages and Classes

- Coder dictionary

Code behavior are categorized into various abstractions

The screenshot displays the Coder Dictionary interface. At the top, there are icons for Add, Duplicate, Remove, and Manage. Below these are tabs for EDIT and PACKAGE. The PACKAGE tab is active, showing three sub-tabs: Storage Classes, Function Customization Templates, and Memory Sections. A search bar is located below the sub-tabs. The main area contains a table of storage classes. The 'Observable' class is highlighted. To the right, the PROPERTY INSPECTOR shows the properties for the selected 'Observable' class.

Name	Storage T...	Data S...	Header File	Definition File	Data Ini...	Memory ...	Source
Volatile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemVolatile	Simulink package
Struct	FlatStructure	Exported	---	---	Auto	None	Simulink package
Reusable	Unstructured	<Instance s...	<Instance specific>	<Instance specific>	Dynamic	None	Simulink package
Observable	Structured	Exported	\$N.h	\$N.c	Dynamic	None	sensor
ImportFromFile	Unstructured	Imported	<Instance specific>	---	Auto	None	Simulink package
ImportedExternPointer	Unstructured	Imported	---	---	Auto	None	Built-in

PROPERTY INSPECTOR

Name: Observable

Description: Description

Source: sensor

Storage Type: Structured

Data Initialization: Dynamic

Data Scope: Exported

Header File:

Defining Code Generation Behavior Without Packages and Classes

- Coder dictionary

The screenshot shows the MATLAB Coder Dictionary interface. At the top, there are icons for 'Add', 'Duplicate', 'Remove', and 'Manage'. Below these are tabs for 'EDIT' and 'PACKAGE'. The main area is divided into sections: 'Storage Classes', 'Function Customization Templates', and 'Memory Sections'. A search bar is present. A table lists various storage classes with columns for Name, Storage Type, Data Scope, and Source. The 'Observable' class is highlighted. To the right, the 'PROPERTY INSPECTOR' for the 'Observable' class is shown, with fields for Name, Description, Source, Storage Type, Data Initialization, Data Scope, and Header File.

Name	Storage T...	Data S...	Source
Volatile	Unstructured	Exported	Simulink package
Struct	FlatStructure	Exported	Simulink package
Reusable	Unstructured	<Instance s...	Simulink package
Observable	Structured	Exported	sensor
ImportFromFile	Unstructured	Imported	Simulink package
ImportedExternPointer	Unstructured	Imported	

PROPERTY INSPECTOR

Name	Observable
Description	Description
Source	sensor
Storage Type	Structured
Data Initialization	Dynamic
Data Scope	Exported
Header File

Users can define named behavior pattern within each category

All available customization in one place for full specification

Govern by data model to ensure correctness by design

Allow user specified usage constraints (18b)

Defining Code Generation Behavior Without Packages and Classes

- Coder dictionary

The screenshot displays the MATLAB Coder Dictionary interface. The main window is titled 'DICTIONARY' and contains a toolbar with 'Add', 'Duplicate', 'Remove', and 'Manage' buttons. Below the toolbar are tabs for 'Storage Classes', 'Function Customization Templates', and 'Memory Sections'. A search bar is located at the top left of the table area. The table lists various named behavior patterns with columns for Name, Storage Type, Data Scope, Header File, Definition File, Data Initialization, Memory, and Source. The 'Observable' pattern is highlighted in blue. To the right of the table is the 'PROPERTY INSPECTOR' for the selected 'Observable' pattern, showing fields for Name, Description, Source, Storage Type, Data Initialization, Data Scope, and Header File.

Name	Storage T...	Data S...	Header File	Definition File	Data Ini...	Memory ...	Source
Volatile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemVolatile	Simulink package
Struct	FlatStructure	Exported	---	---	Auto	None	Simulink package
Reusable	Unstructured	<Instance s...	<Instance specific>	<Instance specific>	Dynamic	None	Simulink package
Observable	Structured	Exported	\$N.h	\$N.c	Dynamic	None	sensor
ImportFromFile	Unstructured	Imported	<Instance specific>	---	Auto	None	Simulink package
ImportedExternPointer	Unstructured	Imported	---	---	Auto	None	Built-in

PROPERTY INSPECTOR

Name: Observable

Description: Description

Source: sensor

Storage Type: Structured

Data Initialization: Dynamic

Data Scope: Exported

Header File:

Overview and easy management of all available named behavior patterns

Dictionary Defaults with Embedded Coder Dictionary

R2018b

Configure dictionary defaults
Set default code mappings on model data and model functions for code generation.

Data Defaults **Function Defaults**

Model Element Category	Storage Class
Inports	myStorageClass
Outports	myStorageClass
Global parameters	Default
Local parameters	ExportedGlobal
Shared local data stores	Default
Global data stores	Default
Internal data	Default
Constants	Default

Storage Class: myStorageClass

OK Cancel Help App



Copyright 1994-2017 The MathWorks, Inc.

Code Mappings - C

Entry-Point Functions **Data Defaults** Function Defaults

Filter contents

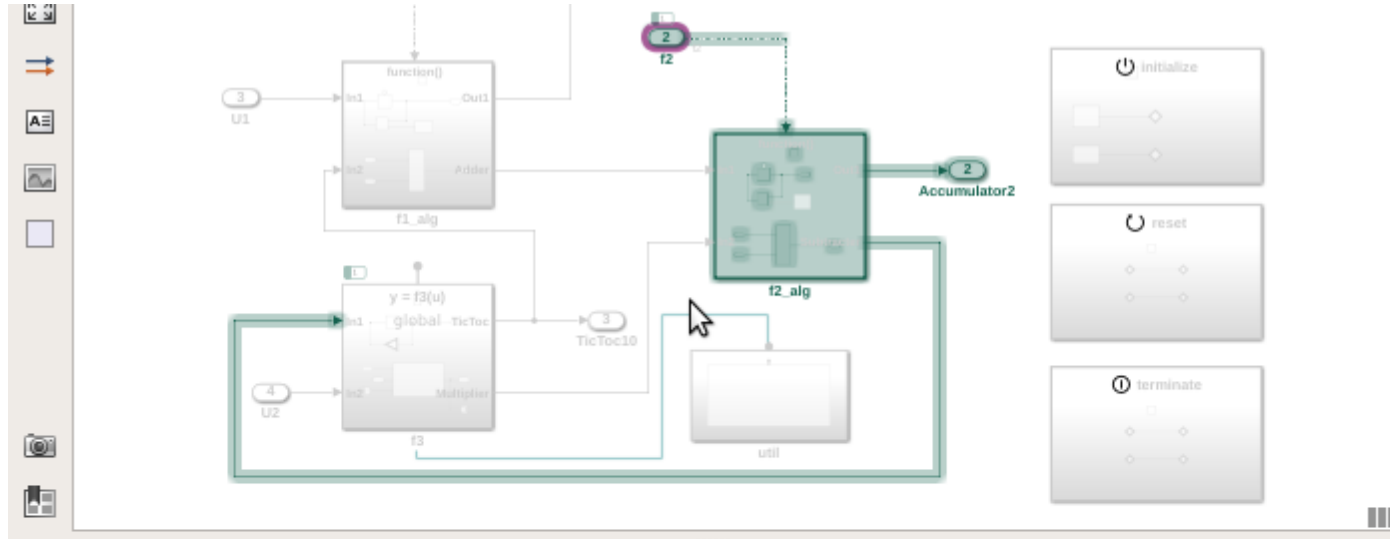
Model Element Category	Storage Class
Inports	Dictionary default: myStorageClass
Outports	Dictionary default: myStorageClass
Global parameters	Dictionary default: Default
Local parameters	Dictionary default: ExportedGlobal
Parameter arguments	Dictionary default: Default

Code Mappings - C Model Data Editor

Ready View diag

Direct Specification of Each Function Interface

- View all entry point functions in-canvas
- Configure prototype and memory section of each function

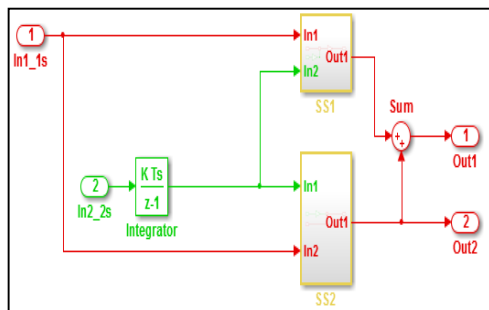


The screenshot displays a Simulink model with several blocks: 'function()', 'f1_alg', 'f2_alg', 'f3', 'util', 'TicToc10', 'Multiplier', and 'Accumulator2'. The 'f2_alg' block is highlighted with a green border. To the right of the canvas are three control panels: 'initialize', 'reset', and 'terminate'. Below the canvas is the 'Code Mappings - C' window, which has three tabs: 'Entry-Point Functions', 'Data Defaults', and 'Function Defaults'. The 'Entry-Point Functions' tab is active, showing a table of function mappings.

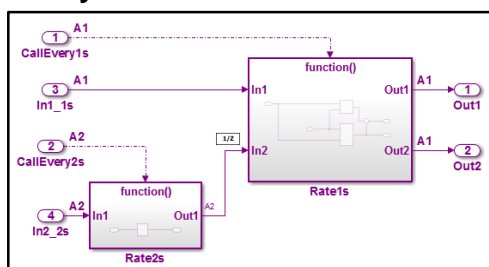
Source	Function Customization	Function Name
Exported Function:f1	ComponentFcn	f1
Exported Function:f2	ComponentFcn	f2_2sec
f3	ComponentFcn	f3
Initialize Function	ComponentFcn	initialize

Uniform Function and Data Specification for All Modeling Styles

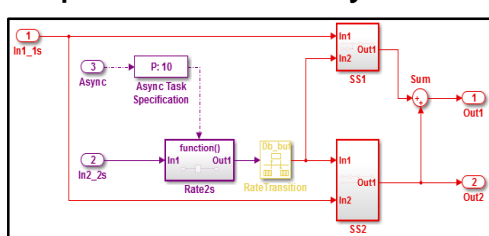
Periodic rate-based



Asynchronous functions



Mixed periodic and asynchronous



Code Mappings - C

Entry-Point Functions | Data Defaults | Function Defaults

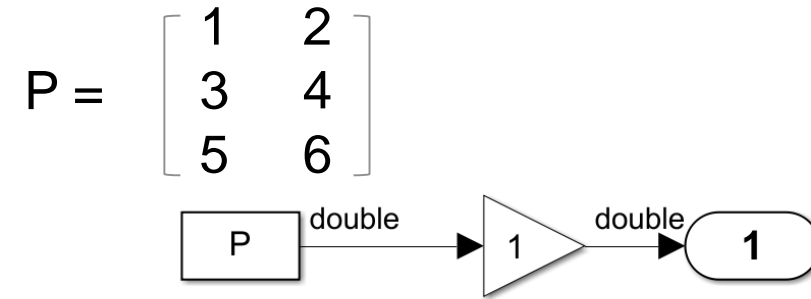
Filter contents

	Source	Function Customization Template	Function Name
fx	Initialize Function	Model default: Default	initialize
fx	Step Function [Sample Time:1s]	Model default: Default	CallEvery1s
fx	Step Function [Sample Time:2s]	Model default: Default	CallEvery2s
fx	Terminate Function	Model default: Default	terminate



```
void CallEvery1s(void);
void CallEvery2s(void);
```

Row Major Code Generation



Column major layout

```

/* ... parameters (as) and storage */
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};

```

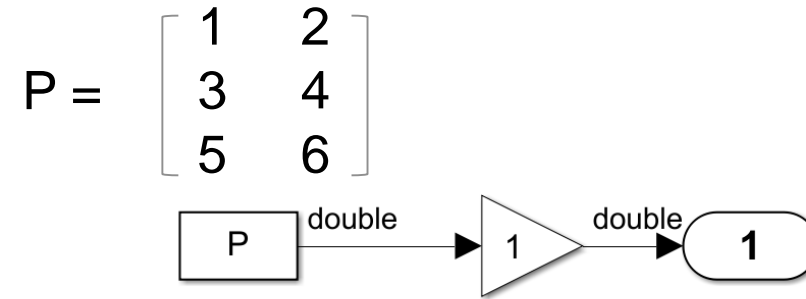
Row major layout

```

P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }
};

```

Row Major Code Generation and Multidimension Indexing



Row major layout

```

P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }
};
  
```

Code generation options

Storage class: ExportToFile (Custom)

Custom attributes

HeaderFile:

DefinitionFile:

Owner:

Preserve Array Dimensions

```

function for custom storage class. ExportToFile /
P[3][2] = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } };
  
```