

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

Reference



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

R2019b



# How to Contact MathWorks



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



Phone: 508-647-7000



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

*Simulink® Check™ Reference*

© COPYRIGHT 2004–2019 by The MathWorks, Inc.

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

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

## Trademarks

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

## Patents

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

## Revision History

September 2017	Online only	New for Version 4.0 (Release 2017b)
March 2018	Online only	Revised for Version 4.1 (Release 2018a)
September 2018	Online only	Revised for Version 4.2 (Release 2018b)
March 2019	Online only	Revised for Version 4.3 (Release 2019a)
September 2019		Revised for Version 4.4 (Release 2019b)

## Functions – Alphabetical List

1

## Model Advisor Checks

2

<b>Simulink Check Checks</b> .....	2-2
Simulink Check Checks .....	2-2
Simulink Requirements Checks .....	2-3
Modeling Standards Checks .....	2-3
Modeling Standards for MAAB .....	2-3
Naming Conventions .....	2-4
Model Architecture .....	2-4
Model Configuration Options .....	2-5
Simulink .....	2-5
Stateflow .....	2-5
MATLAB Functions .....	2-6
Modeling Standards for JMAAB .....	2-6
<b>DO-178C/DO-331 Checks</b> .....	2-8
DO-178C/DO-331 Checks .....	2-8
Check safety-related code generation settings .....	2-8
Check usage of Math Operations blocks .....	2-14
Check usage of Logic and Bit Operations blocks .....	2-17
Check usage of Ports and Subsystems blocks .....	2-19
Display model version information .....	2-22
<b>High Integrity System Modeling Checks</b> .....	2-24
High-Integrity Systems Modeling Checks .....	2-27
Split Checks for High Integrity Systems Modeling .....	2-28
Check for inconsistent vector indexing methods .....	2-30
Check for root Inports with missing properties .....	2-32

Check for root Inports with missing range definitions . . . . .	2-33
Check for root Outports with missing range definitions . . . . .	2-35
Check safety-related diagnostic settings for data store memory . . . . .	2-36
Check safety-related diagnostic settings for data used for debugging . . . . .	2-38
Check safety-related diagnostic settings for parameters . . . .	2-39
Check safety-related solver settings for solver options . . . . .	2-41
Check safety-related solver settings for tasking and sample-time . . . . .	2-42
Check usage of shift operations for Stateflow data . . . . .	2-42
Check usage of Signal Routing blocks . . . . .	2-43
Check state machine type of Stateflow charts . . . . .	2-44
Check Stateflow charts for ordering of states and transitions . . . . .	2-46
Check Stateflow charts for strong data typing . . . . .	2-47
Check Stateflow charts for unary operators . . . . .	2-48
Check Stateflow charts for uniquely defined data objects . . .	2-49
Check Stateflow debugging options . . . . .	2-50
Check usage of lookup table blocks . . . . .	2-51
Check for variant blocks with 'Generate preprocessor conditionals' active . . . . .	2-53
Check safety-related diagnostic settings for signal connectivity . . . . .	2-54
Check safety-related diagnostic settings for bus connectivity . . . . .	2-55
Check safety-related diagnostic settings that apply to function- call connectivity . . . . .	2-57
Check safety-related diagnostic settings for compatibility . . .	2-58
Check safety-related diagnostic settings for model initialization . . . . .	2-59
Check safety-related diagnostic settings for saving . . . . .	2-61
Check MATLAB Code Analyzer messages . . . . .	2-62
Check safety-related diagnostic settings for Merge blocks . .	2-64
Check safety-related diagnostic settings for Stateflow . . . . .	2-65
Check for Strong Data Typing with Simulink I/O . . . . .	2-67
Check for MATLAB Function interfaces with inherited properties . . . . .	2-68
Check usage of Abs blocks . . . . .	2-69
Check usage of Math Function blocks (rem and reciprocal functions) . . . . .	2-71
Check usage of Math Function blocks (log and log10 functions) . . . . .	2-72
Check usage of Assignment blocks . . . . .	2-73

Check safety-related optimization settings for data type conversions . . . . .	<b>2-74</b>
Check safety-related optimization settings for data initialization . . . . .	<b>2-75</b>
Check safety-related optimization settings for application lifespan . . . . .	<b>2-77</b>
Check safety-related block reduction optimization settings . . . . .	<b>2-78</b>
Check safety-related optimization settings for logic signals . . . . .	<b>2-79</b>
Check safety-related optimization settings for division arithmetic exceptions . . . . .	<b>2-80</b>
Check usage of Logical Operator blocks . . . . .	<b>2-81</b>
Check for Relational Operator blocks that equate floating-point types . . . . .	<b>2-82</b>
Check usage of Relational Operator blocks . . . . .	<b>2-83</b>
Check usage of Switch Case blocks and Switch Case Action Subsystem blocks . . . . .	<b>2-84</b>
Check usage of If blocks and If Action Subsystem blocks . . . . .	<b>2-85</b>
Check usage of For Iterator blocks . . . . .	<b>2-86</b>
Check usage of For and While Iterator subsystems . . . . .	<b>2-87</b>
Check usage of While Iterator blocks . . . . .	<b>2-89</b>
Check safety-related code generation settings for comments . . . . .	<b>2-90</b>
Check safety-related code generation interface settings . . . . .	<b>2-91</b>
Check safety-related code generation settings for code style . . . . .	<b>2-93</b>
Check safety-related code generation identifier settings . . . . .	<b>2-95</b>
Check safety-related diagnostic settings for model referencing . . . . .	<b>2-96</b>
Check safety-related diagnostic settings for sample time . . . . .	<b>2-98</b>
Check MATLAB Function metrics . . . . .	<b>2-100</b>
Check safety-related diagnostic settings for type conversions . . . . .	<b>2-102</b>
Check safety-related solver settings for simulation time . . . . .	<b>2-103</b>
Check safety-related diagnostic settings for solvers . . . . .	<b>2-104</b>
Check safety-related model referencing settings . . . . .	<b>2-106</b>
Check Stateflow charts for transition paths that cross parallel state boundaries . . . . .	<b>2-108</b>
Check assignment operations in Stateflow Charts . . . . .	<b>2-109</b>
Check usage of bit operation blocks . . . . .	<b>2-110</b>
Check data types for blocks with index signals . . . . .	<b>2-111</b>
Check model file name . . . . .	<b>2-112</b>
Check if/elseif/else patterns in MATLAB Function blocks . . . . .	<b>2-113</b>
Check switch statements in MATLAB Function blocks . . . . .	<b>2-114</b>
Check global variables in graphical functions . . . . .	<b>2-115</b>

Check for length of user-defined object names . . . . .	2-116
Check usage of Merge blocks . . . . .	2-116
Check usage of conditionally executed subsystems . . . . .	2-117
Check usage of standardized MATLAB function headers . . .	2-120
Check usage of relational operators in MATLAB Function blocks . . . . .	2-121
Check usage of equality operators in MATLAB Function blocks . . . . .	2-122
Check usage of logical operators and functions in MATLAB Function blocks . . . . .	2-123
Check type and size of condition expressions . . . . .	2-124
Check naming of ports in Stateflow charts . . . . .	2-126
Check scoping of Stateflow data objects . . . . .	2-127
Check usage of Gain blocks . . . . .	2-127
Check data type of loop control variables . . . . .	2-128
Check for inappropriate use of transition paths . . . . .	2-129
Check usage of bitwise operations in Stateflow charts . . . .	2-130
Check safety-related diagnostic settings for signal data . . .	2-131
Check for model elements that do not link to requirements	2-134
Check safety-related optimization settings for Loop unrolling threshold . . . . .	2-136
Check safety-related optimization settings for specified minimum and maximum values . . . . .	2-137
Check model object names . . . . .	2-138
Check for blocks not recommended for C/C++ production code deployment . . . . .	2-141
Check configuration parameters for MISRA C:2012 . . . . .	2-142
Check for blocks not recommended for MISRA C:2012 . . . .	2-146
<b>IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks .</b>	<b>2-149</b>
IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks .	2-149
Display model metrics and complexity report . . . . .	2-150
Check for unconnected objects . . . . .	2-151
Check safety-related code generation settings . . . . .	2-152
Check usage of Math Operations blocks . . . . .	2-156
Check usage of Logic and Bit Operations blocks . . . . .	2-158
Check usage of Ports and Subsystems blocks . . . . .	2-160
Display configuration management data . . . . .	2-163
<b>MathWorks Automotive Advisory Board Checks . . . . .</b>	<b>2-165</b>
MathWorks Automotive Advisory Board Checks . . . . .	2-167
Check font formatting . . . . .	2-168
Check transition orientations in flow charts . . . . .	2-170
Check for nondefault block attributes . . . . .	2-171

Check usage of merge blocks . . . . .	<b>2-172</b>
Check signal line labels . . . . .	<b>2-174</b>
Check for propagated signal labels . . . . .	<b>2-176</b>
Check default transition placement in Stateflow charts . . . . .	<b>2-177</b>
Check return value assignments in Stateflow graphical functions . . . . .	<b>2-178</b>
Check entry formatting in State blocks in Stateflow charts	<b>2-179</b>
Check usage of return values from Stateflow graphical functions . . . . .	<b>2-180</b>
Check for pointers in Stateflow charts . . . . .	<b>2-181</b>
Check for event broadcasts in Stateflow charts . . . . .	<b>2-182</b>
Check transition actions in Stateflow charts . . . . .	<b>2-183</b>
Check for MATLAB expressions in Stateflow charts . . . . .	<b>2-184</b>
Check for indexing in blocks . . . . .	<b>2-185</b>
Check file names . . . . .	<b>2-187</b>
Check folder names . . . . .	<b>2-188</b>
Check for prohibited blocks in discrete controllers . . . . .	<b>2-189</b>
Check for prohibited sink blocks . . . . .	<b>2-190</b>
Check positioning and configuration of ports . . . . .	<b>2-192</b>
Check for matching port and signal names . . . . .	<b>2-193</b>
Check whether block names appear below blocks . . . . .	<b>2-194</b>
Check for mixing basic blocks and subsystems . . . . .	<b>2-195</b>
Check for unconnected ports and signal lines . . . . .	<b>2-196</b>
Check position of Trigger and Enable blocks . . . . .	<b>2-197</b>
Check usage of tunable parameters in blocks . . . . .	<b>2-198</b>
Check Stateflow data objects with local scope . . . . .	<b>2-200</b>
Check for Strong Data Typing with Simulink I/O . . . . .	<b>2-201</b>
Check usage of exclusive and default states in state machines . . . . .	<b>2-202</b>
Check Implement logic signals as Boolean data (vs. double)	<b>2-204</b>
Check model diagnostic parameters . . . . .	<b>2-204</b>
Check the display attributes of block names . . . . .	<b>2-207</b>
Check display for port blocks . . . . .	<b>2-209</b>
Check subsystem names . . . . .	<b>2-210</b>
Check port block names . . . . .	<b>2-211</b>
Check character usage in signal labels . . . . .	<b>2-213</b>
Check Simulink bus signal names . . . . .	<b>2-214</b>
Check character usage in block names . . . . .	<b>2-216</b>
Check Trigger and Enable block names . . . . .	<b>2-219</b>
Check for Simulink diagrams using nonstandard display attributes . . . . .	<b>2-220</b>
Check MATLAB code for global variables . . . . .	<b>2-222</b>
Check visibility of block port names . . . . .	<b>2-223</b>
Check orientation of Subsystem blocks . . . . .	<b>2-224</b>

Check usage of Relational Operator blocks . . . . .	2-225
Check usage of Switch blocks . . . . .	2-226
Check usage of buses and Mux blocks . . . . .	2-227
Check for bitwise operations in Stateflow charts . . . . .	2-228
Check fundamental logical and numerical operations . . . . .	2-230
Check logical expressions in If blocks . . . . .	2-232
Check for comparison operations in Stateflow charts . . . . .	2-235
Check usage of restricted variable names . . . . .	2-236
Check unused ports in Variant Subsystems . . . . .	2-237
Check usage of character vector inside MATLAB Function block . . . . .	2-238
Check usage of recommended patterns for Switch/Case statements . . . . .	2-239
Check use of default variants . . . . .	2-240
Check use of single variable variant conditionals . . . . .	2-242
Check nested states in Stateflow charts . . . . .	2-244
Check use of Simulink in Stateflow charts . . . . .	2-245
Check number of Stateflow states per container . . . . .	2-246
Check usage of unary minus operations in Stateflow charts . . . . .	2-247
Check usage of floating-point expressions in Stateflow charts . . . . .	2-248
Check input and output settings of MATLAB Functions . . . . .	2-249
Check MATLAB Function metrics . . . . .	2-250
Check for names of Stateflow ports and associated signals . . . . .	2-252
Check scope of From and Goto blocks . . . . .	2-253
Check the number of function calls in MATLAB Function blocks . . . . .	2-254
Check usage of Goto and From blocks between Subsystems . . . . .	2-254
Check reuse of variables within a Stateflow scope . . . . .	2-255
Check for Stateflow transition appearance . . . . .	2-256
Check usage of enumerated values . . . . .	2-258
Check Simulink signal appearance . . . . .	2-258
Check usage of non-compliant blocks . . . . .	2-260
<b>Japan MATLAB Automotive Advisory Board Checks . . . . .</b>	<b>2-261</b>
Japan MATLAB Automotive Advisory Board Checks . . . . .	2-264
Check for mixing basic blocks and subsystems . . . . .	2-266
Check Implement logic signals as Boolean data (vs. double) . . . . .	2-267
Check for Simulink diagrams using nonstandard display attributes . . . . .	2-268
Check font formatting . . . . .	2-270
Check positioning and configuration of ports . . . . .	2-272
Check whether block names appear below blocks . . . . .	2-274



Check the display attributes of block names .....	<b>2-275</b>
Check position of Trigger and Enable blocks .....	<b>2-277</b>
Check for nondefault block attributes .....	<b>2-278</b>
Check for unconnected ports and signal lines .....	<b>2-279</b>
Check usage of Switch blocks .....	<b>2-280</b>
Check usage of Relational Operator blocks .....	<b>2-281</b>
Check for indexing in blocks .....	<b>2-282</b>
Check usage of tunable parameters in blocks .....	<b>2-284</b>
Check usage of Discrete-Time Integrator block .....	<b>2-286</b>
Check usage of fixed-point data type with non-zero bias ...	<b>2-287</b>
Check input and output datatype for Switch blocks .....	<b>2-287</b>
Check signs of input signals in product blocks .....	<b>2-288</b>
Check transition orientations in flow charts .....	<b>2-289</b>
Check return value assignments in Stateflow graphical functions .....	<b>2-290</b>
Check for Strong Data Typing with Simulink I/O .....	<b>2-291</b>
Check Stateflow data objects with local scope .....	<b>2-292</b>
Check for MATLAB expressions in Stateflow charts .....	<b>2-294</b>
Check for pointers in Stateflow charts .....	<b>2-295</b>
Check for event broadcasts in Stateflow charts .....	<b>2-296</b>
Check for bitwise operations in Stateflow charts .....	<b>2-297</b>
Check usage of unary minus operations in Stateflow charts .....	<b>2-298</b>
Check input and output settings of MATLAB Functions .....	<b>2-299</b>
Check MATLAB code for global variables .....	<b>2-300</b>
Check usage of Memory and Unit Delay blocks .....	<b>2-301</b>
Check block orientation .....	<b>2-302</b>
Check usage of internal transitions in Stateflow states .....	<b>2-303</b>
Check usage of transition conditions in Stateflow transitions .....	<b>2-304</b>
Check usable characters for signal names and bus names ..	<b>2-305</b>
Check usable characters for parameter names .....	<b>2-306</b>
Check length of model file name .....	<b>2-307</b>
Check length of folder name at every level of model path ..	<b>2-308</b>
Check length of subsystem names .....	<b>2-308</b>
Check length of Inport and Outport names .....	<b>2-309</b>
Check length of signal and bus names .....	<b>2-310</b>
Check length of parameter names .....	<b>2-311</b>
Check length of block names .....	<b>2-312</b>
Check if blocks are shaded in the model .....	<b>2-313</b>
Check operator order of Product blocks .....	<b>2-313</b>
Check icon shape of Logical Operator blocks .....	<b>2-314</b>
Check if tunable block parameters are defined as named constants .....	<b>2-315</b>

Check default/else case in Switch Case blocks and If blocks .....	<b>2-316</b>
Check for parentheses in Fcn block expressions .....	<b>2-317</b>
Check undefined initial output for conditional subsystems .	<b>2-318</b>
Check usage of the Saturation blocks .....	<b>2-318</b>
Check type setting by data objects .....	<b>2-319</b>
Check prohibited comparison operation of logical type signals .....	<b>2-320</b>
Check usage of Stateflow comments .....	<b>2-321</b>
Check updates to variables used in state transition conditions .....	<b>2-322</b>
Check for unexpected backtracking in state transitions . . .	<b>2-323</b>
Check usage of parentheses in Stateflow transitions .....	<b>2-324</b>
Check condition actions and transition actions in Stateflow	<b>2-325</b>
Check starting point of internal transition in Stateflow . . . .	<b>2-326</b>
Check prohibited combination of state action and flow chart .....	<b>2-327</b>
Check usage of Lookup Tables .....	<b>2-328</b>
Check Signed Integer Division Rounding mode .....	<b>2-329</b>
Check usage of Merge block .....	<b>2-330</b>
Check for unused data in Stateflow Charts .....	<b>2-331</b>
Check usable number for first index .....	<b>2-332</b>
Check execution timing for default transition path .....	<b>2-333</b>
Check for parallel Stateflow state used for grouping .....	<b>2-333</b>
Check scope of data in parallel states .....	<b>2-334</b>
Check uniqueness of State names .....	<b>2-335</b>
Check usage of State names .....	<b>2-336</b>
Check uniqueness of Stateflow State and Data names .....	<b>2-337</b>
Check repetition of Action types .....	<b>2-338</b>
Check indentation of code in Stateflow states .....	<b>2-339</b>
Check placement of Label String in Transitions .....	<b>2-340</b>
Check duplication of Simulink data names .....	<b>2-341</b>
Comparing floating point types in Simulink .....	<b>2-342</b>
Check for avoiding algebraic loops between subsystems . . .	<b>2-343</b>
Check default transition placement in Stateflow charts . . . .	<b>2-343</b>
Check settings for data ports in Multiport Switch blocks . . .	<b>2-345</b>
Check for usage of text inside states .....	<b>2-346</b>
Check for unconnected objects in Stateflow Charts .....	<b>2-346</b>
Check Stateflow chart action language .....	<b>2-347</b>
Check usable characters for Stateflow data names .....	<b>2-348</b>
Check length of Stateflow data name .....	<b>2-349</b>
Check signal line labels .....	<b>2-350</b>
Check for propagated signal labels .....	<b>2-351</b>
Check trigger signal names .....	<b>2-352</b>

Check for consistency in model element names . . . . .	2-354
Check for sample time setting . . . . .	2-355
Check output data type of operation blocks . . . . .	2-356
Check usage of numeric literals in Stateflow . . . . .	2-357
Check usage of transitions to external states . . . . .	2-358
Check order of state action types . . . . .	2-358
Check if state action type 'exit' is used in the model . . . . .	2-359
Check position of comments in transition labels . . . . .	2-360
Check usage of unconditional transitions in flow charts . . . . .	2-361
Check for comments in unconditional transitions . . . . .	2-362
Check terminal junctions in Stateflow . . . . .	2-363
Check unused data in Simulink Model . . . . .	2-364
Check for use of C-style comment symbols . . . . .	2-365
Check for implicit type casting in Stateflow . . . . .	2-366
Check diagnostic settings for incorrect calculation results . . . . .	2-367
Check block names . . . . .	2-368
Check Model Description . . . . .	2-369
Usable characters for file names . . . . .	2-370
Usable characters for folder names . . . . .	2-371
Usable characters for Subsystem names . . . . .	2-373
Usable characters for Inport block and Outport block . . . . .	2-374
<b>MISRA C:2012 Checks . . . . .</b>	<b>2-377</b>
See Also . . . . .	2-377
Check usage of Assignment blocks . . . . .	2-377
Check for blocks not recommended for MISRA C:2012 . . . . .	2-379
Check for unsupported block names . . . . .	2-381
Check configuration parameters for MISRA C:2012 . . . . .	2-382
Check for equality and inequality operations on floating-point values . . . . .	2-386
Check for bitwise operations on signed integers . . . . .	2-387
Check for recursive function calls . . . . .	2-388
Check for switch case expressions without a default case . . . . .	2-389
Check for blocks not recommended for C/C++ production code deployment . . . . .	2-390
Check for missing error ports for AUTOSAR receiver interfaces . . . . .	2-391
Check for missing const qualifiers in model functions . . . . .	2-393
Check integer word length . . . . .	2-394
Check bus object names that are used as bus element names . . . . .	2-395

<b>Secure Coding Checks for CERT C, CWE, and ISO/IEC TS</b>	
<b>17961 Standards</b> . . . . .	<b>2-396</b>
See Also . . . . .	<b>2-396</b>
Check configuration parameters for secure coding standards	
. . . . .	<b>2-396</b>
Check for blocks not recommended for C/C++ production code	
deployment . . . . .	<b>2-399</b>
Check for blocks not recommended for secure coding standards	
. . . . .	<b>2-401</b>
Check usage of Assignment blocks . . . . .	<b>2-402</b>
Check for switch case expressions without a default case . .	<b>2-404</b>
Check for bitwise operations on signed integers . . . . .	<b>2-405</b>
Check for equality and inequality operations on floating-point	
values . . . . .	<b>2-406</b>
Check integer word length . . . . .	<b>2-407</b>
Detect Dead Logic . . . . .	<b>2-408</b>
Detect Integer Overflow . . . . .	<b>2-411</b>
Detect Division by Zero . . . . .	<b>2-413</b>
Detect Out Of Bound Array Access . . . . .	<b>2-414</b>
Detect Specified Minimum and Maximum Value Violations .	<b>2-416</b>
<b>Model Metrics</b> . . . . .	<b>2-419</b>
Model Metrics . . . . .	<b>2-419</b>
Size Metrics . . . . .	<b>2-419</b>
Architecture Metrics . . . . .	<b>2-420</b>
Compliance Metrics . . . . .	<b>2-421</b>
Readability Metrics . . . . .	<b>2-422</b>
Simulink block metric . . . . .	<b>2-422</b>
Subsystem metric . . . . .	<b>2-424</b>
Library link metric . . . . .	<b>2-425</b>
Effective lines of MATLAB code metric . . . . .	<b>2-426</b>
Stateflow chart objects metric . . . . .	<b>2-427</b>
Lines of code for Stateflow blocks metric . . . . .	<b>2-429</b>
Subsystem depth metric . . . . .	<b>2-430</b>
Input output metric . . . . .	<b>2-432</b>
Diagnostic warnings metric . . . . .	<b>2-433</b>
Explicit input output metric . . . . .	<b>2-434</b>
File metric . . . . .	<b>2-435</b>
Matlab Function metric . . . . .	<b>2-436</b>
Model file count . . . . .	<b>2-437</b>
Parameter metric . . . . .	<b>2-438</b>
Stateflow chart metric . . . . .	<b>2-439</b>
Cyclomatic complexity metric . . . . .	<b>2-440</b>
Clone content metric . . . . .	<b>2-442</b>

Clone detection metric .....	2-443
Library content metric .....	2-444
Nondescriptive block name metric .....	2-445
Data and structure layer separation metric .....	2-446
MATLAB code analyzer warnings .....	2-448
Model Advisor Check Compliance for High-Integrity Systems .....	2-449
Model Advisor Check Compliance for Modeling Standards for MAAB .....	2-450
Model Advisor Check Issues for High-Integrity Systems ...	2-451
Model Advisor Check Issues for MAAB Standards .....	2-452

## **Model Transformer Tasks**

### **3**

<b>Model Transformer Tasks</b> .....	3-2
Transformations .....	3-2
Transform the model to variant system .....	3-3
Eliminate Data Store Blocks .....	3-4
Transform table lookup into prelookup and interpolation ....	3-5

## **Clone Detection Tasks**

### **4**

<b>Clone Detection Checks</b> .....	4-2
Exclude subsystems and referenced models from clone detection .....	4-2

## **Simulink Check Apps**

### **5**



# Functions — Alphabetical List

---

# Advisor.authoring.CustomCheck.actionCallback

**Class:** Advisor.authoring.CustomCheck

**Package:** Advisor.authoring

Register action callback for model configuration check

## Syntax

Advisor.authoring.CustomCheck.actionCallback(task)

## Description

Advisor.authoring.CustomCheck.actionCallback(task) is used as the action callback function when registering custom checks that use an XML data file to specify check behavior.

## Examples

This `sl_customization.m` file registers the action callback for configuration parameter checks with fix actions.

```
function defineModelAdvisorChecks

    rec = ModelAdvisor.Check('com.mathworks.Check1');
    rec.Title = 'Test: Check1';
    rec.setCallbackFcn(@(system)(Advisor.authoring.CustomCheck.checkCallback(system)), ...
        'None', 'StyleOne');
    rec.TitleTips = 'Example check for check authoring infrastructure.';

    % --- data file input parameters
    rec.setInputParametersLayoutGrid([1 1]);
    inputParam1 = ModelAdvisor.InputParameter;
    inputParam1.Name = 'Data File';
    inputParam1.Value = 'Check1.xml';
    inputParam1.Type = 'String';
    inputParam1.Description = 'Name or full path of XML data file.';
    inputParam1.setRowSpan([1 1]);
```



```
inputParam1.setColSpan([1 1]);  
rec.setInputParameters({inputParam1});  
  
% -- set fix operation  
act = ModelAdvisor.Action;  
act.setCallbackFcn(@(task)(Advisor.authoring.CustomCheck.actionCallback(task)));  
act.Name = 'Modify Settings';  
act.Description = 'Modify model configuration settings.';  
rec.setAction(act);  
  
mdladvRoot = ModelAdvisor.Root;  
mdladvRoot.register(rec);  
end
```

## See Also

[Advisor.authoring.CustomCheck.checkCallback](#) |  
[Advisor.authoring.DataFile](#) |  
[Advisor.authoring.generateConfigurationParameterDataFile](#)

## Topics

“Create Check for Model Configuration Parameters”

## addCheck

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Add check to folder

## Syntax

```
addCheck(fg_obj, check_ID)
```

## Description

`addCheck(fg_obj, check_ID)` adds checks, identified by `check_ID`, to the folder specified by `fg_obj`, which is an instantiation of the `ModelAdvisor.FactoryGroup` class.

## Examples

Add four checks to rec:

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
.
.
addCheck(rec, 'com.mathworks.sample.Check0');
addCheck(rec, 'com.mathworks.sample.Check1');
addCheck(rec, 'com.mathworks.sample.Check2');
addCheck(rec, 'com.mathworks.sample.Check3');
```

# addGroup

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Add subfolder to folder

## Syntax

```
addGroup(group_obj, child_obj)
```

## Description

`addGroup(group_obj, child_obj)` adds a new subfolder, identified by `child_obj`, to the folder specified by `group_obj`, which is an instantiation of the `ModelAdvisor.Group` class.

## Examples

Add three checks to rec:

```
group_obj = ModelAdvisor.Group('com.mathworks.sample.group');  
.  
.  
.  
addGroup(group_obj, 'com.mathworks.sample.subgroup1');  
addGroup(group_obj, 'com.mathworks.sample.subgroup2');  
addGroup(group_obj, 'com.mathworks.sample.subgroup3');
```

To add `ModelAdvisor.Task` objects to a group using `addGroup`:

```
mdladvRoot = ModelAdvisor.Root();  
  
% MAT1, MAT2, and MAT3 are registered ModelAdvisor.Task objects  
% Create the group 'My Group'  
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
MAG.DisplayName='My Group';
```

```
% Add the first task to the 'My Group' folder
MAG.addTask(MAT1);

% Create a subfolder 'Folder1'
MAGSUB1 = ModelAdvisor.Group('com.mathworks.sample.Folder1');
MAGSUB1.DisplayName='Folder1';

% Add the second task to Folder1
MAGSUB1.addTask(MAT2);

% Create a subfolder 'Folder2'
MAGSUB2 = ModelAdvisor.Group('com.mathworks.sample.Folder2');
MAGSUB2.DisplayName='Folder2';

% Add the third task to Folder2
MAGSUB2.addTask(MAT3);

% Register the two subfolders. This must be done before calling addGroup
mdladvRoot.register(MAGSUB1);
mdladvRoot.register(MAGSUB2);

% Invoke addGroup to place the subfolders under 'My Group'
MAG.addGroup(MAGSUB1);
MAG.addGroup(MAGSUB2);

mdladvRoot.publish(MAG); % publish under Root
```

# addItem

**Class:** ModelAdvisor.List

**Package:** ModelAdvisor

Add item to list

## Syntax

`addItem(element)`

## Description

`addItem(element)` adds items to the list created by the `ModelAdvisor.List` constructor.

## Input Arguments

<i>element</i>	Specifies an element to be added to a list in one of the following: <ul style="list-style-type: none"><li>• Element</li><li>• Cell array of elements. When you add a cell array to a list, they form different rows in the list.</li><li>• Character vector</li></ul>
----------------	---

## Examples

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass', 'bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass', 'bold'}));
```

## **See Also**

*“Model Advisor Customization”*

## **Topics**

*“Create Model Advisor Checks”*

# addItem

**Class:** ModelAdvisor.Paragraph

**Package:** ModelAdvisor

Add item to paragraph

## Syntax

```
addItem(text, element)
```

## Description

addItem(text, element) adds an element to text. element is one of the following:

- Character vector
- Element
- Cell array of elements

## Examples

Add two lines of text:

```
result = ModelAdvisor.Paragraph;  
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## addProcedure

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Add procedure to folder

## Syntax

```
addProcedure(group_obj, procedure_obj)
```

## Description

`addProcedure(group_obj, procedure_obj)` adds a procedure, specified by `procedure_obj`, to the folder `group_obj`. `group_obj` is an instantiation of the `ModelAdvisor.Group` class.

## Examples

Add three procedures to MAG.

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
  
MAP1=ModelAdvisor.Procedure('com.mathworks.sample.procedure1');  
MAP2=ModelAdvisor.Procedure('com.mathworks.sample.procedure2');  
MAP3=ModelAdvisor.Procedure('com.mathworks.sample.procedure3');  
  
addProcedure(MAG, MAP1);  
addProcedure(MAG, MAP2);  
addProcedure(MAG, MAP3);
```



# addProcedure

**Class:** ModelAdvisor.Procedure

**Package:** ModelAdvisor

Add subprocedure to procedure

## Syntax

```
addProcedure(procedure1_obj, procedure2_obj)
```

## Description

`addProcedure(procedure1_obj, procedure2_obj)` adds a procedure, specified by `procedure2_obj`, to the procedure `procedure1_obj`. `procedure2_obj` and `procedure1_obj` are instantiations of the `ModelAdvisor.Procedure` class.

## Examples

Add three procedures to MAP.

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');  
  
MAP1=ModelAdvisor.Procedure('com.mathworks.sample.procedure1');  
MAP2=ModelAdvisor.Procedure('com.mathworks.sample.procedure2');  
MAP3=ModelAdvisor.Procedure('com.mathworks.sample.procedure3');  
  
addProcedure(MAP, MAP1);  
addProcedure(MAP, MAP2);  
addProcedure(MAP, MAP3);
```

## addRow

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add row to table

## Syntax

```
addRow(ft_obj, {item1, item2, ..., itemn})
```

## Description

`addRow(ft_obj, {item1, item2, ..., itemn})` is an optional method that adds a row to the end of a table in the result. *ft\_obj* is a handle to the template object previously created. {*item1*, *item2*, ..., *itemn*} is a cell array of character vectors and objects to add to the table. The order of the items in the array determines which column the item is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

---

**Note** Before adding rows to a table, you must specify column titles using the `setColTitle` method.

---

## Examples

Find all of the blocks in the model and create a table of the blocks:

```
% Create FormatTemplate object, specify table format
ft = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information to the table
setTableTitle(ft, {'Blocks in Model'});
setColTitles(ft, {'Index', 'Block Name'});
% Find all the blocks in the system and add them to a table.
allBlocks = find_system(system);
for inx = 2 : length(allBlocks)
    % Add information to the table
```

```
    addRow(ft, {inx-1,allBlocks(inx)});  
end
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## addTask

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Add task to folder

## Syntax

```
addTask(group_obj, task_obj)
```

## Description

`addTask(group_obj, task_obj)` adds a task, specified by `task_obj`, to the folder `group_obj`. `group_obj` is an instantiation of the `ModelAdvisor.Group` class.

## Examples

Add three tasks to MAG.

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
addTask(MAG, MAT8);  
addTask(MAG, MAT1);  
addTask(MAG, MAT2);  
addTask(MAG, MAT3);
```

# addTask

**Class:** ModelAdvisor.Procedure

**Package:** ModelAdvisor

Add task to procedure

## Syntax

```
addTask(procedure_obj, task_obj)
```

## Description

`addTask(procedure_obj, task_obj)` adds a task, specified by `task_obj`, to `procedure_obj`. `procedure_obj` is an instantiation of the `ModelAdvisor.Procedure` class.

## Examples

Add three tasks to MAP.

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');
```

```
MAT1=ModelAdvisor.Task('com.mathworks.sample.task1');
```

```
MAT2=ModelAdvisor.Task('com.mathworks.sample.task2');
```

```
MAT3=ModelAdvisor.Task('com.mathworks.sample.task3');
```

```
addTask(MAP, MAT1);
```

```
addTask(MAP, MAT2);
```

```
addTask(MAP, MAT3);
```

## Advisor.Application class

**Package:** Advisor

Run Model Advisor across model hierarchy

### Description

Use instances of `Advisor.Application` to run Model Advisor checks across a model hierarchy. You can use `Advisor.Application` to:

- Run checks on referenced models.
- Select model components for Model Advisor analysis.
- Select checks to run during Model Advisor analysis.

Consider using `Advisor.Application` if you have a large model with subsystems and model references. `Advisor.Application` does not run checks on library models. If you want to run checks on multiple independent models that are not in a model reference hierarchy or you want to leverage parallel processing, use `ModelAdvisor.run` to run Model Advisor checks on your model.

The `Advisor.Application` methods use the following definitions:

- Model component — Model in the system hierarchy. Models that the root model references and that `setAnalysisroot` specifies are model components.
- Check instance — Instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. Each check instance has an instance ID. When you change the Model Advisor configuration, the instance ID can change.

### Construction

To create an `Advisor.Application` object, use `Advisor.Manager.createApplication`.

## Properties

### **AnalysisRoot — Name of root model in the model hierarchy to analyze**

character vector

Name of root model in the model hierarchy to analyze, as specified by the `Advisor.Application.setAnalysisRoot` method. This property is read only.

### **ID — Unique identifier**

character vector

Unique identifier for the `Advisor.Application` object. This property is read only.

### **UseTempDir — Run analysis in a temporary working folder**

false (default) | true

Run analysis in a temporary working folder. Specified by the `Advisor.Manager.createApplication` method. This property is read only.

Data Types: logical

### **AnalyzeVariants — Run analysis on active and inactive variants**

false (default) | true

Run analysis on active and inactive variants based on predefined configurations in the the Variant Manager. For each configuration, produce a Model Advisor report. This property is read/write.

Data Types: logical

## Methods

<code>delete</code>	Delete <code>Advisor.Application</code> object
<code>deselectCheckInstances</code>	Clear check instances from Model Advisor analysis
<code>deselectComponents</code>	Clear model components from Model Advisor analysis
<code>generateReport</code>	Generate report for Model Advisor analysis
<code>getCheckInstanceIDs</code>	Obtain check instance IDs
<code>getResults</code>	Access Model Advisor analysis results
<code>loadConfiguration</code>	Load Model Advisor configuration
<code>run</code>	Run Model Advisor analysis on model components
<code>selectCheckInstances</code>	Select check instances to use in Model Advisor analysis
<code>selectComponents</code>	Select model components for Model Advisor analysis
<code>setAnalysisRoot</code>	Specify model hierarchy for Model Advisor analysis

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see [Copying Objects \(MATLAB\)](#).

## Examples

### Run Model Advisor Checks on Referenced Model

This example shows how to run a check on model `sldemo_mdhref_counter` referenced from `sldemo_mdhref_basic`.

In the Command Window, open model `sldemo_mdhref_basic` and referenced model `sldemo_mdhref_counter`.

```
open_system('sldemo_mdhref_basic');  
open_system('sldemo_mdhref_counter');
```

Save a copy of the models to a work folder, renaming them to `mdhref_basic` and `mdhref_counter`.



```
save_system('sldemo_mdref_basic','mdlref_basic');
save_system('sldemo_mdref_counter','mdlref_counter');
```

In `mdlref_basic`, change model reference from `sldemo_mdref_counter` to `mdlref_counter`. Save `mdlref_basic`.

```
set_param('mdlref_basic/CounterA','modelName','mdlref_counter');
set_param('mdlref_basic/CounterB','modelName','mdlref_counter');
set_param('mdlref_basic/CounterC','modelName','mdlref_counter');
save_system('mdlref_basic');
```

Set root model to `mdlref_basic`.

```
RootModel='mdlref_basic';
```

Create an Application object.

```
app = Advisor.Manager.createApplication();
```

Set root analysis.

```
setAnalysisRoot(app,'Root',RootModel);
```

Clear all check instances from Model Advisor analysis.

```
deselectCheckInstances(app);
```

Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
selectCheckInstances(app,'IDs',checkinstanceID);
```

Run Model Advisor analysis.

```
run(app);
```

Get analysis results.

```
getResults(app);
```

Generate and view the Model Advisor report. The Model Advisor runs the check on both `mdlref_basic` and `mdlref_counter`.

```
report = generateReport(app);
web(report)
```

Close the models.

```
close_system('mdlref_basic');  
close_system('mdlref_counter');
```

## Run Model Advisor Checks on a Subsystem

This example shows how to run a check on subsystem CounterA referenced from sldemo\_mdlref\_basic.

In the Command Window, open model sldemo\_mdlref\_basic.

```
open_system('sldemo_mdlref_basic');
```

Set root model to sldemo\_mdlref\_basic.

```
RootModel='sldemo_mdlref_basic';
```

Create an Application object.

```
app = Advisor.Manager.createApplication();
```

Set root analysis to subsystem sldemo\_mdlref\_basic/CounterA.

```
setAnalysisRoot(app, 'Root', 'sldemo_mdlref_basic/CounterA', 'RootType', 'Subsystem');
```

Clear all check instances from Model Advisor analysis.

```
deselectCheckInstances(app);
```

Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app, 'mathworks.design.UnconnectedLinesPorts');  
checkinstanceID = instanceID(1);  
selectCheckInstances(app, 'IDs', checkinstanceID);
```

Run Model Advisor analysis.

```
run(app);
```

Get analysis results.

```
getResults(app);
```

Generate and view the Model Advisor report. The Model Advisor runs the check on subsystem `sldemo_mdref_basic/CounterA`.

```
report = generateReport(app);  
web(report)
```

Close the model.

```
close_system('sldemo_mdref_basic');
```

## See Also

### Topics

[Class Attributes \(MATLAB\)](#)

[Property Attributes \(MATLAB\)](#)

**Introduced in R2015b**

## ModelAdvisor.CheckResult

Access Model Advisor check results

### Description

View Model Advisor check results obtained by using the `run` method.

### Creation

To access the properties of a `Check.Result` object, access the `CheckResultsObj`s property of a `System.Result` object.

### Properties

**system — Model or subsystem that Model Advisor analyzes**

character vector

Model or subsystem that the Model Advisor analyzes.

Data Types: char

**status — Model Advisor check status**

'Not Run' | 'Pass' | 'Fail' | 'Warn'

Status of Model Advisor check

Data Types: char

**checkID — Model Advisor check ID**

character vector

Model Advisor check ID

Data Types: char

**checkName — Model Advisor check name**

character vector

Name of Model Advisor check

Data Types: char

## Examples

### Run Model Advisor Checks on Referenced Model

This example shows how to run a check on model `sldemo_mdhref_counter` referenced from `sldemo_mdhref_basic`.

In the Command Window, open model `sldemo_mdhref_basic` and referenced model `sldemo_mdhref_counter`.

```
open_system('sldemo_mdhref_basic');  
open_system('sldemo_mdhref_counter');
```

Save a copy of the models to a work folder, renaming them to `mdhref_basic` and `mdhref_counter`.

```
save_system('sldemo_mdhref_basic','mdhref_basic');  
save_system('sldemo_mdhref_counter','mdhref_counter');
```

In `mdhref_basic`, change model reference from `sldemo_mdhref_counter` to `mdhref_counter`. Save `mdhref_basic`.

```
set_param('mdhref_basic/CounterA','ModelName','mdhref_counter');  
set_param('mdhref_basic/CounterB','ModelName','mdhref_counter');  
set_param('mdhref_basic/CounterC','ModelName','mdhref_counter');  
save_system('mdhref_basic');
```

Set root model to `mdhref_basic`.

```
RootModel='mdhref_basic';
```

Create an Application object.

```
app = Advisor.Manager.createApplication();
```

Set root analysis.

```
setAnalysisRoot(app,'Root',RootModel);
```

Clear check instances from Model Advisor analysis.

```
deselectCheckInstances(app);
```

Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app, 'mathworks.design.UnconnectedLinesPorts');  
checkinstanceID = instanceID(1);  
selectCheckInstances(app, 'IDs', checkinstanceID);
```

Run Model Advisor analysis.

```
run(app);
```

Get analysis results and view the properties of the `ModelAdvisor.SystemResult` and `ModelAdvisor.CheckResult` objects.

```
getResults(app);
```

Close the models.

```
close_system('mdlref_basic');  
close_system('mdlref_counter');
```

## See Also

`Advisor.Manager`

**Introduced in R2015b**

# ModelAdvisor.SystemResult

Access system-level Model Advisor results

## Description

View Model Advisor results obtained by using the `run` method.

## Creation

To access the properties of a `System.Result` object, apply the `getResults` method to an `Advisor.Application` object.

## Properties

### **system** — Model or subsystem that Model Advisor analyzes

character vector

Model or subsystem that the Model Advisor analyzes.

Data Types: `char`

### **Type** — Model component

'Model' | 'Subsystem'

Type of model component that Model Advisor analyzes.

Data Types: `char`

### **numPass** — Number of Model Advisor checks that pass

double

Obtain the number of Model Advisor checks that pass after running the Model Advisor using the `run` method.

Data Types: `double`

## **numFail — Number of Model Advisor checks that fail**

double

Obtain the number of Model Advisor checks that fail after running the Model Advisor using the `run` method.

Data Types: double

## **numNotRun — Number of Model Advisor checks that do not run**

double

Obtain the number of Model Advisor checks that do not run after running the Model Advisor using the `run` method.

Data Types: double

## **numWarn — Number of Model Advisor checks that warn**

double

Obtain the number of Model Advisor checks that warn after running the Model Advisor using the `run` method.

Number of Model Advisor checks that warn.

Data Types: double

## **CheckResultObjs — Model Advisor Check results**

cell array of `ModelAdvisor.CheckResult` objects

Cell array containing a `ModelAdvisor.CheckResult` object for each Model Advisor check. Access these object properties to obtain Model Advisor check results.

Data Types: double

# Examples

## **Run Model Advisor Checks on Referenced Model**

This example shows how to run a check on model `sldemo_mdref_counter` referenced from `sldemo_mdref_basic`.



In the Command Window, open model `sldemo_mdref_basic` and referenced model `sldemo_mdref_counter`.

```
open_system('sldemo_mdref_basic');
open_system('sldemo_mdref_counter');
```

Save a copy of the models to a work folder, renaming them to `mdlref_basic` and `mdlref_counter`.

```
save_system('sldemo_mdref_basic','mdlref_basic');
save_system('sldemo_mdref_counter','mdlref_counter');
```

In `mdlref_basic`, change model reference from `sldemo_mdref_counter` to `mdlref_counter`. Save `mdlref_basic`.

```
set_param('mdlref_basic/CounterA','ModelName','mdlref_counter');
set_param('mdlref_basic/CounterB','ModelName','mdlref_counter');
set_param('mdlref_basic/CounterC','ModelName','mdlref_counter');
save_system('mdlref_basic');
```

Set root model to `mdlref_basic`.

```
RootModel='mdlref_basic';
```

Create an Application object.

```
app = Advisor.Manager.createApplication();
```

Set root analysis.

```
setAnalysisRoot(app,'Root',RootModel);
```

Clear check instances from Model Advisor analysis.

```
deselectCheckInstances(app);
```

Select check **Identify unconnected lines, input ports, and output ports** using check instance ID.

```
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');
checkinstanceID = instanceID(1);
selectCheckInstances(app,'IDs',checkinstanceID);
```

Run Model Advisor analysis.

```
run(app);
```

Get analysis results and view the properties of the `ModelAdvisor.SystemResult` and `ModelAdvisor.CheckResult` objects.

```
Results=getResults(app);
```

Generate and view the Model Advisor report. The Model Advisor runs the check on both `mdlref_basic` and `mdlref_counter`.

```
report = generateReport(app);  
web(report)
```

Close the models.

```
close_system('mdlref_basic');  
close_system('mdlref_counter');
```

### See Also

`Advisor.Manager`

**Introduced in R2015b**

# Advisor.authoring.generateConfigurationParameterDataFile

**Package:** Advisor.authoring

Generate XML data file for custom configuration parameter check

## Syntax

```
Advisor.authoring.generateConfigurationParameterDataFile(dataFile,  
source)
```

```
Advisor.authoring.generateConfigurationParameterDataFile(dataFile,  
source,Name,Value)
```

## Description

`Advisor.authoring.generateConfigurationParameterDataFile(dataFile, source)` generates an XML data file named `dataFile` specifying the configuration parameters for `source`. The data file uses tagging to specify the configuration parameter settings you want. When you create a check for configuration parameters, you use the data file. Each model configuration parameter specified in the data file is a subcheck.

`Advisor.authoring.generateConfigurationParameterDataFile(dataFile, source,Name,Value)` generates an XML data file named `dataFile` specifying the configuration parameters for `source`. It also specifies additional options by one or more optional `Name,Value` arguments. The data file uses tagging to specify the configuration parameter settings you want. When you create a check for configuration parameters, you use the data file. Each model configuration parameter specified in the data file is a subcheck.

## Examples

## Create data file for configuration parameter check

Create a data file with all the configuration parameters. You use the data file to create a configuration parameter.

```
model = 'vdp';  
dataFile = 'myDataFile.xml';  
Advisor.authoring.generateConfigurationParameterDataFile( ...  
    dataFile, model);
```

Data file `myDataFile.xml` has tagging specifying subcheck information for each configuration parameter. `myDataFile.xml` specifies the configuration parameters settings you want. The following specifies XML tagging for configuration parameter `AbsTol`. If the configuration parameter is set to `1e-6`, the configuration parameter subcheck specified in `myDataFile.xml` passes.

```
<!-- Absolute tolerance: (AbsTol)-->  
  <PositiveModelParameterConstraint>  
    <parameter>AbsTol</parameter>  
    <value>1e-6</value>  
  </PositiveModelParameterConstraint>
```

## Create data file for Solver pane configuration parameter check with fix action

Create a data file with configuration parameters for the **Solver** pane. You use the data file to create a **Solver** pane configuration parameter check with fix actions.

```
model = 'vdp';  
dataFile = 'myDataFile.xml';  
Advisor.authoring.generateConfigurationParameterDataFile( ...  
    dataFile, model, 'Pane', 'Solver', 'FixValues', true);
```

Data file `myDataFile.xml` has tagging specifying subcheck information for each configuration parameter. `myDataFile.xml` specifies the configuration parameters settings that you want. The following specifies XML tagging for configuration parameter `AbsTol`. If the configuration parameter is set to `1e-6`, the configuration parameter subcheck specified in `myDataFile.xml` passes. If the subcheck does not pass, the check fix action modifies the configuration parameter to `1e-6`.

```
<!-- Absolute tolerance: (AbsTol)-->  
  <PositiveModelParameterConstraint>  
    <parameter>AbsTol</parameter>
```

```
<value>1e-6</value>  
<fixvalue>1e-6</fixvalue>  
</PositiveModelParameterConstraint>
```

## Input Arguments

### **dataFile** — Name of data file to create

character vector

Name of XML data file to create, specified as a character vector.

Example: 'myDataFile.xml'

### **source** — Name of model or configuration set

character vector | Simulink.ConfigSet

Name of model or Simulink.ConfigSet object used to specify configuration parameters

Example: 'vdp'

## Name-Value Pair Arguments

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

Example: 'Pane', 'Solver', 'FixValues', true specifies a dataFile with Solver pane configuration parameters and fix tagging.

### **Pane** — Limit the configuration parameters in the dataFile

Solver | Data Import/Export | Optimization | Diagnostics | Hardware Implementation | Model Referencing | Code Generation

Option to limit the configuration parameters in the data file to the pane specified as the comma-separated pair of 'Pane' and one of the following:

- Solver
- Data Import/Export

- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Code Generation

Example: 'Pane', 'Solver' limits the dataFile to configuration parameters on the Solver pane.

Data Types: char

### **FixValues — Create fix tagging in the dataFile**

false | true

Setting FixValues to true provides the dataFile with fix tagging. When you generate a custom configuration parameter check using a dataFile with fix tagging, each configuration parameter subcheck has a fix action. Specified as the comma-separated pair of 'FixValues' and either true or false.

Example: 'FixValues, true specifies fix tagging in the dataFile.

Data Types: logical

## **See Also**

### **Topics**

“Create Check for Model Configuration Parameters”

“Data File for Configuration Parameter Check”

**Introduced in R2014a**

# Advisor.authoring.generateBlockConstraintsDataFile

**Package:** Advisor.authoring

Generate XML data file for custom check for block constraints

## Syntax

```
Advisor.authoring.generateBlockConstraintsDataFile(dataFile,  
'constraints',constraintslist)
```

## Description

Advisor.authoring.generateBlockConstraintsDataFile(dataFile, 'constraints',constraintslist) generates an XML data file named dataFile. This data file specifies the constraints that a custom check contains. The data file uses tagging to specify the constraint information. When you create a custom check, you use the data file.

Define constraint objects in the base workspace and then pass these objects as inputs to this function. These constraints may be root constraints and prerequisites to root constraints. You can also define a composite constraint. If you specify multiple root constraints and no composite constraint, Simulink implements a composite constraint with a CompositeOperator of and.

## Examples

### Create Data File for Custom Check for Block Constraints

Create a custom check for this JMAAB 4.0 check: **jc\_0632: Default case port in Multiport Switch block**. For Multiport Switch blocks, the check contains a constraint that checks that the **Data port order** parameter setting is Specify indices. If the parameter has this setting, there are constraints that check that the **Data port for**

**default case** parameter setting is Additional data port and the **Diagnostic for default case** setting is None.

Create three `PositiveBlockParameter` constraint objects.

```
c1 = Advisor.authoring.PositiveBlockParameterConstraint();
c1.ID = 'ID_A2';
c1.BlockType = 'MultiPortSwitch';
c1.ParameterName = 'DataPortOrder';
c1.SupportedParameterValues = {'Specify indices'};
c1.ValueOperator = 'eq';

c2 = Advisor.authoring.PositiveBlockParameterConstraint();
c2.ID = 'ID_A3';
c2.BlockType = 'MultiPortSwitch';
c2.ParameterName = 'DataPortForDefault';
c2.SupportedParameterValues = {'Additional data port'};
c2.ValueOperator = 'eq';

c3 = Advisor.authoring.PositiveBlockParameterConstraint();
c3.ID = 'ID_A4';
c3.BlockType = 'MultiPortSwitch';
c3.ParameterName = 'DiagnosticForDefault';
c3.SupportedParameterValues = {'None'};
c3.ValueOperator = 'eq';
```

Use the `addPreRequisiteConstraintID` method to make `c1` a prerequisite to checking constraints `c2` and `c3`.

```
c2.addPreRequisiteConstraintID('ID_A2');
c3.addPreRequisiteConstraintID('ID_A2');
```

Create a composite constraint that specifies that if a Multiport Switch block does not meet constraints `c2` and `c3`, the block is in violation of this check.

```
cc = Advisor.authoring.CompositeConstraint();
cc.addConstraintID('ID_A3');
cc.addConstraintID('ID_A4');
cc.CompositeOperator = 'and';
```

Create a data file that contains the constraints.

```
dataFile = 'myDataFile.xml';
Advisor.authoring.generateBlockConstraintsDataFile( ...
    dataFile, 'constraints', {c1,c2,c3,cc});
```

Data file `myDataFile.xml` has tagging specifying the constraint information for the custom check.

```
<?xml version="1.0" encoding="utf-8"?>
<customcheck>
```



```

<checkdata>
  <PositiveBlockParameterConstraint BlockType="MultiPortSwitch" id="ID_A2">
    <parameter type="string">DataPortOrder</parameter>
    <value>Specify indices</value>
    <operator>eq</operator>
  </PositiveBlockParameterConstraint>
  <PositiveBlockParameterConstraint BlockType="MultiPortSwitch" id="ID_A3">
    <parameter type="string">DataPortForDefault</parameter>
    <value>Additional data port</value>
    <operator>eq</operator>
    <dependson>ID_A2</dependson>
  </PositiveBlockParameterConstraint>
  <PositiveBlockParameterConstraint BlockType="MultiPortSwitch" id="ID_A4">
    <parameter type="string">DiagnosticForDefault</parameter>
    <value>None</value>
    <operator>eq</operator>
    <dependson>ID_A2</dependson>
  </PositiveBlockParameterConstraint>
  <CompositeConstraint>
    <ID>ID_A3</ID>
    <ID>ID_A4</ID>
    <operator>and</operator>
  </CompositeConstraint>
</checkdata>
</customcheck>

```

---

**Note** For model configuration parameter constraints, use the `Advisor.authoring.generateBlockConstraintsDataFile` method only when specifying model configuration parameter constraints as prerequisites to block constraints or as part of a composite constraint consisting of both block and model configuration parameter constraints. For other cases, use the `Advisor.authoring.generateConfigurationParameterDatafile` method.

---

## Input Arguments

### **dataFile** — Name of data file to create

character vector

Name of XML data file to create, specified as a character vector.

Example: 'myDataFile.xml'

### **constraintslist** — cell array of constraint objects

cell array of objects

Use these classes to create constraint objects:

- `Advisor.authoring.PositiveBlockParameterConstraint`
- `Advisor.authoring.NegativeBlockParameterConstraint`
- `Advisor.authoring.PositiveModelParameterConstraint`
- `Advisor.authoring.NegativeModelParameterConstraint`
- `Advisor.authoring.PositiveBlockTypeConstraint`
- `Advisor.authoring.NegativeBlockTypeConstraint`
- `Advisor.authoring.CompositeConstraint`

Example: `{c1,c2,c3}`

## See Also

`Advisor.authoring.createBlockConstraintCheck` | `CompositeConstraint` | `NegativeBlockParameterConstraint` | `NegativeBlockTypeConstraint` | `PositiveBlockParameterConstraint` | `PositiveBlockTypeConstraint`

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

# Advisor.authoring.createBlockConstraintCheck

**Package:** Advisor.authoring

Create Model Advisor check for registering block constraints

## Syntax

```
check_obj = Advisor.authoring.createBlockConstraintCheck(check_ID)
```

## Description

`check_obj = Advisor.authoring.createBlockConstraintCheck(check_ID)` creates a `ModelAdvisor.check` object, `check_obj`, and assigns it a unique identifier, `check_ID`. Specify the block constraints data file as an input parameter to this object. Use the `Advisor.authoring.generateBlockConstraintsDataFile` function to create the block constraints data file.

## Examples

### Create Model Advisor Checks from Constraint

This code shows how to specify and register a Model Advisor constraint check in the `sl_customization` file. Just below the `%check` comment, the `Advisor.authoring.createBlockConstraintCheck` function creates the `ModelAdvisor.check` object `rec`. The `inputParam1.value` is the name of the data file that contains the block constraints. In this example, that data file is `myDataFile.xml`. For an example of how to create this data file, see `Advisor.authoring.generateBlockConstraintsDataFile`.

```
function sl_customization(cm)
% register custom checks.
cm.addModelAdvisorCheckFcn(@defineModelAdvisorChecks);
```

```
% register items to factory group.
cm.addModelAdvisorTaskFcn(@defineModelAdvisorGroups);

% defineModelAdvisorChecks
function defineModelAdvisorChecks

% check
rec = Advisor.authoring.createBlockConstraintCheck('com.mathworks.sample.Check1');
rec.Title = 'Example: Check block parameter constraints';
rec.setCallbackFcn(@(system)(Advisor.authoring.CustomCheck.checkCallback...
    (system)), 'None', 'Style0ne');
rec.TitleTips = 'Example check block parameter constraints';

% --- data file input parameters
rec.setInputParametersLayoutGrid([1 1]);
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Data File';
inputParam1.Value = 'myDataFile.xml';
inputParam1.Type = 'String';
inputParam1.Description = 'Name or full path of XML data file.';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
rec.setInputParameters({inputParam1});
rec.SupportExclusion = false;
rec.SupportLibrary = true;

mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);

% defineModelAdvisorGroups
function defineModelAdvisorGroups
mdladvRoot = ModelAdvisor.Root;

% --- sample factory group 1
rec = ModelAdvisor.FactoryGroup('com.mathworks.Test.factoryGroup');
rec.DisplayName='Example: My Group';
rec.addCheck('com.mathworks.sample.Check1');

mdladvRoot.publish(rec);
```

## Input Arguments

### **check\_ID** — Name of Model Advisor check

character vector

Name of model advisor check, specified as a character vector

Example: 'com.mathworks.sample.Check1'

## Output Arguments

### **check\_obj** — Model Advisor check object

character vector

New ModelAdvisor.check object with default property values.

---

**Note** The ModelAdvisor.Check object that you create using the Advisor.authoring.createBlockConstraintCheck function does not support setting exclusions.

---

## See Also

Advisor.authoring.generateBlockConstraintsDataFile |  
CompositeConstraint | NegativeBlockParameterConstraint |  
NegativeBlockTypeConstraint | PositiveBlockParameterConstraint |  
PositiveBlockTypeConstraint

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

## Advisor.authoring.CustomCheck class

**Package:** `Advisor.authoring`

Define custom check

### Description

Instances of the `Advisor.authoring.CustomCheck` class provide a container for static methods used as callback functions when defining a configuration parameter check. The configuration parameter check is defined in an XML data file.

### Methods

<code>actionCallback</code>	Register action callback for model configuration check
<code>checkCallback</code>	Register check callback for model configuration check

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB® Programming Fundamentals documentation.

### See Also

`Advisor.authoring.DataFile` |  
`Advisor.authoring.generateConfigurationParameterDataFile`

### Topics

“Create Check for Model Configuration Parameters”

# Advisor.authoring.DataFile class

**Package:** Advisor.authoring

Interact with data file for model configuration checks

## Description

The `Advisor.authoring.DataFile` class provides a container for a static method used when interacting with the data file for configuration parameter checks.

## Methods

`validate` Validate XML data file used for model configuration check

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## See Also

`Advisor.authoring.CustomCheck` |  
`Advisor.authoring.generateConfigurationParameterDataFile`

## Topics

“Create Check for Model Configuration Parameters”

## Advisor.Manager class

**Package:** Advisor

Manage applications

### Description

The `Advisor.Manager` class defines application objects.

### Methods

<code>createApplication</code>	Create <code>Advisor.Application</code> object
<code>getApplication</code>	Return handle to <code>Advisor.Application</code> object
<code>refresh_customizations</code>	Refresh Model Advisor check information cache

### Copy Semantics

Handle. To learn how handle classes affect copy operations, see [Copying Objects \(MATLAB\)](#).

### See Also

#### Topics

[Class Attributes \(MATLAB\)](#)

[Property Attributes \(MATLAB\)](#)

**Introduced in R2015b**



# Advisor.authoring.CustomCheck.checkCallback

**Class:** Advisor.authoring.CustomCheck

**Package:** Advisor.authoring

Register check callback for model configuration check

## Syntax

Advisor.authoring.CustomCheck.checkCallback(system, CheckObj)

## Description

Advisor.authoring.CustomCheck.checkCallback(system, CheckObj) is used as the check callback function when registering custom checks that use an XML data file to specify check behavior.

## Examples

In the following example, the `sl_customization.m` file registers a configuration parameter check using `Advisor.authoring.CustomCheck.checkCallback(system)`.

```
function defineModelAdvisorChecks

    rec = ModelAdvisor.Check('com.mathworks.Check1');
    rec.Title = 'Test: Check1';
    rec.setCallbackFcn(@system)(Advisor.authoring.CustomCheck.checkCallback(system)), ...
        'None', 'StyleOne');
    rec.TitleTips = 'Example check for check authoring infrastructure.';

    % --- data file input parameters
    rec.setInputParametersLayoutGrid([1 1]);
    inputParam1 = ModelAdvisor.InputParameter;
    inputParam1.Name = 'Data File';
    inputParam1.Value = 'Check1.xml';
    inputParam1.Type = 'String';
```

```
inputParam1.Description = 'Name or full path of XML data file.';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
rec.setInputParameters({inputParam1});

% -- set fix operation
act = ModelAdvisor.Action;
act.setCallbackFcn(@(task)(Advisor.authoring.CustomCheck.actionCallback(task)));
act.Name = 'Modify Settings';
act.Description = 'Modify model configuration settings.';
rec.setAction(act);

mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);
end
```

## See Also

Advisor.authoring.CustomCheck.actionCallback |  
Advisor.authoring.DataFile |  
Advisor.authoring.generateConfigurationParameterDataFile

## Topics

“Create Check for Model Configuration Parameters”

# Advisor.Manager.createApplication

**Class:** Advisor.Manager

**Package:** Advisor

Create Advisor.Application object

## Syntax

```
app = Advisor.Manager.createApplication()  
app = Advisor.Manager.createApplication(Name, Value)
```

## Description

`app = Advisor.Manager.createApplication()` constructs an Advisor.Application object.

`app = Advisor.Manager.createApplication(Name, Value)` constructs an Advisor.Application object that operates in a temporary working folder.

## Input Arguments

### Name-Value Pair Arguments

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

Example: 'UseTempDir', true specifies that Advisor.Application object operates in a temporary working folder.

**UseTempDir — Create Advisor.Application object that operates in a temporary working folder**

false (default) | true

Data Types: logical

## **Output Arguments**

### **app — Application**

Advisor.Application object

Constructed Advisor.Application object.

## **See Also**

Advisor.Application | getApplication

**Introduced in R2015b**

# delete

**Class:** `Advisor.Application`

**Package:** `Advisor`

Delete `Advisor.Application` object

## Syntax

```
delete(app)
```

## Description

`delete(app)` deletes the `Application` object when you close the root model specified using `Advisor.Application.setAnalysisRoot`, `Application` objects are implicitly closed.

## Examples

```
app = Advisor.Manager.createApplication();  
delete(app)
```

## Input Arguments

**app** — `Advisor.Application` object to destroy

*handle*

`Advisor.Application` object to destroy, as specified by `Advisor.Manager.createApplication`.

## See Also

`Advisor.Manager.createApplication` | `setAnalysisRoot`

**Introduced in R2015b**

# deselectCheckInstances

**Class:** `Advisor.Application`

**Package:** `Advisor`

Clear check instances from Model Advisor analysis

## Syntax

```
deselectCheckInstances(app)
```

```
deselectCheckInstances(app,Name,Value)
```

## Description

You can clear check instances from Model Advisor analysis. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. To obtain the check instance ID, use the `getCheckInstanceIDs` method.

`deselectCheckInstances(app)` clears all check instances from Model Advisor analysis.

`deselectCheckInstances(app,Name,Value)` clears check instances specified by `Name,Value` pair arguments from Model Advisor analysis.

## Input Arguments

**app** — **Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

## **IDs — Checks instance IDs**

cell array

Check instances to clear from Model Advisor analysis, as specified by a cell array of IDs

Data Types: cell

## **Examples**

### **Clear All Check Instances from Model Advisor Analysis**

This example shows how to set the root model, create an `Application` object, set root analysis, and clear checks instances from Model Advisor analysis.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Deselect all checks
deselectCheckInstances(app);
```

### **Clear Check Instance from Model Advisor Analysis Using Instance ID**

This example shows how to set the root model, create an `Application` object, set root analysis, and deselect checks instances using instance IDs.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);
```



```
% Deselect "Identify unconnected lines, input ports, and output  
% ports" check using instance ID  
instanceID = getCheckInstanceIDs(app, 'mathworks.design.UnconnectedLinesPorts');  
checkinstanceID = instanceID(1);  
deselectCheckInstances(app, 'IDs', checkinstanceID);
```

## See Also

Advisor.Manager.createApplication | getCheckInstanceIDs |  
selectCheckInstances | setAnalysisRoot

**Introduced in R2015b**

## deselectComponents

**Class:** `Advisor.Application`

**Package:** `Advisor`

Clear model components from Model Advisor analysis

### Syntax

```
deselectComponents(app)  
deselectComponents(app,Name,Value)
```

### Description

You can clear model components from Model Advisor analysis. A model component is a model in the system hierarchy. Models that the root model references and that `Advisor.Application.setAnalysisRoot` specifies are model components.

`deselectComponents(app)` clears all components from Model Advisor analysis.

`deselectComponents(app,Name,Value)` clears model components specified by `Name,Value` pair arguments from Model Advisor analysis.

### Input Arguments

**app — Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

### **IDs — Component IDs**

cell array

Components to clear from Model Advisor analysis, as specified by a cell array of IDs

Data Types: cell

### **HierarchicalSelection — Clear component and component children**

false (default) | true

Clear components specified by IDs and component children from Model Advisor analysis

Data Types: logical

## **Examples**

### **Clear All Components from Model Advisor Analysis**

This example shows how to set the root model, create an `Application` object, set root analysis, and clear all components from Model Advisor analysis.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';
```

```
% Create an Application object
app = Advisor.Manager.createApplication();
```

```
% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);
```

```
% Deselect all components
deselectComponents(app);
```

### **Clear Components from Model Advisor Analysis Using IDs**

This example shows how to set the root model, create an `Application` object, set root analysis, and clear model components using IDs.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Deselect component using IDs
deselectComponents(app, 'IDs', RootModel);
```

## See Also

[Advisor.Manager.createApplication](#) | [selectComponents](#) | [setAnalysisRoot](#)

**Introduced in R2015b**

# generateReport

**Class:** Advisor.Application

**Package:** Advisor

Generate report for Model Advisor analysis

## Syntax

```
generateReport(app)  
generateReport(app,Name,Value)
```

## Description

Generate a Model Advisor report for an `Application` object analysis.

`generateReport(app)` generates a Model Advisor report for each component specified by the `Application` object. By default, a report with the name of the analysis root is generated in the current folder.

`generateReport(app,Name,Value)` generates a Model Advisor report for each component specified by the `Application` object. Use the `Name,Value` pairs to specify the location and name of the report.

## Input Arguments

**app — Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

## **Location — Path to report location**

character vector

## **Name — Report name**

character vector

# Examples

## **Generate Report**

This example shows how to generate a report with the analysis root name in the current folder.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Run Model Advisor analysis
run(app);

% Generate report
report = generateReport(app);

% Open the report in web browser
web(report);
```

## **Generate Report with Specified Name and Location**

This example shows how to generate a report with a specified name and location.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';
```

```
% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Run Model Advisor analysis
run(app);

% Generate report in my_work directory
mkdir my_work
report = generateReport(app,'Location','my_work','Name','RootModelReport');

%Open the report in web browser
web(report);
```

## See Also

`Advisor.Manager.createApplication` | `run` | `setAnalysisRoot`

**Introduced in R2015b**

## getApplication

**Class:** `Advisor.Manager`

**Package:** `Advisor`

Return handle to `Advisor.Application` object

### Syntax

```
app = getApplication(Name,Value)
```

### Description

`app = getApplication(Name,Value)` returns the handle to an `Advisor.Application` object by using the object properties.

### Input Arguments

#### Name-Value Pair Arguments

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

Example: `'Id',appID` returns handle to an `Advisor.Application` using the object ID.

#### **Id** — `Advisor.Application` object ID

`Advisor.Application` object

Data Types: `function_handle`

#### **Root** — Root model name

character vector

Data Types: `char`



**RootType — Type of root analysis**

'Model' (default) | 'Subsystem'

Data Types: char

## Output Arguments

**app — Handle to Advisor.Application object**

Advisor.Application object

Data Types: function\_handle

## See Also

Advisor.Application | Advisor.Manager.createApplication

**Introduced in R2015b**

## getCheckInstanceIDs

**Class:** `Advisor.Application`

**Package:** `Advisor`

Obtain check instance IDs

### Syntax

```
CheckInstanceIDs = getCheckInstanceIDs (app)
```

```
CheckInstanceIDs = getCheckInstanceIDs (app, CheckID)
```

### Description

Obtain the check instance ID for a check using the check ID. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. The check ID is a static identifier that does not change.

`CheckInstanceIDs = getCheckInstanceIDs (app)` returns a cell array of IDs.

`CheckInstanceIDs = getCheckInstanceIDs (app, CheckID)` returns a instance ID for a check.

### Input Arguments

**app — Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

**CheckID — Check ID associated with Model Advisor check**

character vector

Check ID associated with Model Advisor check.

Example: 'mathworks.design.UnconnectedLinesPorts'

## Output Arguments

### CheckInstanceIDs — Cell array of check instance IDs

cell array

Check instance IDs, returned as a cell array of IDs

## Examples

### Obtain Check Instance IDs

This example shows how to set the root model, create an `Application` object, set root analysis, and obtain the check instance ID.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);

% Select all check instances
selectCheckInstances(app);

% Obtain check instance IDs
CheckInstanceIDs = getCheckInstanceIDs(app);
```

### Obtain Check Instance ID for a Check

This example shows how to set the root model, create an `Application` object, set root analysis, and obtain the check instance ID for check **Identify unconnected lines, input ports**.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';
```

```
% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Select all check instances
selectCheckInstances(app);

% Obtain check instance ID for Model Advisor check "Identify unconnected lines,
%   input ports"
CheckInstanceIDs = getCheckInstanceIDs(app, 'mathworks.design.UnconnectedLinesPorts');
```

## Alternatives

In the left-hand pane of the Model Advisor window, right-click the check and select **Send Check Instance ID to Workspace**.

## See Also

`Advisor.Manager.createApplication` | `selectCheckInstances` | `setAnalysisRoot`

**Introduced in R2015b**

## getEntry

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Get table cell contents

## Syntax

```
content = getEntry(table, row, column)
```

## Description

`content = getEntry(table, row, column)` gets the contents of the specified cell.

## Input Arguments

table	Instantiation of the ModelAdvisor.Table class
row	An integer specifying the row
column	An integer specifying the column

## Output Arguments

content	An element object or object array specifying the content of the table entry
---------	---

## Examples

Get the content of the table cell in the third column, third row:

```
table1 = ModelAdvisor.Table(4, 4);
```

.

```
.  
.content = getEntry(table1, 3, 3);
```

### **See Also**

“Model Advisor Customization”

### **Topics**

“Create Model Advisor Checks”

## getID

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Return check identifier

## Syntax

```
id = getID(check_obj)
```

## Description

`id = getID(check_obj)` returns the ID of the check `check_obj`. `id` is a unique identifier for the check.

You create this unique identifier when you create the check. This unique identifier is the equivalent of the `ModelAdvisor.Check ID` property.

## See Also

“Model Advisor Customization”

## Topics

“Define Custom Checks”

“Create Model Advisor Checks”

## execute

**Class:** `slmetric.Engine`

**Package:** `slmetric`

Collect metric data

## Syntax

```
execute(metric_engine)
execute(slmetric_obj, MetricIDs)
```

## Description

Collect model metric data for the specified metric engine object. The model metric data is based on defined architectural components. The components are these Simulink objects:

- Model
- Subsystem block
- Chart
- MATLAB Function block
- Protected model

`execute(metric_engine)` collects metric data for available model metrics, which can include MathWorks metrics and custom metrics.

`execute(slmetric_obj, MetricIDs)` collects metric data for only the specified metrics, which can be MathWorks metrics or custom metrics.

## Input Arguments

**metric\_engine** — Metric engine object

`slmetric.Engine` object

Create a `slmetric.Engine` object.



```
metric_engine = slmetric.Engine();
```

### **MetricIDs — Metric identifier**

character vector | cell array of character vectors

Metric identifier for “Model Metrics” on page 2-419 or custom model metrics that you create. You can specify one or multiple metric identifiers. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

## **Examples**

### **Collect and Access Metric Data for a Model**

Collect and access model metric data for the model `sldemo_mdref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();
```

```
% Include referenced models and libraries in the analysis.
```

```
% These properties are on by default.
```

```
metric_engine.ModelReferencesSimulationMode = 'AllModes';
```

```
metric_engine.AnalyzeLibraries = 1;
```

```
setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdref_basic');
```

Collect model metric data

```
execute(metric_engine);
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;
        for m=1:length(result)
```

```
        disp(['MetricID: ', result(m).MetricID]);
        disp([' ComponentPath: ', result(m).ComponentPath]);
        disp([' Value: ', num2str(result(m).Value)]);
        disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
    end
else
    disp(['No results for:', result(n).MetricID]);
end
disp(' ');
end
```

## Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdref_basic`.

Create an `slmetric.Engine` object. Include referenced models and libraries in the analysis and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = 1;
setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Display the results for the `mathworks.metrics.ExplicitIOCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ', result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
            disp([' Value: ', num2str(result(m).Value)]);
            disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
            disp([' Measures: ', num2str(result(m).Measures)]);
            disp([' AggregatedMeasures: ', num2str(result(m).AggregatedMeasures)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

Here are the results:

```
MetricID: mathworks.metrics.ExplicitIOCount
  ComponentPath: sldemo_mdref_basic
  Value: 3
  AggregatedValue: 4
  Measures: 0 3
  AggregatedMeasures: 3 3
MetricID: mathworks.metrics.ExplicitIOCount
  ComponentPath: sldemo_mdref_basic/More Info
  Value: 0
  AggregatedValue: 0
  Measures: 0 0
  AggregatedMeasures: 0 0
MetricID: mathworks.metrics.ExplicitIOCount
  ComponentPath: sldemo_mdref_counter
  Value: 4
  AggregatedValue: 4
  Measures: 3 1
  AggregatedMeasures: 3 1
```

For the ComponentPath: `sldemo_mdref_basic`, the value is 3 because there are 3 outputs. The three outputs are in the second element of the Measures array. The `slmetric.metric.AggregationMode` is Max, so the AggregatedValue is 4 which is the number of inputs and outputs to `sldemo_mdref_counter`. The AggregatedMeasures array contains the maximum number of inputs and outputs for a component or subcomponent.

## See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2016a**

## getAnalysisRootMetric

**Class:** `slmetric.Engine`

**Package:** `slmetric`

Get metric data for one metric for analysis root only

### Syntax

```
metricResult = getAnalysisRootMetric(metric_engine, MetricID)
```

### Description

Get metric data from the metric engine where the root of analysis was set using `setAnalysisRoot`.

`metricResult = getAnalysisRootMetric(metric_engine, MetricID)` get the metric data from `metric_engine`, for a specified metric identifier, `MetricID`, only for the analysis root.

### Input Arguments

**metric\_engine** — Collects and accesses metric data

`slmetric.Engine` object

When you call `execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

**MetricID** — Metric identifier

character vector

Metric identifier for “Model Metrics” on page 2-419 or custom model metrics, that you create. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

## Output Arguments

**metricResult** — Result of metric analysis on the analysis root

`slmetric.metric.Result` object

Outputs the object of the `slmetric.metric.Result` object containing the result data for the requested analysis root and metric.

## Examples

### Collect and Access Metric Data for the Analysis Root

This example shows how to set the analysis root, collect, and access the metric data for a metric.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify the model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics for only the analysis root
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);

metricResult = getAnalysisRootMetric(metric_engine, metricID);
```

## See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2017a**

## getErrorLog

**Class:** slmetric.Engine

**Package:** slmetric

Get error log

## Syntax

```
metricLog = getErrorLog(metric_engine)
```

## Description

Get a log of errors and warnings that occurred during metric data collection of a specified metric engine object. The log includes errors that occurred during the execution of metric algorithms, model compilation, and metric data validation.

```
metricLog = getErrorLog(metric_engine).
```

## Input Arguments

**metric\_engine** — Metric engine object

slmetric.Engine object

Constructed slmetric.Engine object.

## Output Arguments

**metricLog** — Log of metric errors and warnings

string array

The `metricLog` string contains the errors and warnings from metric analysis and is formatted in HTML.

## Examples

### Get Error Log

This example shows how to create a `slmetric.Engine` object, set the analysis root, generate metrics, and create and display the error log for the model `sldemo_fuelsys`.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics for only the analysis root
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);

metricLog = getErrorLog(metricEngine);
disp(metricLog);
```

### See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

### Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2017a**



# getMetricDistribution

**Class:** `slmetric.Engine`

**Package:** `slmetric`

Get metric distribution

## Syntax

```
getMetricDistribution(metric_engine, MetricID)
```

## Description

`getMetricDistribution(metric_engine, MetricID)` generates distribution for a specific metric, `MetricID`, for the metric data in the `slmetric.Engine` object, `metric_engine`. The distribution is on the metric data from the `Value` property of a `slmetric.metric.Result` object.

## Input Arguments

**metric\_engine** — Collects and accesses metric data

`slmetric.Engine` object

When you call `execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

**MetricID** — Metric identifier

character vector

Metric identifier for a model metric, specified as a character vector.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

## Output Arguments

### **dist** — Distribution of the metric data

`slmetric.metric.MetricDistribution` object

Distribution of the metric data contains the following properties:

- `MetricID` is a char array that returns the metric ID specified in the `getMetricDistribution` function call.
- `BinCounts` is an `uint64` array of the number of components corresponding to a bin.
- `BinEdges` is a double array of equally spaced edges of each bin.

## Examples

### Generate Metric Distribution

To generate the distribution for a specific metric, create a `slmetric.Engine` object, set the analysis root for the `sldemo_fuelsys` model, and create a histogram of the data. The histogram shows the number of components corresponding to a number of blocks.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'sldemo_fuelsys');

% Collect model metrics and get distribution
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);
dist = getMetricDistribution(metric_engine, metricID);

% View the distribution using a histogram.
histogram('BinEdges',dist.BinEdges,'BinCounts',dist.BinCounts);
```

## See Also

`histcounts` | `slmetric.Engine` | `slmetric.metric.Result` |  
`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

## **Topics**

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2017a**

## getMetrics

**Class:** slmetric.Engine

**Package:** slmetric

Access model metric data

## Syntax

```
Results = getMetrics(metric_engine)
Results = getMetrics(metric_engine, MetricIDs)
Results = getMetrics(metric_engine, MetricIDs, 'AggregationDepth', ad)
```

## Description

Access model metric data from the specified model metric engine. When you call execute, the metric engine collects the metric data. The returned metric data is based on defined architectural components. The components are these Simulink objects:

- Model
- Subsystem block
- Chart
- MATLAB Function block
- Protected model

`Results = getMetrics(metric_engine)` returns metric data for all metrics that the metric engine executed.

`Results = getMetrics(metric_engine, MetricIDs)` returns metric data for the specified metric identifiers.

`Results = getMetrics(metric_engine, MetricIDs, 'AggregationDepth', ad)` returns metric data for the specified metric identifiers and specifying how to aggregate data.

## Input Arguments

### **metric\_engine — Collects and accesses metric data**

`slmetric.Engine` object

When you call `execute`, `metric_engine` collects metric data for all available MathWorks metrics or for the specified `MetricIDs`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

### **MetricIDs — Metric identifier**

character vector | cell array of character vectors

Metric identifier for “Model Metrics” on page 2-419 or custom model metrics that you create. You can specify one or multiple metric identifiers. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

### **AggregationDepth — Depth or level in the component hierarchy to which getMetrics aggregates the metric data**

All (default) | None

Depth or level in the component for which `getMetrics` aggregates the metric data, specified as a name-value pair argument. Values are one of the following:

- **All** — `getMetrics` aggregates the detailed results to the component level. Then, the component level results are used to calculate the aggregated values by traversing the component hierarchy. `getMetrics` returns only the component-level results.
- **None** — Do not aggregate measures and values. If you specify this option, `getMetrics` returns metric values as collected by the metric algorithm. For example, if the metric algorithm returns detailed results, the detailed results are returned without aggregation. `AggregatedValue` and `AggregatedMeasures` properties of the returned `slmetric.metric.Result` objects are empty.

Example: `'AggregationDepth', 'None'`

Data Types: `char`

## Output Arguments

### Results — Metric data from the metric engine

array of `slmetric.metric.Result` objects

Metric data from the metric engine.

## Examples

### Collect and Access Metric Data for a Model

Collect and access model metric data for the model `sldemo_mdref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis.
% These properties are on by default.
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ', result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
            disp([' Value: ', num2str(result(m).Value)]);
            disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
end
```

```
    disp(' ');  
end
```

## See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` |  
`slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2016a**

## getResults

**Class:** `Advisor.Application`

**Package:** `Advisor`

Access Model Advisor analysis results

### Syntax

```
Results = getResults(app)
```

```
Results = getResults(app,Name,Value)
```

### Description

Access `Application` object analysis results.

`Results = getResults(app)` provides access to Model Advisor analysis results.

`Results = getResults(app,Name,Value)`

### Input Arguments

#### **app** — Application

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

### Name-Value Pair Arguments

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

#### **IDs** — Component IDs

cell array



Component IDs, as specified as a cell array of IDs

Data Types: cell

## Output Arguments

### **Result — Analysis results**

cell array of `ModelAdvisor.SystemResult` objects

Analysis results, returned as a cell array of `ModelAdvisor.SystemResult` objects.

## See Also

`Advisor.Manager.createApplication` | `ModelAdvisor.run` |  
`deselectCheckInstances` | `run` | `selectCheckInstances` | `setAnalysisRoot`

**Introduced in R2015b**

## getStatistics

**Class:** `slmetric.Engine`

**Package:** `slmetric`

Get statistics on metric data

### Syntax

```
stats = getStatistics(metric_engine, MetricID)
```

### Description

Generate statistics on the Value properties of the `slmetric.metric.Result` objects for the specified metric engine object, `metric_engine`.

`stats = getStatistics(metric_engine, MetricID)` generate statistics for the specified metric identifier.

### Input Arguments

**metric\_engine** — Collects and accesses metric data

`slmetric.Engine` object

When you call `execute`, `metric_engine` collects metric data for all available metrics or for the specified `MetricID`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

**MetricID** — Metric identifier

character vector

Metric identifier for “Model Metrics” on page 2-419 or custom model metrics that you create. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

Example: `'mathworks.metrics.DescriptiveBlockNames'`

## Output Arguments

### stats — Metric statistics

`slmetric.metric.Statistics` object

The `Statistics` object contains the following properties:

- `MinValue` is a double that returns the minimum of the `Value` of the `slmetric.metric.Result` object.
- `MaxValue` is a double that returns the maximum of the `Value` of the `slmetric.metric.Result` object.
- `MeanValue` is a double that returns the mean of the `Value` of the `slmetric.metric.Result` object.
- `StandardDeviation` is a double that returns the standard deviation of the `Value` of the `slmetric.metric.Result` object.

## Examples

### Collect Statistics

This example shows how to create a `slmetric.Engine` object, set the analysis root, collect the block count metric, and collect statistics for the model `sldemo_fuelsys`.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root','sldemo_fuelsys');

% Generate and collect model metrics
metricID = 'mathworks.metrics.SimulinkBlockCount';
execute(metric_engine, metricID);
stats = getStatistics(metric_engine, metricID);
```

### See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

**Topics**

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2017a**

# loadConfiguration

**Class:** Advisor.Application

**Package:** Advisor

Load Model Advisor configuration

## Syntax

```
loadConfiguration(app, filename)
```

## Description

loadConfiguration(app, filename) loads a Model Advisor configuration MAT-file.

## Input Arguments

**app — Application**

Advisor.Application object

Advisor.Application object, created by Advisor.Manager.createApplication

**filename — Name of Model Advisor configuration MAT-file**

character vector

Name of Model Advisor configuration MAT-file, specified as a character vector.

Example: 'MyConfiguration.mat'

Data Types: char

## See Also

Advisor.Manager.createApplication | setAnalysisRoot

**Introduced in R2015b**

# mdltransformer

Open Model Transformer

## Syntax

```
mdltransformer(model)
```

## Description

`mdltransformer(model)` opens the Model Transformer for a model specified by `model`. If the specified model is not open, this command opens it.

## Examples

### Open Model Transformer for model

Open the Model Transformer for `rtwdemo_reusable_sys_outputs` example model:

```
mdltransformer('rtwdemo_reusable_sys_outputs')
```

## Input Arguments

### **model** — Model name

character vector

Model name or handle, specified as a character vector.

Data Types: char

## **See Also**

### **Topics**

“Transform Model to Variant System”

“Improve Model Readability by Eliminating Local Data Store Blocks”

**Introduced in R2016b**



# metricsdashboard

Open Metrics Dashboard

## Syntax

```
metricsdashboard(system)
```

## Description

`metricsdashboard(system)` opens the Metrics Dashboard for a system specified by `system`. The *system* can be either a model name or a block path to a subsystem. The system cannot be a Configurable Subsystem block.

## Examples

### Open Metrics Dashboard for system

Open the Metrics Dashboard for vdp example model:

```
metricsdashboard('vdp')
```

## Input Arguments

### **system** — System name

character vector

System name, specified as a character vector.

Data Types: char

## **See Also**

### **Topics**

“Collect and Explore Metric Data by Using the Metrics Dashboard”

**Introduced in R2017b**

# slmetric.metric.Metric class

**Package:** slmetric.metric

Abstract class for creating model metrics

## Description

Abstract base class for creating model metrics. To create a model metric, create a MATLAB class that derives from the `slmetric.metric.Metric` class.

## Properties

### **CompileContext** — Compile mode

character vector

Compile mode for metric calculation. If your model metric requires model compilation, specify `PostCompile`. If your model metric does not require model compilation, specify `None`.

Example: `'PostCompile'`

Data Types: `char`

### **ComponentScope** — Component scope

array of `Advisor.component.Types` enum values

Model components for which metric is calculated. The metric is calculated for all components that match the type.

### **Description** — Metric description

character vector

Metric description.

Data Types: `char`

### **ID** — Metric ID

character vector

Unique metric identifier.

Data Types: `char`

### **Version — Metric version number**

`integer`

Use this property to communicate changes in your metric algorithm to the metric engine.

Data Types: `uint32`

### **Name — Name of the metric algorithm**

`character vector`

Specify a name for the custom metric algorithm.

Data Types: `char`

### **ResultChecksumCoverage — Reuse metric data**

`logical`

If `true`, results produced by the metric algorithm change only if the model or library source files change. If the source file and the metric `Version` have not changed, metric data is not regenerated. If `false`, each call to `slmetric.Engine.execute` collects new data for this metric and stores it in the metric repository.

Data Types: `logical`

### **AggregationMode — How the metric algorithm aggregates the metric data**

`character array`

Specify the operation to aggregate the `slmetric.metric.Result` object properties `Value` and `Measure` across the component hierarchy. The metric algorithm outputs the aggregated values in the `slmetric.metric.Result` object properties `AggregatedValues` and `AggregatedMeasures`. Options are:

- **Sum**: Returns the sum of the `Value` property and the `Value` properties of all its children components across the component hierarchy.
- **Max**: Returns the maximum of the `Value` property and the `Value` properties of all its children components across the component hierarchy.
- **None**: No aggregation of metric values.

Data Types: `char`

**AggregateComponentDetails — Aggregate detailed results to the component level**

logical

If `true`, metric results that do not cover the full component scope are aggregated to the component level. Values and measures of the detailed results belonging to a component are summed and a new result is created that spans the complete component.

If `false`, returns the detailed results of the component. Detailed results are not aggregated.

Data Types: logical

**SupportsResultDetails — Specify whether Details property contains data**

logical

Specify whether the `slmetric.metric.Result` object property `Details` contains data. The default value is `false`. Metrics Dashboard filters are enabled when you set the value of this property to `false`.

Data Types: logical

**Methods**

`algorithm`            Specify logic for metric data analysis

**See Also**

`slmetric.Engine` | `slmetric.metric.Result` |  
`slmetric.metric.createNewMetricClass` |  
`slmetric.metric.getAvailableMetrics`

**Topics**

“Create a Custom Model Metric for Nonvirtual Block Count”  
 “Model Metrics” on page 2-419

**Introduced in R2016a**

## algorithm

**Class:** `slmetric.metric.Metric`

**Package:** `slmetric.metric`

Specify logic for metric data analysis

### Syntax

`Result = algorithm(Metric,Component)`

### Description

Specify logic for metric algorithm analysis. Custom-authored metric algorithms are not called for library links and external MATLAB file components.

`Result = algorithm(Metric,Component)` specifies logic for metric algorithm analysis.

### Input Arguments

**Metric — New model metric class**

`slmetric.metric.Metric` object

Model metric class you are defining for a new metric.

**Component — Component for metric analysis**

`Advisor.component.Component` object

Instance of `Advisor.component.Component` for metric analysis.

### Output Arguments

**Result — Algorithm result data**

array of `slmetric.metric.Result` objects

Algorithm data, returned as an array of `slmetric.metric.Result` objects.

## Examples

### Create Metric Algorithm for Nonvirtual Block Count

This example shows how to use the `algorithm` method to create a nonvirtual block count metric.

Using the `createNewMetricClass` function, create a metric class with the name `nonvirtualblockcount`. The function creates the `nonvirtualblockcount.m` file in the current working folder.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Open and edit the metric algorithm file `nonvirtualblockcount.m`. The file contains an empty metric algorithm method.

```
edit(className);
```

Copy and paste the following code into the `nonvirtualblockcount.m` file. Save `nonvirtualblockcount.m`. The code provides a metric algorithm for counting the nonvirtual blocks.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    % nonvirtualblockcount calculate number of non-virtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as non-virtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisibility','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
        end

        function res = algorithm(this, component)
            % create a result object for this component
            res = slmetric.metric.Result();
        end
    end
end
```

```
% set the component and metric ID
res.ComponentID = component.ID;
res.MetricID = this.ID;

% use find_system to get all blocks inside this component
blocks = find_system(getComponentSource(component), ...
    'FollowLinks','on', 'SearchDepth', 1, ...
    'Type', 'Block', ...
    'FollowLinks', 'On');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outputport'
                % Outputport: Virtual when the block resides within
                % any SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            case 'Trigger'
                % Virtual when the output port is not present.
                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                    isNonVirtual(n) = false;
                end
            case 'Enable'
                % Virtual unless connected directly to an Outputport block.
                isNonVirtual(n) = false;

                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
                    pc = get_param(blocks{n}, 'PortConnectivity');

                    if ~isempty(pc.DstBlock) && ...
                        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                            'Outputport')
                end
            end
        end
    end
end
```



```
        isNonVirtual(n) = true;
    end
end
end
end
    blocks = blocks(isNonVirtual);
    res.Value = length(blocks);
end
end
end
```

## See Also

`slmetric.metric.Result` | `slmetric.metric.createNewMetricClass`

## Topics

“Create a Custom Model Metric for Nonvirtual Block Count”

“Model Metrics” on page 2-419

**Introduced in R2016a**

## slmetric.metric.ResultDetail class

**Package:** slmetric.metric

Details about instances of slmetric.metric.Result objects

### Description

Details about what the metric engine counts for the slmetric.metric.Result object property Value.

### Construction

Calling the slmetric.Engine.execute method creates the slmetric.metric.Result objects, which optionally includes the slmetric.metric.ResultDetail objects. Details1 = slmetric.metric.ResultDetail = (ID, Name) creates an slmetric.metric.ResultDetail object. You must supply the ID and Name as inputs to the constructor.

### Properties

#### **ID — Unique identifier**

character vector

Unique identifier for the entity that the result detail instance counts. This property is read/write.

Data Types: char

#### **Name — Name of model entity**

character vector

Name of model entity that result detail instance counts. This property is read/write.

Data Types: char

**Value — Value of ID property**

double

Scalar value generated by metric algorithm for ID. This property is read/write.

Data Types: double

**Methods**

setGroup	Set the name and identifier for a group of <code>slmetric.metric.ResultDetail</code> objects
getGroupIdentifier	Obtain the identifier for a group of <code>slmetric.metric.ResultDetail</code> objects
getGroupName	Obtain the name for a group of <code>slmetric.metric.ResultDetail</code> objects

**Examples****Obtain Clone Group Names and Identifiers**

Use the `getGroupName` and `getGroupIdentifier` methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the `execute` method. Apply the `getMetrics` method for `themathworks.metric.CloneDetection` metric.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', 'ex_clone_detection', 'RootType', 'Model');
execute(metric_engine);
rc = getMetrics(metric_engine, 'mathworks.metrics.CloneDetection');
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetail` object, display the clone group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
```

```
for m=1:length(rc.Results(n).Details)
    disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
    disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
    disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
end
else
    disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
end
disp(' ');
end
```

The results show that the model contains one clone group, `CloneGroup1`, which contains two clones.

## Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file, `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
```

```

        'blocks and groups them by the corresponding Data Store Memory block.';
    end

function res = algorithm(this, component)
    % Use find_system to get all blocks inside this component.
    dswBlocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'BlockType', 'DataStoreWrite');
    dsrBlocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'BlockType', 'DataStoreRead');

    % Create a ResultDetail object for each data store read and write block.
    % Group ResultDetails by the data store name.
    details1 = slmetric.metric.ResultDetail.empty();
    for i=1:length(dswBlocks)
        details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
            get_param(dswBlocks{i}, 'Name'));
        groupID = get_param(dswBlocks{i}, 'DataStoreName');
        groupName = get_param(dswBlocks{i}, 'DataStoreName');
        details1(i).setGroup(groupID, groupName);
        details1(i).Value = 1;
    end

    details2 = slmetric.metric.ResultDetail.empty();
    for i=1:length(dsrBlocks)
        details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
            get_param(dsrBlocks{i}, 'Name'));
        groupID = get_param(dsrBlocks{i}, 'DataStoreName');
        groupName = get_param(dsrBlocks{i}, 'DataStoreName');
        details2(i).setGroup(groupID, groupName);
        details2(i).Value = 1;
    end

    res = slmetric.metric.Result();
    res.ComponentID = component.ID;
    res.MetricID = this.ID;
    res.Value = length(dswBlocks)+ length(dsrBlocks);
    res.Details = [details1 details2];
end
end
end

```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_md1ref_dsm` model.

Load the `sldemo_md1ref_dsm` model.

```
model = 'sldemo_mdref_dsm';  
load_system(model);
```

Create a metric engine object and set the analysis root.

```
metric_engine = slmetric.Engine();  
setAnalysisRoot(metric_engine, 'Root', model, 'RootType', 'Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);  
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)  
    if rc.Results(n).Value > 0  
        for m=1:length(rc.Results(n).Details)  
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);  
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);  
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);  
        end  
    else  
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);  
    end  
    disp(' ');  
end
```

Here are the results.

```
ComponentPath: sldemo_mdref_dsm  
Group Name: ErrorCond  
Group Identifier: ErrorCond
```

```
No results for ComponentPath: sldemo_mdref_dsm/A
```

```
No results for ComponentPath: sldemo_mdref_dsm/A1
```

```
No results for ComponentPath: sldemo_mdref_dsm/More Info1
```

```
ComponentPath: sldemo_mdref_dsm_bot  
Group Name: RefSignalVal  
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdref_dsm_bot2  
Group Name: ErrorCond  
Group Identifier: ErrorCond
```

```
ComponentPath: sldemo_mdref_dsm_bot/PositiveSS  
Group Name: RefSignalVal  
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdref_dsm_bot/NegativeSS  
Group Name: RefSignalVal  
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;  
bdclose('all');
```

## See Also

```
slmetric.metric.Result | slmetric.metric.ResultCollection |  
slmetric.metric.ResultDetail | slmetric.metric.getAvailableMetrics
```

**Introduced in R2017b**

## setGroup

**Class:** `slmetric.metric.ResultDetail`

**Package:** `slmetric.metric`

Set the name and identifier for a group of `slmetric.metric.ResultDetail` objects

## Syntax

```
setGroup(groupIdentifier,groupName)
```

## Description

For a custom-authored metric, set the identifier and name for a group of `slmetric.metric.ResultDetail` objects. Apply this method from within the part of the metric algorithm that specifies the details for `getMetrics` objects.

`setGroup(groupIdentifier,groupName)` sets the values of the group name and identifier for an `slmetric.metric.ResultDetail` object.

## Input Arguments

**groupIdentifier** — Group identifier

character vector

Specify a value for the identifier for a group of `slmetric.metric.ResultDetail` objects.

**groupName** — Group name

character vector

Specify a value for the name of a group of `slmetric.metric.ResultDetail` objects.

## Examples



## Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                'blocks and groups them by the corresponding Data Store Memory block.';
        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');

            % Create a ResultDetail object for each data store read and write block.
            % Group ResultDetails by the data store name.
            details1 = slmetric.metric.ResultDetail.empty();
            for i=1:length(dswBlocks)
                details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
```

```
        get_param(dswBlocks{i}, 'Name'));
groupID = get_param(dswBlocks{i}, 'DataStoreName');
groupName = get_param(dswBlocks{i}, 'DataStoreName');
details1(i).setGroup(groupID, groupName);
details1(i).Value = 1;
end

details2 = slmetric.metric.ResultDetail.empty();
for i=1:length(dsrBlocks)
    details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
        get_param(dsrBlocks{i}, 'Name'));
    groupID = get_param(dsrBlocks{i}, 'DataStoreName');
    groupName = get_param(dsrBlocks{i}, 'DataStoreName');
    details2(i).setGroup(groupID, groupName);
    details2(i).Value = 1;
end

res = slmetric.metric.Result();
res.ComponentID = component.ID;
res.MetricID = this.ID;
res.Value = length(dswBlocks)+ length(dsrBlocks);
res.Details = [details1 details2];
end
end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_mdhref_dsm` model.

Load the `sldemo_mdhref_dsm` model.

```
model = 'sldemo_mdhref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root..

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', model, 'RootType', 'Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
        for m=1:length(rc.Results(n).Details)
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
end
disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdhref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
No results for ComponentPath: sldemo_mdhref_dsm/More Info1
```

```
ComponentPath: sldemo_mdhref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
ComponentPath: sldemo_mdhref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot/NegativeSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;  
bdclose('all');
```

## **See Also**

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` |  
`slmetric.metric.ResultDetail` | `slmetric.metric.getAvailableMetrics`

**Introduced in R2017b**

## getGroupIdentifier

**Class:** `slmetric.metric.ResultDetail`

**Package:** `slmetric.metric`

Obtain the identifier for a group of `slmetric.metric.ResultDetail` objects

### Syntax

```
groupIdentifier = getGroupIdentifier(mrd)
```

### Description

Obtain the identifier for a group of `slmetric.metric.ResultDetail` objects. Calling the `execute` method collects metric data. Calling `getMetrics` accesses the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects. Apply the `getGroupIdentifier` method to the `slmetric.metric.ResultDetail` object.

`groupIdentifier = getGroupIdentifier(mrd)` obtains the group identifier for the `slmetric.metric.ResultDetail` object `mrd`.

### Input Arguments

**mrd** — `slmetric.metric.ResultDetail` object

character vector

Calling the `slmetric.Engine.execute` method creates the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects.

## Output Arguments

### **groupIdentifier** — Group identifier

character vector

Identifier for a group of `slmetric.metric.ResultDetail` objects.

## Examples

### **Obtain Clone Group Names and Identifiers**

Use the `getGroupName` and `getGroupIdentifier` methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the `execute` method. Apply the `getMetrics` method for the `mathworks.metric.CloneDetection` metric.

```
metric_engine = slmetric.Engine();  
setAnalysisRoot(metric_engine,'Root','ex_clone_detection','RootType','Model');  
execute(metric_engine);  
rc = getMetrics(metric_engine,'mathworks.metrics.CloneDetection');
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetail` object, display the clone group name and identifier.

```
for n=1:length(rc.Results)  
    if rc.Results(n).Value > 0  
        for m=1:length(rc.Results(n).Details)  
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);  
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);  
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);  
        end  
    else  
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);  
    end  
    disp(' ');  
end
```

The results show that the model contains one clone group, CloneGroup1, which contains two clones.

## Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a new metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file, `DataStoreCount.m` in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                'blocks and groups them by the corresponding Data Store Memory block.';
        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');

            % Create a ResultDetail object for each data store read and write block.
```

```
% Group ResultDetails by the data store name.
details1 = slmetric.metric.ResultDetail.empty();
for i=1:length(dswBlocks)
    details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
        get_param(dswBlocks{i}, 'Name'));
groupID = get_param(dswBlocks{i}, 'DataStoreName');
groupName = get_param(dswBlocks{i}, 'DataStoreName');
    details1(i).setGroup(groupID, groupName);
    details1(i).Value = 1;
end

details2 = slmetric.metric.ResultDetail.empty();
for i=1:length(dsrBlocks)
    details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
        get_param(dsrBlocks{i}, 'Name'));
groupID = get_param(dsrBlocks{i}, 'DataStoreName');
groupName = get_param(dsrBlocks{i}, 'DataStoreName');
    details2(i).setGroup(groupID, groupName);
    details2(i).Value = 1;
end

res = slmetric.metric.Result();
res.ComponentID = component.ID;
res.MetricID = this.ID;
res.Value = length(dswBlocks)+ length(dsrBlocks);
res.Details = [details1 details2];
end
end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_mdhref_dsm` model.

Load the `sldemo_mdhref_dsm` model.

```
model = 'sldemo_mdhref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root..

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', model, 'RootType', 'Model');
```

Collect metric data for the Data Store count metric.



```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
        for m=1:length(rc.Results(n).Details)
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
end
disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdhref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
No results for ComponentPath: sldemo_mdhref_dsm/More Info1
```

```
ComponentPath: sldemo_mdhref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
ComponentPath: sldemo_mdhref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot/NegativeSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;  
bdclose('all');
```

## **See Also**

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` |  
`slmetric.metric.ResultDetail` | `slmetric.metric.getAvailableMetrics`

**Introduced in R2017b**

## getGroupName

**Class:** `slmetric.metric.ResultDetail`

**Package:** `slmetric.metric`

Obtain the name for a group of `slmetric.metric.ResultDetail` objects

### Syntax

```
groupName = getGroupName(mrd)
```

### Description

Obtain the name of a group of `slmetric.metric.ResultDetail` objects. Calling the `execute` method collects metric data. Calling `getMetrics` accesses the `slmetric.metric.Result` objects which include the `slmetric.metric.ResultDetail` objects. Apply the `getGroupName` method to the `slmetric.metric.ResultDetail` object.

`groupName = getGroupName(mrd)` obtains the name for the `slmetric.metric.ResultDetail` object `mrd`.

### Input Arguments

**mrd** — `slmetric.metric.ResultDetail` object

character vector

Calling the `slmetric.Engine.execute` method creates the `slmetric.metric.Result` objects, which include the `slmetric.metric.ResultDetail` objects.

## Output Arguments

### **groupName** — Group name

character vector

Name for a group of `slmetric.metric.ResultDetail` objects

## Examples

### **Obtain Clone Group Names and Identifiers**

Use the `getGroupName` and `getGroupIdentifier` methods to obtain the name and identifier for a group of clones.

Open the example model.

```
open_system([docroot '\toolbox\simulink\examples\ex_clone_detection.slx']);
```

Save the example model to your current working folder.

Call the `execute` method. Apply the `getMetrics` method for the `mathworks.metric.CloneDetection` metrics.

```
metric_engine = slmetric.Engine();  
setAnalysisRoot(metric_engine,'Root','ex_clone_detection','RootType','Model');  
execute(metric_engine);  
rc = getMetrics(metric_engine,'mathworks.metrics.CloneDetection');
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetail` object, display the clone group name and identifier.

```
for n=1:length(rc.Results)  
    if rc.Results(n).Value > 0  
        for m=1:length(rc.Results(n).Details)  
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);  
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);  
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);  
        end  
    else  
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);  
    end  
    disp(' ');  
end
```

The results show that the model contains one clone group, CloneGroup1, which contains two clones.

## Set Group Names and Group Identifiers for a Custom Model Metric

Use the `setGroup` method to group detailed results. When you create a custom model metric, you apply this method as part of the `algorithm` method.

Using the `createNewMetricClass` function, create a metric class named `DataStoreCount`. This metric counts the number of Data Store Read and Data Store Write blocks and groups them together by the corresponding Data Store Memory block. The `createNewMetricClass` function creates a file, `DataStoreCount.m`, in the current working folder. The file contains a constructor and empty metric algorithm method. For this example, make sure that you are working in a writable folder.

```
className = 'DataStoreCount';
slmetric.metric.createNewMetricClass(className);
```

To write the metric algorithm, open the `DataStoreCount.m` file and add the metric to the file. For this example, you can create the metric algorithm by copying this logic into the `DataStoreCount.m` file.

```
classdef DataStoreCount < slmetric.metric.Metric
    % Count the number of Data Store Read and Data Store Write
    % blocks and correlate them across components.

    methods
        function this = DataStoreCount()
            this.ID = 'DataStoreCount';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
            this.SupportsResultDetails = true;

            %Textual information on the metric algorithm
            this.Name = 'Data store usage';
            this.Description = 'Metric that counts the number of Data Store Read and Write';
                'blocks and groups them by the corresponding Data Store Memory block.';
        end

        function res = algorithm(this, component)
            % Use find_system to get all blocks inside this component.
            dswBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreWrite');
            dsrBlocks = find_system(getPath(component), ...
                'SearchDepth', 1, ...
                'BlockType', 'DataStoreRead');

            % Create a ResultDetail object for each data store read and write block.
```

```
% Group ResultDetails by the data store name.
details1 = slmetric.metric.ResultDetail.empty();
for i=1:length(dswBlocks)
    details1(i) = slmetric.metric.ResultDetail(getfullname(dswBlocks{i}),...
        get_param(dswBlocks{i}, 'Name'));
groupID = get_param(dswBlocks{i}, 'DataStoreName');
groupName = get_param(dswBlocks{i}, 'DataStoreName');
    details1(i).setGroup(groupID, groupName);
    details1(i).Value = 1;
end

details2 = slmetric.metric.ResultDetail.empty();
for i=1:length(dsrBlocks)
    details2(i) = slmetric.metric.ResultDetail(getfullname(dsrBlocks{i}),...
        get_param(dsrBlocks{i}, 'Name'));
groupID = get_param(dsrBlocks{i}, 'DataStoreName');
groupName = get_param(dsrBlocks{i}, 'DataStoreName');
    details2(i).setGroup(groupID, groupName);
    details2(i).Value = 1;
end

res = slmetric.metric.Result();
res.ComponentID = component.ID;
res.MetricID = this.ID;
res.Value = length(dswBlocks)+ length(dsrBlocks);
res.Details = [details1 details2];
end
end
end
```

In the `DataStoreCount` metric class, the `SupportsResultDetail` method is set to true. The metric algorithm contains the logic for the `setGroup` method.

Now that your new model metric is defined in `DataStoreCount.m`, register the new metric.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To collect metric data on models, use instances of `slmetric.Engine`. Using the `getMetrics` method, specify the metric that you want to collect. For this example, specify the data store count metric for the `sldemo_mdref_dsm` model.

Load the `sldemo_mdref_dsm` model.

```
model = 'sldemo_mdref_dsm';
load_system(model);
```

Create a metric engine object and set the analysis root.

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', model, 'RootType', 'Model');
```

Collect metric data for the Data Store count metric.

```
execute(metric_engine);
rc=getMetrics(metric_engine, id_metric);
```

For each `slmetric.metric.Result` object, display the `ComponentPath`. For each `slmetric.metric.ResultDetails` object, display the Data Store group name and identifier.

```
for n=1:length(rc.Results)
    if rc.Results(n).Value > 0
        for m=1:length(rc.Results(n).Details)
            disp(['ComponentPath: ',rc.Results(n).ComponentPath]);
            disp(['Group Name: ',rc.Results(n).Details(m).getGroupName]);
            disp(['Group Identifier: ',rc.Results(n).Details(m).getGroupIdentifier]);
        end
    else
        disp(['No results for ComponentPath: ',rc.Results(n).ComponentPath]);
    end
end
disp(' ');
end
```

Here are the results.

```
ComponentPath: sldemo_mdhref_dsm
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
No results for ComponentPath: sldemo_mdhref_dsm/More Info1
```

```
ComponentPath: sldemo_mdhref_dsm_bot
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot2
Group Name: ErrorCond
Group Identifier: ErrorCond
```

```
ComponentPath: sldemo_mdhref_dsm_bot/PositiveSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

```
ComponentPath: sldemo_mdhref_dsm_bot/NegativeSS
Group Name: RefSignalVal
Group Identifier: RefSignalVal
```

For this example, unregister the data store count metric.

```
slmetric.metric.unregisterMetric(id_metric);
```

Close the model.

```
clear;  
bdclose('all');
```

## **See Also**

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` |  
`slmetric.metric.ResultDetail` | `slmetric.metric.getAvailableMetrics`

**Introduced in R2017b**



# slmetric.config.Classification class

**Package:** `slmetric.config`

Specify categorical metric data ranges

## Description

Use the `slmetric.config.Classification` class to classify metric data ranges as `Compliant`, `Warning`, and `NonCompliant`. The Metrics Dashboard indicates the range that your metric data falls under.

## Construction

For an `slmetric.config.Threshold` object, there must be one `slmetric.config.Classification` object corresponding to the `Compliant` range. There can be only one compliant range. You can specify multiple `slmetric.config.Classification` objects corresponding to `Warning` and `Noncompliant` ranges.

By default, threshold objects contain an `slmetric.config.Classification` object with a `Compliant` range of `-inf` to `inf`. To add additional classification objects, use the `slmetric.config.Classification.addClassification` method.

## Properties

### Category — Categorize metric data

'Compliant' (default) | 'Warning' | 'NonCompliant'

You can classify metric data values into these three categories:

- `Compliant` — Metric data that is in an acceptable range.
- `Warning` — Metric data that requires review.
- `Noncompliant` — Metric data that requires you to modify your model.

This property is read/write.

Data Types: char

## Range — Metric range object

`slmetric.config.MetricRange` object

For each `slmetric.metric.config.Classification` object, specify the properties of the `slmetric.config.MetricRange` object. This property is read/write.

## Examples

### Specify Metric Thresholds to Add to Metric Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in CONF.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the Warning metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and 5. This range does not include `-inf`. It does include 5.

Specify values for the NonCompliant metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than 100 is noncompliant. This range includes 100. It does not include `inf`.

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';
save(CONF, 'FileName', configName);
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

### See Also

`slmetric.config.Configuration` | `slmetric.config.MetricRange` |  
`slmetric.config.Threshold` | `slmetric.config.ThresholdConfiguration` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration` |  
`slmetric.metric.ResultClassification`

### Topics

[“Collect and Explore Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

# slmetric.config.Configuration class

**Package:** slmetric.config

Specify metric data categories and custom metric families

## Description

Instances of `slmetric.config.Configuration` contain customizations pertaining to thresholds and custom metric families. The metric engine uses these customizations when collecting data and displays them on the Metrics Dashboard.

## Construction

Use the `slmetric.config.Configuration` class to add metric threshold values and custom metric families to the Metrics Dashboard. To create an `slmetric.config.Configuration` object, use the `new` method. Each `slmetric.config.Configuration` object contains one `slmetric.config.ThresholdConfiguration` object.

## Properties

### **Name — Configuration object name**

character vector | string scalar

Name of configuration object that you use to create Metrics Dashboard customizations. This property is read/write.

Data Types: char

### **FileName — Name of XML file that contains custom configurations**

character vector | string scalar

Name of the XML file that contains Metrics Dashboard customizations. This property is read/write.

Data Types: char

## Location — Location of XML file that contains custom configuration

character vector | string scalar

Location of the XML file that contains Metrics Dashboard customizations. This property is optional and is read/write.

## Methods

<code>getThresholdConfigurations</code>	Specify metric threshold configurations
<code>new</code>	Create configuration object for customizing the Metrics Dashboard
<code>openDefaultConfiguration</code>	Return shipping Metrics Dashboard configuration object in base workspace
<code>open</code>	Create configuration object associated with XML configuration file in the base workspace
<code>save</code>	Save contents of <code>slmetric.config.Configuration</code> object to XML file
<code>getMetricFamilyParameterValues</code>	Obtain metric family Check Group IDs
<code>isMetricFamilyParameterParameterized</code>	Determine whether Metrics Dashboard configuration object has metric family parameter values
<code>resetMetricFamilyParameterValues</code>	Clear metric family parameter values
<code>setMetricFamilyParameterValues</code>	Obtain compliance and issues metric data on your Model Advisor configuration

## Examples

### Specify Metric Thresholds to Add to Metrics Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the `Warning` metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and 5. This range does not include `-inf`. It does include 5.

Specify values for the `NonCompliant` metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than 100 is noncompliant. This range includes 100. It does not include `inf`.

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';
save(CONF, 'FileName', configName);
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

## See Also

`slmetric.config.Classification` | `slmetric.config.MetricRange` |  
`slmetric.config.Threshold` | `slmetric.config.ThresholdConfiguration` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration` |  
`slmetric.metric.ResultClassification`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



# slmetric.config.MetricRange class

**Package:** slmetric.config

Specify metric data threshold values

## Description

Specify metric data thresholds corresponding to the `Category` property of an `slmetric.config.Classification` object. These thresholds define metric data ranges for these three categories: complaint, noncompliant, and warning. The Metrics Dashboard alerts you to the category that your data falls under.

## Construction

Use the `slmetric.config.Threshold.getClassifications` method to access the default Compliant `slmetric.config.Classification` object. Or, use the `slmetric.config.Threshold.addClassification` method to create NonCompliant and Warning `slmetric.config.Classification` objects. Then write directly to the `slmetric.config.MetricRange` properties.

## Properties

### Start — Beginning of a metric data range

-inf (default)

Specify the beginning of a metric range corresponding to the `Category` property of an `slmetric.config.Classification` object. This property is read/write.

Data Types: double

### End — End of a metric data range

inf (default)

Specify the end of a metric range corresponding to the `Category` property of an `slmetric.config.Classification` object. This property is read/write.

Data Types: `double`

## **IncludeStart** — Include the value of the Start property

0 (default)

Specify whether to include the Start value in the metric data range corresponding to the Category property of an `slmetric.config.Classification` object. This property is read/write.

Data Types: `logical`

## **IncludeEnd** — Include the value of the End property

0 (default)

Specify whether to include the End value in the metric data range corresponding to the Category property of an `slmetric.config.Classification` object. This property is read/write.

Data Types: `logical`

---

**Note** For the **High Integrity Compliance**, **MAAB Compliance**, **Actual Reuse**, and **Potential Reuse** widgets, you must specify the metric ranges as fractions.

---

## Examples

### **Specify Metric Thresholds to Add to Metrics Dashboard**

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in CONF.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the `Warning` metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and 5. This range does not include `-inf`. It does include 5.

Specify values for the `NonCompliant` metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than 100 is noncompliant. This range includes 100. It does not include `inf`.

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

`validate(T)`

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';  
save(CONF, 'FileName', configName);  
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

## See Also

```
slmetric.config.Classification | slmetric.config.Configuration |  
slmetric.config.Threshold | slmetric.config.ThresholdConfiguration |  
slmetric.config.getActiveConfiguration |  
slmetric.config.setActiveConfiguration |  
slmetric.metric.ResultClassification
```

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.config.Threshold class

**Package:** slmetric.config

Object for holding metric result thresholds

## Description

Object for holding metric data thresholds

## Construction

For an `slmetric.config.ThresholdConfiguration` object, use the `addThreshold` method to create an `slmetric.config.Threshold` object. You can add multiple threshold objects to the same threshold configuration object. Each threshold object is for specifying threshold values for a specific model metric. You can specify metric values for the `Value` or `AggregatedValue` properties of an `slmetric.metric.Result` object.

## Properties

### **MetricID — Metric identifier**

character vector | string scalar

Metric identifier for model metric or custom model metric that you create. This property is read-only.

Example: `'mathworks.metrics.SimulinkBlockCount'`

Data Types: char

### **AppliesTo — Result object property**

character vector | string scalar

`slmetric.metric.Result` property to which you apply thresholds. You can apply thresholds to the `Value` and `AggregatedValue` properties. This property is read-only.

Data Types: char

## Methods

<code>addClassification</code>	Add metric data classification to <code>slmetric.config.Threshold</code> object
<code>getClassifications</code>	Obtain metric data classifications
<code>removeClassification</code>	Remove metric threshold classification
<code>validate</code>	Validate metric range thresholds

## Examples

### Specify Metric Thresholds to Add to Metrics Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- **Compliant** — Metric data that is an acceptable range.
- **Warning** — Metric data that requires review.
- **Noncompliant** — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the **Compliant**

category. Use the `slmetric.metric.MetricRange` class to specify metric values for the Compliant metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the Warning metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and 5. This range does not include `-inf`. It does include 5.

Specify values for the NonCompliant metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than 100 is noncompliant. This range includes 100. It does not include `inf`.

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';  
save(CONF, 'FileName', configName);  
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

### See Also

```
slmetric.config.Classification | slmetric.config.Configuration |  
slmetric.config.MetricRange | slmetric.config.ThresholdConfiguration |  
slmetric.config.getActiveConfiguration |  
slmetric.config.setActiveConfiguration |  
slmetric.metric.ResultClassification
```

### Topics

[“Collect and Explore Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

### Introduced in R2018b



# slmetric.config.ThresholdConfiguration class

**Package:** slmetric.config

Specify metric and slmetric.metric.Result property for thresholding

## Description

Instances of slmetric.config.ThresholdConfiguration contain thresholds that you specify for a metric. Each threshold specification corresponds to an slmetric.config.Threshold object. An slmetric.config.ThresholdConfiguration object can hold multiple slmetric.config.Threshold objects.

## Construction

For an slmetric.config.Configuration object, use the getThresholdConfigurations method to access an slmetric.config.ThresholdConfiguration object.

## Methods

addThreshold	Create an slmetric.config.Threshold object
getThresholds	Obtain properties of threshold objects
removeThreshold	Remove threshold object from threshold configuration object

## Examples

## Specify Metric Thresholds to Add to Metrics Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the `Warning` metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and `5`. This range does not include `-inf`. It does include `5`.

Specify values for the `NonCompliant` metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than `100` is noncompliant. This range includes `100`. It does not include `inf`.

Use the `slmetric.config.validate` function to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';
save(CONF, 'FileName', configName);
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

## See Also

`slmetric.config.Classification` | `slmetric.config.Configuration` |  
`slmetric.config.MetricRange` | `slmetric.config.Threshold` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration` |  
`slmetric.metric.ResultClassification`

## **Topics**

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.metric.ResultClassification class

**Package:** `slmetric.metric`

Access metric data thresholds results

## Description

For the `Value` and `AggregatedValue` properties of an `slmetric.metric.Result` object, access properties of the `slmetric.metric.ResultClassification` class to determine the metric data ranges that correspond to the `Compliant`, `NonCompliant`, and `Warning` categories. From an `slmetric.metric.ResultClassification` object, also determine which of the three categories your metric data falls under.

## Construction

The value of the `Classifications` property of an `slmetric.metric.Result` object is the `slmetric.metric.ResultClassification` object.

## Properties

**Threshold** — Model metric and `slmetric.metric.Result` property with thresholds

`slmetric.config.Threshold` object

Access this property to determine the model metric and the `slmetric.metric.Result` property that has thresholds.

**Classification** — Status of component data

'Compliant' | 'NonCompliant' | 'Warning' | 'Uncategorized'

Metric data values fall into one of these four categories:

- **Compliant** — Metric data that is in an acceptable range.
- **Warning** — Metric data that requires review.

- `NonCompliant` — Metric data that requires you to modify your model.
- `Uncategorized` — Metric data that has no threshold values set.

If at least one component is `NonCompliant`, this property returns `NonCompliant`. If at least one component is `Warning` and no components are `NonCompliant`, this property returns `Warning`. If all components are `Compliant`, this category returns `Compliant`.

This property is read-only.

## Examples

### Collect and Classify Metric Data

For the `mathworks.metric.SimulinkBlockCount` metric, define `slmetric.metric.Result` values corresponding to `Compliant`, `NonCompliant`, and `Warning` categories. For the `sldemo_md1_ref` model, run the metrics engine and categorize results for this metric.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant`, `NonCompliant`, and `Warning` metric ranges.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
```

```

C.Range.IncludeEnd = 0;

C = addClassification(T,'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1

C = addClassification(T,'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;

```

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```

configName = 'Config.xml';
save(CONF,'FileName', configName);
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));

```

Create an `slmetric.Engine` object, set the root in the model for analysis, and collect data for the `mathworks.metrics.SimulinkBlockCount` metric.

```

metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdlref_basic');
execute(metric_engine, 'mathworks.metrics.SimulinkBlockCount');

```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.SimulinkBlockCount');
```

Display the results for the `mathworks.metrics.SimulinkBlockCount` metric.

```

for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ',result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
        end
    end
end

```

```
        disp([' Value: ', num2str(result(m).Value)]);
        disp([' Classifications: ', result(m).Classifications.Category]);
        disp([' Measures: ', num2str(result(m).Measures)]);
        disp([' AggregatedMeasures: ', num2str(result(m).AggregatedMeasures)]);
    end
else
    disp(['No results for:', result(n).MetricID]);
end
disp(' ');
end
```

```
MetricID: mathworks.metrics.SimulinkBlockCount
ComponentPath: sldemo_mdhref_basic
Value: 12
Classifications: Compliant
Measures:
AggregatedMeasures:
MetricID: mathworks.metrics.SimulinkBlockCount
ComponentPath: sldemo_mdhref_basic/More Info
Value: 0
Classifications: Warning
Measures:
AggregatedMeasures:
MetricID: mathworks.metrics.SimulinkBlockCount
ComponentPath: sldemo_mdhref_counter
Value: 18
Classifications: Compliant
Measures:
AggregatedMeasures:
```

For ComponentPath: `sldemo_mdhref_basic` and ComponentPath: `sldemo_mdhref_counter`, the results are Compliant because of the values 12 and 18, respectively. For ComponentPath: `sldemo_mdhref_basic/More Info`, the results fall under the Warning category because of the 0 value.

## See Also

```
slmetric.config.Classification | slmetric.config.Configuration |
slmetric.config.MetricRange | slmetric.config.Threshold |
slmetric.config.ThresholdConfiguration |
slmetric.config.getActiveConfiguration |
slmetric.config.setActiveConfiguration
```

**Introduced in R2018b**



# getThresholdConfigurations

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Specify metric threshold configurations

## Syntax

```
TH = getThresholdConfigurations(CO)
```

## Description

`TH = getThresholdConfigurations(CO)` returns the `slmetric.config.ThresholdConfiguration` object that an `slmetric.config.Configuration` object owns. Use this object to hold specific metric threshold configurations. Metric threshold configurations are compliant, warning, and noncompliant ranges for a specific metric.

## Input Arguments

**CO — Configuration object**

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object for which you create a metric threshold configuration. By default, an `slmetric.config.Configuration` object holds an empty `slmetric.config.ThresholdConfiguration` object.

## Output Arguments

**TH — Metric threshold configuration object**

`slmetric.config.ThresholdConfiguration` object

`slmetric.config.ThresholdConfiguration` object for which you add thresholds corresponding to compliant, noncompliant, and warning ranges for a specific metric.

## Examples

### Add Thresholds to a Threshold Configuration Object

By default, an `slmetric.config.Configuration` object holds one `slmetric.config.ThresholdConfiguration` object. Use the `getThresholdConfigurations` method to add this object to the base workspace. You can then use the `slmetric.config.addThreshold` method to add `slmetric.config.Threshold` objects to this `slmetric.config.ThresholdConfiguration` object.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object `TC`. This threshold is for the `mathworks.metrics.SubSystemCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
E = addThreshold(TC, 'mathworks.metrics.SubSystemCount', 'Value');
```

Use the `slmetric.config.Classification` and `slmetric.config.MetricRange` class properties to specify threshold values corresponding to the `mathworks.metrics.SubsystemCount` metric.

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration` | `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## **slmetric.config.Configuration.new**

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Create configuration object for customizing the Metrics Dashboard

### **Syntax**

```
Co = slmetric.config.Configuration.new('Name', 'Config')
```

### **Description**

Create an `slmetric.config.Configuration` object for holding Metrics Dashboard customizations pertaining to metric thresholds and custom metric families. Use the `save` command to create and store the associated XML configuration file.

`Co = slmetric.config.Configuration.new('Name', 'Config')` creates a configuration object.

### **Input Arguments**

**Name — Name of configuration object that is tagged in XML file**

character vector | string scalar

Name of configuration object in XML file that contains Metrics Dashboard customizations pertaining to metric thresholds and custom metric families.

Data Types: char

### **Output Arguments**

**co — Configuration object**

character vector | string scalar

Name of `slmetric.config.Configuration` object that contains Metrics Dashboard customizations pertaining to metric thresholds and custom families.

Data Types: `char`

## Examples

### Create a Configuration Object

Use the new method to create an `slmetric.config.Configuration` object. The configuration object contains information on custom metric families and metric thresholds. As an input, specify a configuration object name. This name is then associated with a tag in the configuration object XML file. After adding information to the configuration object, use the `slmetric.config.Configuration.save` method to create and store the associated XML file.

```
CONF = slmetric.config.Configuration.new('Name', 'Config')
```

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## openDefaultConfiguration

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Return shipping Metrics Dashboard configuration object in base workspace

### Syntax

```
DT = slmetric.config.Configuration.openDefaultConfiguration
```

### Description

`DT = slmetric.config.Configuration.openDefaultConfiguration` returns the `slmetric.config.Configuration` object corresponding to the shipping Metrics Dashboard configuration in the base workspace. Use this object to add or remove threshold values corresponding to Compliant, NonCompliant, or Warning categories. These MetricIDs contain default shipping thresholds:

- `mathworks.metrics.CloneContent`
- `mathworks.metrics.CyclomaticComplexity`
- `mathworks.metrics.DiagnosticWarningsCount`
- `mathworks.metrics.MatlabCodeAnalyzerWarnings`
- `mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178`
- `mathworks.metrics.ModelAdvisorCheckCompliance.maab`
- `mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178`
- `mathworks.metrics.ModelAdvisorCheckIssues.maab`

You can also use this object to obtain compliance and issues metric data on your Model Advisor configuration.

## Output Arguments

### **DT — Default Metric Dashboard threshold configuration object**

`slmetric.config.Configuration` object

`slmetric.config.ThresholdConfiguration` object for adding and removing thresholds corresponding to Compliant, Noncompliant, and Warning Categories for a specific metric.

## Examples

### **Open the Shipping `slmetric.config.Configuration` Object**

Use the `openDefaultConfiguration` method to add the shipping `slmetric.config.Configuration` object to the base workspace. If you modify the information that this configuration object contains, use the `slmetric.config.Configuration.save` method to save this information to an XML file.

```
Config = slmetric.config.Configuration.openDefaultConfiguration
```

## See Also

`slmetric.config.Classification` | `slmetric.config.MetricRange` |  
`slmetric.config.Threshold` | `slmetric.config.ThresholdConfiguration` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration` |  
`slmetric.metric.ResultClassification`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## slmetric.config.Configuration.open

**Class:** slmetric.config.Configuration

**Package:** slmetric.config

Create configuration object associated with XML configuration file in the base workspace

### Syntax

```
Co = slmetric.config.Configuration.open(  
'FileName','myConfig.xml',... 'Location', pwd)
```

### Description

Reads the contents of an XML file containing Metrics Dashboard customizations into memory and returns the corresponding configuration object. The XML file contains customizations pertaining to metric thresholds and custom metric families. If you modify the contents of the configuration object, invoke the `save` method to write to the associated XML file.

```
Co = slmetric.config.Configuration.open(  
'FileName','myConfig.xml',... 'Location', pwd) reads a configuration file.
```

---

**Note** If you do not supply an input argument, the `slmetric.config.Configuration.open` command reads the contents of the default Metrics Dashboard configuration XML file into memory and returns the corresponding `slmetric.dashboard.Configuration` object.

---

### Input Arguments

**FileName** — Name of XML file

character vector | string scalar

Name of XML file containing Metrics Dashboard customizations pertaining to metric thresholds and custom metric families.



Data Types: char

### **Location — Folder containing XML file**

character vector | string scalar

Name of folder containing XML file that contains Metrics Dashboard customizations pertaining to metric thresholds and custom metric families. This input argument is optional.

Data Types: char

### **Locale — Name of folder containing XML file**

character vector | string scalar

Name of folder containing XML file that contains Metrics Dashboard customizations. This input argument is optional.

Data Types: char

## **Output Arguments**

### **Co — Configuration object**

character vector | string scalar

slmetric.config.Configuration object that you want to open.

Data Types: char

## **Examples**

### **Access an Existing Configuration Object**

Use the `open` method to add an existing `slmetric.config.Configuration` object to the base workspace. As an input, specify the name of the XML file that contains the information on the custom metric families and metric thresholds corresponding to the configuration object. If you modify the information that this configuration object contains, use the `slmetric.config.Configuration.save` method to save this information to the XML file.

```
CONF = slmetric.config.Configuration.open('FileName', 'myConfig.xml',...  
    'Location', pwd());
```

### See Also

[slmetric.config.Configuration](#) | [slmetric.config.getActiveConfiguration](#)  
| [slmetric.config.setActiveConfiguration](#)

### Topics

[“Collect and Explore Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## save

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Save contents of `slmetric.config.Configuration` object to XML file

## Syntax

```
save(Config, 'FileName', 'myConfig.xml', ... 'Location', pwd, 'locale',  
'ja_JP');
```

## Description

Save the contents of a configuration object to an XML file. The configuration object contains Metrics Dashboard customizations pertaining to metric thresholds and custom metric families.

```
save(Config, 'FileName', 'myConfig.xml', ... 'Location', pwd, 'locale',  
'ja_JP');
```

 saves the contents of a configuration object to an XML file.

---

**Note** Do not manually edit the XML file.

---

## Input Arguments

### **Config** — Configuration object

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object to create Metrics Dashboard customizations. This property is read/write.

### **FileName** — Name of XML file

character vector | string scalar

Name of XML file that contains Metrics Dashboard customizations pertaining to metric thresholds and custom metric families.

Data Types: char

### **Location — Name of folder containing XML file**

character vector | string scalar

Name of folder containing XML file, which contains Metrics Dashboard customizations pertaining to metric thresholds and custom metric families.

Data Types: char

### **locale — Create folder that is to contain XML file**

character vector | string scalar

Name of new folder that is to contain XML file that contains information on Metrics Dashboard customizations pertaining to metric thresholds and custom metric families. If you do not specify a value for `locale`, Simulink creates the XML file in the folder that you specify with the `Location` property. This input argument is optional.

Data Types: char

## Examples

### **Serialize a Configuration Object to XML File**

Serialize configuration object to XML file.

Use the `save` method to add an existing `slmetric.config.Configuration` object to the base workspace. As an input, specify the name of the XML file that contains information on the custom metric families and metric thresholds corresponding to the configuration object. If you modify the information that this configuration object contains, use the `slmetric.config.Configuration.save` method to save the information to this file.

```
save(CONF, 'config', 'FileName', 'Configfile.xml', 'Location', pwd)
```

Use the `slmetric.config.setActiveConfiguration` function to specify that the metric engine use this configuration.

```
slmetric.config.setActiveConfiguration('C:\temp\Configfile.xml');
```

## See Also

[slmetric.config.Configuration](#) | [slmetric.config.getActiveConfiguration](#)  
| [slmetric.config.setActiveConfiguration](#)

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## addClassification

**Class:** `slmetric.config.Threshold`

**Package:** `slmetric.config`

Add metric data classification to `slmetric.config.Threshold` object

### Syntax

```
TC = addClassification(T,Category)
```

### Description

Create and add a classification object to a threshold object. For the classification object, use the `slmetric.config.MetricRange` class to specify ranges corresponding to `Compliant`, `NonCompliant`, and `Warning`. By default, a classification object has a `Compliant` range of `-inf` to `inf`. The Metrics Dashboard indicates the range that your metric data falls under.

`TC = addClassification(T,Category)` adds a classification range to an `slmetric.config.Classification` object.

### Input Arguments

**T — Add metric data classifications**

`slmetric.config.threshold` object

`slmetric.config.threshold` object for which you specify threshold values corresponding to `Category` property.

**Category — Threshold category**

'Compliant' (default) | 'Warning' | 'NonCompliant'

Classify metric data values into these three categories:

- Compliant—Metric data that is in an acceptable range.
- Warning—Metric data that requires review.
- NonCompliant—Metric data that requires you to modify your model.

Data Types: char

## Output Arguments

### TC — Add classification to threshold object

`slmetric.config.Classification` object

`slmetric.config.Classification` object for which you want to specify ranges corresponding to Compliant, Warning, or NonCompliant.

## Examples

### Specify Metric Thresholds to Add to Metric Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in CONF.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the `Warning` metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and 5. This range does not include `-inf`. It does include 5.

Specify values for the `NonCompliant` metric range.

```
C = addClassification(T, 'NonCompliant');
C.Range.Start = 100;
C.Range.IncludeStart = 1;
C.Range.End = inf;
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than 100 is noncompliant. This range includes 100. It does not include `inf`.

Use the `validate` method to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.



Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';  
save(CONF, 'FileName', configName);  
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

## See Also

`slmetric.config.Threshold` | `slmetric.config.getActiveConfiguration` | `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

## Introduced in R2018b

## getClassifications

**Class:** `slmetric.config.Threshold`

**Package:** `slmetric.config`

Obtain metric data classifications

### Syntax

```
C = getClassifications(T)
```

### Description

Identify the classification objects that are in a threshold object.

`C = getClassifications(T)` returns a `slmetric.config.Classification` object or an array of `slmetric.config.Classification` objects.

### Input Arguments

**T — Determine metric data thresholds**

`slmetric.config.Threshold` object

`slmetric.config.Threshold` object for which you want metric data classifications

### Output Arguments

**C — Classification object**

`slmetric.config.Classification` object | array of `slmetric.config.classification` objects

`slmetric.config.Classification` object that contains metric data classifications.

## Examples

### Remove Classification Object from Threshold Object

Use the `getClassifications` method to identify the `slmetric.config.Classification` objects that belong to an `slmetric.config.Threshold` object. Then, use the `removeClassification` method to remove an `slmetric.config.Classification` object from the `slmetric.config.Threshold` object.

An `slmetric.config.Threshold` object `T` contains three `slmetric.config.Classification` objects. Each one corresponds to `Compliant`, `Noncompliant`, and `Warning` categories.

```
P = getClassifications(T)
```

```
P =
```

```
1x3 Classification array with properties:
```

```
Category
Range
```

Choose which `slmetric.config.Classification` object to remove from the `slmetric.config.Threshold` object.

```
P.Category
```

```
P.Category
```

```
ans =
```

```
'Warning'
```

```
ans =
```

```
'Compliant'
```

```
ans =
```

```
'NonCompliant'
```

Use the `removeClassification` method to remove the a category from the `slmetric.config.Threshold` object. This command removes the `slmetric.config.Classification` object corresponding to the `Warning` category.

```
removeClassification(T,P(1))
```

### See Also

`slmetric.config.Threshold` | `slmetric.config.getActiveConfiguration` | `slmetric.config.setActiveConfiguration`

### Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# removeClassification

**Class:** `slmetric.config.Threshold`

**Package:** `slmetric.config`

Remove metric threshold classification

## Syntax

```
C = removeClassification(T,C1)
```

## Description

Remove a classification object from a threshold object.

`C = removeClassification(T,C1)` removes the `slmetric.config.Classification` object `C1` from the `slmetric.config.Threshold` object `T`.

## Input Arguments

**T** — **Threshold object from which you want to remove a classification object**

`slmetric.config.Threshold` object

Remove an `slmetric.config.Classification` object from this `slmetric.config.Threshold` object.

**C1** — **Classification object that you want to remove from a threshold object**

`slmetric.config.Classification` object

Remove this `slmetric.config.Classification` object from an `slmetric.config.Threshold` object.

## Examples

## Remove Classification Object from Threshold Object

Identify the `slmetric.config.Classification` objects that belong to an `slmetric.config.Threshold` object. Then, use the `removeClassification` method to remove an `slmetric.config.Classification` object from the `slmetric.config.Threshold` object.

An `slmetric.config.Threshold` object `T` contains three `slmetric.config.Classification` objects. Each one corresponds to `Compliant`, `Noncompliant`, and `Warning` categories.

```
P = getClassifications(T)
```

```
P =
```

```
1x3 Classification array with properties:
```

```
Category  
Range
```

Choose which `slmetric.config.Classification` object to remove from the `slmetric.config.Threshold` object.

```
P.Category
```

```
P.Category
```

```
ans =
```

```
'Warning'
```

```
ans =
```

```
'Compliant'
```

```
ans =
```

```
'NonCompliant'
```

Use the `removeClassification` method to remove the a category from the `slmetric.config.Threshold` object. This command removes the `slmetric.config.Classification` object corresponding to the `Warning` category.

```
removeClassification(T,P(1))
```

## See Also

```
slmetric.config.Threshold | slmetric.config.getActiveConfiguration |  
slmetric.config.setActiveConfiguration
```

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## validate

**Class:** `slmetric.config.Threshold`

**Package:** `slmetric.config`

Validate metric range thresholds

## Syntax

```
validate(T)
```

## Description

`validate(T)` checks each `slmetric.config.Classification` object, to verify that the values specified for the `Category` and `Range` properties are valid. The range corresponding to each category must not overlap and the ranges together must cover from `-inf` to `inf`. If the values are not valid, you get a message informing you of what you must fix.

## Input Arguments

**T — Threshold object to validate**

`slmetric.config.Threshold` object

`slmetric.config.Threshold` object for which you want to validate the metric threshold values.

## Examples



## Specify Metric Thresholds to Add to Metric Dashboard

Use the `slmetric.config` packaged classes to add threshold information to the Metrics Dashboard. You can add thresholds that define metric data ranges for these three categories:

- Compliant — Metric data that is an acceptable range.
- Warning — Metric data that requires review.
- Noncompliant — Metric data that requires you to modify your model.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object. This threshold is for the `mathworks.metrics.SimulinkBlockCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
T = addThreshold(TC, 'mathworks.metrics.SimulinkBlockCount', 'Value');
```

An `slmetric.config.Threshold` object contains a default `slmetric.config.Classification` object that corresponds to the `Compliant` category. Use the `slmetric.metric.MetricRange` class to specify metric values for the `Compliant` metric range.

```
C = getClassifications(T); % default classification is Compliant
C.Range.Start = 5;
C.Range.IncludeStart = 0;
C.Range.End = 100;
C.Range.IncludeEnd = 0;
```

These values specify that a compliant range is a block count from 5 to 100. This range does not include the values 5 and 100.

Specify values for the `Warning` metric range.

```
C = addClassification(T, 'Warning');
C.Range.Start = -inf;
C.Range.IncludeStart = 0;
C.Range.End = 5;
C.Range.IncludeEnd = 1
```

These values specify that a warning is a block count between `-inf` and `5`. This range does not include `-inf`. It does include `5`.

Specify values for the `NonCompliant` metric range.

```
C = addClassification(T, 'NonCompliant');  
C.Range.Start = 100;  
C.Range.IncludeStart = 1;  
C.Range.End = inf;  
C.Range.IncludeEnd = 0;
```

These values specify that a block count greater than `100` is noncompliant. This range includes `100`. It does not include `inf`.

Use the `slmetric.config.validate` function to validate the metric ranges corresponding to the thresholds in the `slmetric.config.ThresholdConfiguration` object.

```
validate(T)
```

If the ranges are not valid, you get an error message. In this example, the ranges are valid.

Save the changes to the configuration file. Use the `slmetric.config.setActiveConfiguration` function to activate this configuration for the metric engine to use.

```
configName = 'Config.xml';  
save(CONF, 'FileName', configName);  
slmetric.config.setActiveConfiguration(fullfile(pwd, configName));
```

You can now run the Metrics Dashboard with this custom configuration on a model.

## See Also

[slmetric.config.Threshold](#) | [slmetric.config.getActiveConfiguration](#) | [slmetric.config.setActiveConfiguration](#)

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## addThreshold

**Class:** `slmetric.config.ThresholdConfiguration`

**Package:** `slmetric.config`

Create an `slmetric.config.Threshold` object

### Syntax

`TH = addThreshold(TC,metricid,thresholdproperty)`

### Description

Create an `slmetric.config.Threshold` object to apply thresholds to the data for a specific metric.

`TH = addThreshold(TC,metricid,thresholdproperty)` creates an `slmetric.config.Threshold` object.

### Input Arguments

#### **TC — Threshold configuration object**

`slmetric.config.ThresholdConfiguration` object

`slmetric.config.ThresholdConfiguration` object for which you add an `slmetric.config.Threshold` object.

Data Types: `char`

#### **metricid — Metric identifier**

character vector | string scalar

Metric identifier for model metric or custom model metric that you create.

Example: `'mathworks.metrics.SimulinkBlockCount'`

Data Types: `char`

### **thresholdproperty — Result object property**

character vector | string scalar

`slmetric.metric.Result` property to which you apply thresholds. You can apply thresholds to the `Value` and `AggregatedValue` properties.

## **Output Arguments**

### **TH — Threshold object**

`slmetric.config.Threshold` object

`slmetric.config.Threshold` object for applying thresholds to either the `Value` or `AggregatedValue` properties of an `slmetric.metric.Result` object.

## **Examples**

### **Add Thresholds to a Threshold Configuration Object**

By default, an `slmetric.config.Configuration` object holds one `slmetric.config.ThresholdConfiguration` object. Use the `getThresholdConfigurations` method to add this object to the base workspace. Use the `slmetric.config.addThreshold` method to add `slmetric.config.Threshold` objects to this `slmetric.config.ThresholdConfiguration` object.

Create an `slmetric.config.Configuration` object.

```
CONF = slmetric.config.Configuration.new('name', 'Config');
```

Get the default `slmetric.config.ThresholdConfiguration` object in `CONF`.

```
TC = getThresholdConfigurations(CONF);
```

Add an `slmetric.config.Threshold` object to the `slmetric.config.ThresholdConfiguration` object `TC`. This threshold is for the `mathworks.metrics.SubSystemCount` metric and the `Value` property of the `slmetric.metric.Results` object.

```
E = addThreshold(TC, 'mathworks.metrics.SubSystemCount', 'Value');
```

Use the `slmetric.config.Classification` and `slmetric.config.MetricRange` classes to specify threshold values corresponding to the `mathworks.metrics.SubsystemCount` metric.

### See Also

`slmetric.config.ThresholdConfiguration` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration`

### Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# getThresholds

**Class:** `slmetric.config.ThresholdConfiguration`

**Package:** `slmetric.config`

Obtain properties of threshold objects

## Syntax

```
T = getThresholds(TH,metricid)
```

## Description

Determine the properties of the threshold objects that a threshold configuration object holds. You can also use this method with the `slmetric.config.ThresholdConfiguration.removeThresholds` method to identify and remove a threshold object from a `slmetric.config.ThresholdConfiguration` object.

`T = getThresholds(TH,metricid)` creates a threshold object or an array of threshold objects.

## Input Arguments

### **TH — Threshold configuration object**

`slmetric.config.ThresholdConfiguration` object

`slmetric.config.ThresholdConfiguration` object for which you want information on the threshold objects it holds.

### **MetricID — Metric identifier**

character vector | string scalar

Metric identifier for model metric or custom model metric that you create. This argument is optional. If you do not specify a `metricID`, you get information on all thresholds that the Threshold configuration object holds.

Example: 'mathworks.metrics.SimulinkBlockCount'

Data Types: char

## Output Arguments

### T — Threshold object or array of threshold objects

character vector | string scalar | array of character vectors | array of string scalars

`slmetric.config.Threshold` object or array of `slmetric.config.Threshold` objects corresponding to the `slmetric.config.ThresholdConfiguration` object that you specify as an input.

## Examples

### Identify Threshold Objects in a Threshold Configuration Object

Use the `getThresholds` method to identify the `slmetric.config.Threshold` objects that belong to an `slmetric.config.ThresholdConfiguration` object.

For the `slmetric.config.ThresholdConfiguration` object `TC`, use the `getThresholds` method.

```
getThresholds(TC)
```

```
1×2 Threshold array with properties:
```

```
    MetricID  
    AppliesTo
```

The `slmetric.config.ThresholdConfiguration` object `TC` contains two `slmetric.config.Threshold` objects.

## See Also

`slmetric.config.ThresholdConfiguration` |  
`slmetric.config.getActiveConfiguration` |  
`slmetric.config.setActiveConfiguration`



## **Topics**

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## removeThreshold

**Class:** `slmetric.config.ThresholdConfiguration`

**Package:** `slmetric.config`

Remove threshold object from threshold configuration object

### Syntax

```
removeThreshold(TC, T)
```

### Description

Remove a threshold object from a threshold configuration object.

`removeThreshold(TC, T)` removes the `slmetric.config.Threshold` object `T` from the `slmetric.config.ThresholdConfiguration` object `TC`.

### Input Arguments

#### **TC — Threshold configuration object**

`slmetric.config.ThresholdConfiguration` object

`slmetric.config.ThresholdConfiguration` object from which you want to remove an `slmetric.config.Threshold` object.

Data Types: `char`

#### **T — Threshold object**

`slmetric.config.Threshold` | object

`slmetric.config.Threshold` object that you want to remove from an `slmetric.config.ThresholdConfiguration` object.

Data Types: `char`

## Examples

### Remove Threshold Object from a Threshold Configuration Object

Use the `getThresholds` method to identify the `slmetric.config.Threshold` objects that belong to an `slmetric.config.ThresholdConfiguration` object. Then, use the `removeThreshold` method to remove an `slmetric.config.Threshold` object.

For the `slmetric.config.ThresholdConfiguration` object `TC`, use the `getThresholds` method.

```
A = getThresholds(TC)
```

```
A =
```

```
1x2 Threshold array with properties:
```

```
    MetricID  
    AppliesTo
```

The `slmetric.config.ThresholdConfiguration` object `TC` contains two `slmetric.config.Threshold` objects.

Identify the `slmetric.config.Threshold` object that you want to remove from the `slmetric.config.ThresholdConfiguration` object.

```
A.MetricID
```

```
ans =
```

```
    'mathworks.metrics.SimulinkBlockCount'
```

```
ans =
```

```
    'mathworks.metricchecks.SubSystemCount'
```

Remove the second element of the array that corresponds to the `mathworks.metricchecks.SubSystemCount` metric.

```
removeThreshold(TC,A(2))
```

The `slmetric.ThresholdConfiguration` object now contains one `slmetric.config.Threshold` object corresponding to the `mathworks.metricchecks.SubSystemCount` metric.

```
getThresholds(TC)
```

```
ans =
```

```
Threshold with properties:
```

```
    MetricID: 'mathworks.metrics.SimulinkBlockCount'  
    AppliesTo: 'Value'
```

### See Also

```
slmetric.config.ThresholdConfiguration |  
slmetric.config.getActiveConfiguration |  
slmetric.config.setActiveConfiguration
```

### Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.config.getActiveConfiguration

**Package:** slmetric.config

Obtain file path and name of XML file containing active Metrics Dashboard custom configuration

## Syntax

Path = slmetric.config.getActiveConfiguration

## Description

Path = slmetric.config.getActiveConfiguration returns the file path and name of the active Metrics Dashboard custom configuration file.

## Examples

### Get Default Metrics Dashboard Configuration

At the MATLAB command line, enter this command to get the active Metrics Dashboard configuration:

```
slmetric.config.getActiveConfiguration();
```

## Output Arguments

### Path — File path to XML file

character vector | string scalar

Full file path to folder containing XML file, which contains the active Metrics Dashboard custom configuration.

---

**Note** Passing an empty string to the `slmetric.config.setActionConfiguration` function (that is, `slmetric.config.setActiveConfiguration('')`), resets the configuration to the default, which is the shipping configuration. If you then enter the `slmetric.config.getActiveConfiguration` method, the method returns an empty array.

---

Data Types: `char`

### See Also

`slmetric.config.Configuration` | `slmetric.config.setActiveConfiguration`

### External Websites

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.config.setActiveConfiguration

**Package:** slmetric.config

Activate custom configuration for metric engine to use

## Syntax

```
slmetric.config.setActiveConfiguration(fullfile)
```

## Description

`slmetric.config.setActiveConfiguration(fullfile)` sets the custom configuration as the default configuration. When you collect metric data that uses the Metrics Dashboard or the `slmetric.Engine.execute` method, the metric engine uses this custom configuration.

---

**Note** Passing an empty string to this function (that is, `slmetric.config.setActiveConfiguration('')`), resets the configuration to the default, which is shipping configuration.

---

## Examples

### Activate Custom Configuration

To set the active metric configuration, at the MATLAB command line:

```
slmetric.config.setActiveConfiguration('C:\temp\MyConfig.xml');
```

## Input Arguments

**fullfile** — File path to XML file

character vector | string scalar

Full file path to folder containing the XML file, which contains Metrics Dashboard custom configurations.

Example: 'C:\temp\MyConfig.xml'

Data Types: char

### See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`

### Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



# Advisor.component.Component class

**Package:** Advisor.component

Create component for metric analysis

## Description

Model component used for metric analysis. When you define a custom model metric, the component object defines the component for metric analysis.

## Construction

`component_obj = Advisor.component.Component` creates a model component object.

## Properties

### **ID — Component ID**

character vector

Component identifier. This property is read/write.

### **Type — Component type**

enum

Component type, as specified by `Advisor.component.Types`. This property is read/write.

### **Name — Component name**

character vector

Model component name. This property is read/write.

### **IsLinked — Specifies if the component is linked to a library**

logical

IsLinked is true if the component is linked to a library. Components of type Model, ModelBlock, ProtectedModel cannot be linked. For these properties, the IsLinked is always true.

## Methods

getPath                      Retrieve component path

## See Also

Advisor.component.Types | slmetric.metric.Metric

## Topics

“Create a Custom Model Metric for Nonvirtual Block Count”  
“Model Metrics” on page 2-419

**Introduced in R2016a**

## getPath

**Class:** `Advisor.component.Component`

**Package:** `Advisor.component`

Retrieve component path

## Syntax

```
path = getPath(component)
```

## Description

`path = getPath(component)` retrieves the path to the component.

## Input Arguments

**component** — **Component**

`Advisor.component.Component` model object

Constructed `Advisor.component.Component` model object.

## Output Arguments

**path** — **Model component path**

character vector

Model component path, specified as a character vector.

## See Also

`Advisor.component.Types`

**Introduced in R2016a**

# Advisor.component.Types class

**Package:** `Advisor.component`

Create enum class specifying component type

## Description

Create an enumeration `Advisor.component.Types` class to specify the model component type.

## Construction

`enum_comp_type = Advisor.component.Types.Model` creates an enumeration of component type `Model`. The following table lists the component types.

Type	Description
<code>Model</code>	Simulink block diagram.
<code>LibraryBlock</code>	Library linked block.
<code>MFile</code>	MATLAB code file.
<code>ProtectedModel</code>	Protect Simulink block diagram.
<code>SubSystem</code>	Simulink subsystem block.
<code>Chart</code>	Stateflow® chart or Stateflow block.
<code>MATLABFunction</code>	MATLAB function block.

## See Also

`Advisor.component.Component` | `slmetric.metric.Metric`

## Topics

“Create a Custom Model Metric for Nonvirtual Block Count”

“Model Metrics” on page 2-419

**Introduced in R2016a**

# ModelAdvisor.Action class

**Package:** ModelAdvisor

Add actions to custom checks

## Description

Instances of this class define actions you take when the Model Advisor checks do not pass. Users access actions by clicking the **Action** button that you define in the Model Advisor window.

## Construction

ModelAdvisor.Action	Add actions to custom checks
---------------------	------------------------------

## Methods

setCallbackFcn	Specify action callback function
----------------	----------------------------------

## Properties

Description	Message in <b>Action</b> box
Name	Action button label

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
myAction.Name='Fix block fonts';
myAction.Description=...
    'Click the button to update all blocks with specified font';
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”



# ModelAdvisor.Action

**Class:** ModelAdvisor.Action

**Package:** ModelAdvisor

Add actions to custom checks

## Syntax

```
action_obj = ModelAdvisor.Action
```

## Description

`action_obj = ModelAdvisor.Action` creates a handle to an action object.

---

### Note

- Include an action definition in a check definition.
  - Each check can contain only one action.
- 

## Examples

```
% define action (fix) operation  
myAction = ModelAdvisor.Action;
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.Check class

**Package:** ModelAdvisor

Create custom checks

### Description

The `ModelAdvisor.Check` class creates a Model Advisor check object. Checks must have an associated `ModelAdvisor.Task` object to be displayed in the Model Advisor tree.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product > Simulink** folder and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When you use checks in task definitions, the following rules apply:

- If you define the properties of the check in the check definition and the task definition, the task definition takes precedence. The Model Advisor displays the information contained in the task definition. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property and in the check definition using the `ModelAdvisor.Check.Title` property, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.
- If you define the properties of the check in the check definition but not the task definition, the task uses the properties from the check. For example, if you define the name of the check in the check definition using the `ModelAdvisor.Check.Title` property, and you register the check using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Check.Title`.
- If you define the properties of the check in the task definition but not the check definition, the Model Advisor displays the information as long as you register the task with the Model Advisor instead of the check. For example, if you define the name of the check in the task definition using the `ModelAdvisor.Task.DisplayName` property instead of the `ModelAdvisor.Check.Title` property, and you register the check using a task definition, the Model Advisor displays the information provided in `ModelAdvisor.Task.DisplayName`.

## Construction

ModelAdvisor.Check

Create custom checks

## Methods

getID

Return check identifier

setAction

Specify action for check

setCallbackFcn

Specify callback function for check

setInputParameters

Specify input parameters for check

setInputParametersLayoutGrid

Specify layout grid for input parameters

setResultDetails

Associates result details with a check object

## Properties

CallbackContext	Specify when to run check
CallbackHandle	Callback function handle for check
CallbackStyle	Callback function type
EmitInputParametersToReport	Display check input parameters in the Model Advisor report
Enable	Indicate whether user can enable or disable check
ID	Identifier for check
LicenseName	Product license names required to display and run check
ListViewVisible	Status of <b>Explore Result</b> button
Result	Results cell array
SupportExclusion	Set to support exclusions
SupportLibrary	Set to support library models
Title	Name of check
TitleTips	Description of check
Value	Status of check
Visible	Indicate to display or hide check
ResultDetails	Result details in a cell array

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
```

## See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## ModelAdvisor.Check

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Create custom checks

### Syntax

```
check_obj = ModelAdvisor.Check(check_ID)
```

### Description

`check_obj = ModelAdvisor.Check(check_ID)` creates a check object, `check_obj`, and assigns it a unique identifier, `check_ID`. `check_ID` must remain constant. To display checks in the Model Advisor tree, checks must have an associated `ModelAdvisor.Task` or `ModelAdvisor.Root` object.

---

**Note** You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

---

### Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
```

### See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## ModelAdvisor.FactoryGroup class

**Package:** ModelAdvisor

Define subfolder in **By Task** folder

### Description

The ModelAdvisor.FactoryGroup class defines a new subfolder to add to the **By Task** folder.

### Construction

ModelAdvisor.FactoryGroup	Define subfolder in <b>By Task</b> folder
---------------------------	---

### Methods

addCheck	Add check to folder
----------	---------------------

### Properties

Description	Description of folder
DisplayName	Name of folder
ID	Identifier for folder
MAObj	Model Advisor object

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.



## Examples

```
% --- sample factory group  
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.FactoryGroup

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Define subfolder in **By Task** folder

### Syntax

```
fg_obj = ModelAdvisor.FactoryGroup(fg_ID)
```

### Description

`fg_obj = ModelAdvisor.FactoryGroup(fg_ID)` creates a handle to a factory group object, `fg_obj`, and assigns it a unique identifier, `fg_ID`. `fg_ID` must remain constant.

### Examples

```
% --- sample factory group  
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

# ModelAdvisor.FormatTemplate class

**Package:** ModelAdvisor

Template for formatting Model Advisor analysis results

## Description

Use the `ModelAdvisor.FormatTemplate` class to format the result of a check in the analysis result pane of the Model Advisor for a uniform look and feel among the checks you create. There are two formats for the analysis result:

- Table
- List

## Construction

`ModelAdvisor.FormatTemplate` Construct template object for formatting Model Advisor analysis results

## Methods

<code>addRow</code>	Add row to table
<code>setCheckText</code>	Add description of check to result
<code>setColTitles</code>	Add column titles to table
<code>setInformation</code>	Add description of subcheck to result
<code>setListObj</code>	Add list of hyperlinks to model objects
<code>setRecAction</code>	Add Recommended Action section and text
<code>setRefLink</code>	Add See Also section and links
<code>setSubBar</code>	Add line between subcheck results
<code>setSubResultStatus</code>	Add status to the check or subcheck result
<code>setSubResultStatusText</code>	Add text below status in result
<code>setSubTitle</code>	Add title for subcheck in result
<code>setTableInfo</code>	Add data to table
<code>setTableTitle</code>	Add title to table

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

The following code creates two template objects, `ft1` and `ft2`, and uses them to format the result of running the check in a table and a list. The result identifies the blocks in the model. The graphics following the code display the output as it appears in the Model Advisor when the check passes and fails.

```
function sl_customization(cm)

% register custom checks
cm.addModelAdvisorCheckFcn(@defineModelAdvisorChecks);

% register custom factory group
cm.addModelAdvisorTaskFcn(@defineModelAdvisorTasks);
```

```

% -----
% defines Model Advisor Checks
% -----
function defineModelAdvisorChecks

% Define and register a sample check
rec = ModelAdvisor.Check('mathworks.example.SampleStyleOne');
rec.Title = 'Sample check for Model Advisor using the ModelAdvisor.FormatTemplate';
setCallbackFcn(rec, @SampleStyleOneCallback,'None','StyleOne');

mdladvRoot = ModelAdvisor.Root;
mdladvRoot.register(rec);

% -----
% defines Model Advisor Tasks
% -----
function defineModelAdvisorTasks
mdladvRoot = ModelAdvisor.Root;

% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='My Group 1';
rec.Description='Demo Factory Group';
rec.addCheck('mathworks.example.SampleStyleOne');
mdladvRoot.publish(rec); % publish inside By Group list

% -----
% Sample Check With Subchecks Callback Function
% -----
function ResultDescription = SampleStyleOneCallback(system)
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system); % get object

% Initialize variables
ResultDescription={};
ResultStatus = false; % Default check status is 'Warning'
mdladvObj.setCheckResultStatus(ResultStatus);

% Create FormatTemplate object for first subcheck, specify table format
ft1 = ModelAdvisor.FormatTemplate('TableTemplate');

% Add information describing the overall check
setCheckText(ft1, ['Find and report all blocks in the model. ...
'(setCheckText method - Description of what the check reviews)']);

% Add information describing the subcheck
setSubTitle(ft1, 'Table of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft1, ['Find and report all blocks in a table. ...
'(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
setRefLink(ft1, {'Standard 1 reference (setRefLink method)',
{'Standard 2 reference (setRefLink method)'}]);

```

```
% Add information to the table
setTableTitle(ft1, {'Blocks in the Model (setTableTitle method)'});
setColTitles(ft1, {'Index (setColTitles method)',
    'Block Name (setColTitles method)'});

% Perform the check actions
allBlocks = find_system(system);
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft1, 'Warn');
    setSubResultStatusText(ft1, ['The model does not contain blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft1, {'Add blocks to the model. '...
        '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
else
    % Add status for subcheck
    setSubResultStatus(ft1, 'Pass');
    setSubResultStatusText(ft1, ['The model contains blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    for inx = 2 : length(allBlocks)
        % Add information to the table
        addRow(ft1, {inx-1,allBlocks(inx)});
    end
    ResultStatus = true;
end

% Pass table template object for subcheck to Model Advisor
ResultDescription{end+1} = ft1;

% Create FormatTemplate object for second subcheck, specify list format
ft2 = ModelAdvisor.FormatTemplate('ListTemplate');

% Add information describing the subcheck
setSubTitle(ft2, 'List of Blocks (setSubTitle method - Title of the subcheck)');
setInformation(ft2, ['Find and report all blocks in a list. '...
    '(setInformation method - Description of what the subcheck reviews)']);

% Add See Also section for references to standards
setRefLink(ft2, {'Standard 1 reference (setRefLink method)',
    {'Standard 2 reference (setRefLink method)'}));

% Last subcheck, suppress line
setSubBar(ft2, false);

% Perform the subcheck actions
if length(find_system(system)) == 1
    % Add status for subcheck
    setSubResultStatus(ft2, 'Warn');
    setSubResultStatusText(ft2, ['The model does not contain blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    setRecAction(ft2, {'Add blocks to the model. '...
        '(setRecAction method - Description of how to fix the problem)'});
    ResultStatus = false;
end
```

```
else
    % Add status for subcheck
    setSubResultStatus(ft2, 'Pass');
    setSubResultStatusText(ft2, ['The model contains blocks. '...
        '(setSubResultStatusText method - Description of result status)']);
    % Add information to the list
    setListObj(ft2, allBlocks);
end

% Pass list template object for the subcheck to Model Advisor
ResultDescription{end+1} = ft2;
% Set overall check status
mdladvObj.setCheckResultStatus(ResultStatus);
```

The following graphic displays the output as it appears in the Model Advisor when the check passes.

Result:  Passed

### Table of Blocks (setSubTitle method - Title of the subcheck)

Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

#### See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

#### Passed

The model contains blocks. (setSubResultStatusText method - Description of result status)

Blocks in the Model (setTableTitle method)

Index (setColTitles method)	Block Name (setColTitles method)
1	<a href="#">model/Constant</a>
2	<a href="#">model/Constant1</a>
3	<a href="#">model/Gain</a>
4	<a href="#">model/Product</a>
5	<a href="#">model/Out1</a>

---

### List of Blocks (setSubTitle method - Title of the subcheck)

Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

#### See Also

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

#### Passed

The model contains blocks. (setSubResultStatusText method - Description of result status)

- [model](#)
- [model/Constant](#)
- [model/Constant1](#)
- [model/Gain](#)
- [model/Product](#)
- [model/Out1](#)



The following graphic displays the output as it appears in the Model Advisor when the check fails.

Result:  Warning

Find and report all blocks in the model. (setCheckText method - Description of what the check reviews)

**Table of Blocks (setSubTitle method - Title of the subcheck)**

Find and report all blocks in a table. (setInformation method - Description of what the subcheck reviews)

**See Also**

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Warning**

The model does not contain blocks. (setSubResultStatusText method - Description of result status)

**Recommended Action**

Add blocks to the model.

(setRecAction method - Description of how to fix the problem)

---

**List of Blocks (setSubTitle method - Title of the subcheck)**

Find and report all blocks in a list. (setInformation method - Description of what the subcheck reviews)

**See Also**

- Standard 1 reference (setRefLink method)
- Standard 2 reference (setRefLink method)

**Warning**

The model does not contain blocks. (setSubResultStatusText method - Description of result status)

**Recommended Action**

Add blocks to the model.

(setRecAction method - Description of how to fix the problem)

## **Alternatives**

Use the Model Advisor Formatting API to format check analysis results. However, use the `ModelAdvisor.FormatTemplate` class for a uniform look and feel among the checks you create.

## **See Also**

[“Model Advisor Customization”](#)

## **Topics**

[“Create Model Advisor Checks”](#)

[“Format Check Results”](#)

# ModelAdvisor.FormatTemplate

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Construct template object for formatting Model Advisor analysis results

## Syntax

```
obj = ModelAdvisor.FormatTemplate('type')
```

## Description

*obj* = ModelAdvisor.FormatTemplate('type') creates a handle, *obj*, to an object of the ModelAdvisor.FormatTemplate class. *type* is a character vector identifying the format type of the template, either list or table. Valid values are ListTemplate and TableTemplate.

You must return the result object to the Model Advisor to display the formatted result in the analysis result pane.

---

**Note** Use the ModelAdvisor.FormatTemplate class in check callbacks.

---

## Examples

Create a template object, *ft*, and use it to create a list template:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”

“Format Check Results”

# ModelAdvisor.Group class

**Package:** ModelAdvisor

Define custom folder

## Description

The `ModelAdvisor.Group` class defines a folder that is displayed in the Model Advisor tree. Use folders to consolidate checks by functionality or usage.

## Construction

`ModelAdvisor.Group`

Define custom folder

## Methods

`addGroup`

Add subfolder to folder

`addProcedure`

Add procedure to folder

`addTask`

Add task to folder

## Properties

`Description`

Description of folder

`DisplayName`

Name of folder

`ID`

Identifier for folder

`MAObj`

Model Advisor object

## **Copy Semantics**

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## **See Also**

*“Model Advisor Customization”*

## **Topics**

*“Create Model Advisor Checks”*

# ModelAdvisor.Group

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Define custom folder

## Syntax

```
group_obj = ModelAdvisor.Group(group_ID)
```

## Description

`group_obj = ModelAdvisor.Group(group_ID)` creates a handle to a group object, `group_obj`, and assigns it a unique identifier, `group_ID`. `group_ID` must remain constant.

## Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.Image class

**Package:** ModelAdvisor

Include image in Model Advisor output

### Description

The ModelAdvisor.Image class adds an image to the Model Advisor output.

### Construction

ModelAdvisor.Image	Include image in Model Advisor output
--------------------	---------------------------------------

### Methods

setHyperlink	Specify hyperlink location
setImageSource	Specify image location

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

“Format Check Results”



# ModelAdvisor.Image

**Class:** ModelAdvisor.Image

**Package:** ModelAdvisor

Include image in Model Advisor output

## Syntax

```
object = ModelAdvisor.Image
```

## Description

`object = ModelAdvisor.Image` creates a handle to an image object, `object`, that the Model Advisor displays in the output. The Model Advisor supports many image formats, including, but not limited to, JPEG, BMP, and GIF.

## Examples

```
image_obj = ModelAdvisor.Image;
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## ModelAdvisor.InputParameter class

**Package:** ModelAdvisor

Add input parameters to custom checks

### Description

Instances of the `ModelAdvisor.InputParameter` class specify the input parameters a custom check uses in analyzing the model. Access input parameters in the Model Advisor window.

### Construction

`ModelAdvisor.InputParameter`      Add input parameters to custom checks

### Methods

`setColSpan`      Specify number of columns for input parameter  
`setRowSpan`      Specify rows for input parameter

### Properties

Description	Description of input parameter
Entries	Drop-down list entries
Name	Input parameter name
Type	Input parameter type
Value	Value of input parameter

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## See Also

*"Model Advisor Customization"*

## Topics

*"Create Model Advisor Checks"*

## ModelAdvisor.InputParameter

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Add input parameters to custom checks

### Syntax

```
input_param = ModelAdvisor.InputParameter
```

### Description

`input_param = ModelAdvisor.InputParameter` creates a handle to an input parameter object, `input_param`.

---

**Note** You must include input parameter definitions in a check definition.

---

### Examples

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvndemo_mdadv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.setInputParametersLayoutGrid([3 2]);
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
inputParam1.Description = 'sample tooltip';
inputParam1.setRowSpan([1 1]);
inputParam1.setColSpan([1 1]);
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
```

```
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
inputParam2.setRowSpan([2 2]);
inputParam2.setColSpan([1 1]);
inputParam3 = ModelAdvisor.InputParameter;
inputParam3.Name='Valid font';
inputParam3.Type='Combobox';
inputParam3.Description='sample tooltip';
inputParam3.Entries={'Arial', 'Arial Black'};
inputParam3.setRowSpan([2 2]);
inputParam3.setColSpan([2 2]);
rec.setInputParameters({inputParam1,inputParam2,inputParam3});
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## **ModelAdvisor.LineBreak class**

**Package:** ModelAdvisor

Insert line break

### **Description**

Use instances of the `ModelAdvisor.LineBreak` class to insert line breaks in the Model Advisor outputs.

### **Construction**

`ModelAdvisor.LineBreak`

Insert line break

### **Copy Semantics**

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

### **See Also**

“Model Advisor Customization”

### **Topics**

“Create Model Advisor Checks”

“Format Check Results”

# ModelAdvisor.LineBreak

**Class:** ModelAdvisor.LineBreak

**Package:** ModelAdvisor

Insert line break

## Syntax

ModelAdvisor.LineBreak

## Description

ModelAdvisor.LineBreak inserts a line break into the Model Advisor output.

## Examples

Add a line break between two lines of text:

```
result = ModelAdvisor.Paragraph;  
addItem(result, [resultText1 ModelAdvisor.LineBreak resultText2]);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## ModelAdvisor.List class

**Package:** ModelAdvisor

Create list class

### Description

Use instances of the ModelAdvisor.List class to create list-formatted outputs.

### Construction

ModelAdvisor.List

Create list class

### Methods

addItem

Add item to list

setType

Specify list type

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

“Format Check Results”



# ModelAdvisor.List

**Class:** ModelAdvisor.List

**Package:** ModelAdvisor

Create list class

## Syntax

```
list = ModelAdvisor.List
```

## Description

`list = ModelAdvisor.List` creates a list object, `list`.

## Examples

```
subList = ModelAdvisor.List();
setType(subList, 'numbered')
addItem(subList, ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
addItem(subList, ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## ModelAdvisor.ListViewParameter class

**Package:** ModelAdvisor

Add list view parameters to custom checks

### Description

The Model Advisor uses list view parameters to populate the Model Advisor Result Explorer. Access the information in list views by clicking **Explore Result** in the Model Advisor window.

### Construction

ModelAdvisor.ListViewParameter      Add list view parameters to custom checks

### Properties

Attributes	Attributes to display in Model Advisor Report Explorer
Data	Objects in Model Advisor Result Explorer
Name	Drop-down list entry

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvndemo_mdadv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

```
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system);
mdladvObj.setCheckResultStatus(true);

% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
myLVParam.Attributes = {'FontName'}; % name is default property
mdladvObj.setListViewParameters({myLVParam});
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.ListViewParameter

**Class:** ModelAdvisor.ListViewParameter

**Package:** ModelAdvisor

Add list view parameters to custom checks

### Syntax

```
lv_param = ModelAdvisor.ListViewParameter
```

### Description

lv\_param = ModelAdvisor.ListViewParameter defines a list view, lv\_param.

---

**Note** Include list view parameter definitions in a check definition.

---

### See Also

“Model Advisor Customization”

### Topics

“Define Model Advisor Result Explorer Views”

“Create Model Advisor Checks”

“Batch-Fix Warnings or Failures” (Simulink)

“Customization Example”

getListViewParameters

setListViewParameters

# ModelAdvisor.lookupCheckID

**Package:** ModelAdvisor

Look up Model Advisor check ID

## Syntax

```
NewID = ModelAdvisor.lookupCheckID('OldCheckID')
```

## Description

`NewID = ModelAdvisor.lookupCheckID('OldCheckID')` returns the check ID of the check specified by `OldCheckID`.

## Input Arguments

### **OldCheckID**

`OldCheckID` is the prior ID of a check.

## Output Arguments

### **NewID**

Check ID that corresponds to the previous check ID identified by `OldCheckID`.

## Examples

The check ID for Check for root Outports with missing range definitions prior to Release 2018b was `mathworks.iec61508.OutportRange`. Use the `ModelAdvisor.lookupCheckID` function to return the current check ID.

```
>> NewID = ModelAdvisor.lookupCheckID('mathworks.iec61508.OutputRange')  
  
NewID =  
'mathworks.hism.hisl_0026'
```

## Alternatives

“Archive and View Results”

## See Also

ModelAdvisor.run

## Topics

“Archive and View Results”

**Introduced in R2010b**

# ModelAdvisor.Paragraph class

**Package:** ModelAdvisor

Create and format paragraph

## Description

The ModelAdvisor.Paragraph class creates and formats a paragraph object.

## Construction

ModelAdvisor.Paragraph

Create and format paragraph

## Methods

addItem            Add item to paragraph

setAlign           Specify paragraph alignment

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

```
% Check Simulation optimization setting
ResultDescription = ModelAdvisor.Paragraph(['Check Simulation '...
'optimization settings:']);
```

## **See Also**

*“Model Advisor Customization”*

## **Topics**

*“Create Model Advisor Checks”*

*“Format Check Results”*



# ModelAdvisor.Paragraph

**Class:** ModelAdvisor.Paragraph

**Package:** ModelAdvisor

Create and format paragraph

## Syntax

```
para_obj = ModelAdvisor.Paragraph
```

## Description

`para_obj = ModelAdvisor.Paragraph` defines a paragraph object `para_obj`.

## Examples

```
% Check Simulation optimization setting  
ResultDescription = ModelAdvisor.Paragraph(['Check Simulation '...  
'optimization settings:']);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.Procedure class

**Package:** ModelAdvisor

Define custom procedures

### Description

The `ModelAdvisor.Procedure` class defines a procedure that is displayed in the Model Advisor tree. Use procedures to organize additional procedures or checks by functionality or usage.

### Construction

`ModelAdvisor.Procedure`

Define custom procedures

### Properties

#### Description

Provides information about the procedure. Details about the procedure are displayed in the right pane of the Model Advisor.

**Default:** '' (empty character vector)

#### Name

Specifies the name of the procedure that is displayed in the Model Advisor.

**Default:** '' (empty character vector)

#### ID

Specifies a permanent, unique identifier for the procedure.

---

**Note**

- You must specify this field.
  - The value of ID must remain constant.
  - The Model Advisor generates an error if ID is not unique.
  - Procedure definitions must refer to other procedures by ID.
- 

**MAObj**

Specifies a handle to the current Model Advisor object.

**Methods**

addProcedure	Add subprocedure to procedure
addTask	Add task to procedure

**Copy Semantics**

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

**See Also**

“Model Advisor Customization”

**Topics**

“Create Procedures”

“Create Procedural-Based Configurations”

“Create Model Advisor Checks”

## ModelAdvisor.Procedure

**Class:** ModelAdvisor.Procedure

**Package:** ModelAdvisor

Define custom procedures

### Syntax

```
procedure_obj = ModelAdvisor.Procedure(procedure_ID)
```

### Description

`procedure_obj = ModelAdvisor.Procedure(procedure_ID)` creates a handle to a procedure object, `procedure_obj`, and assigns it a unique identifier, `procedure_ID`. `procedure_ID` must remain constant.

### Examples

```
MAP = ModelAdvisor.Procedure('com.mathworks.sample.ProcedureSample');
```

### See Also

“Model Advisor Customization”

### Topics

“Create Procedures”

“Create Procedural-Based Configurations”

“Create Model Advisor Checks”

# ModelAdvisor.Root class

**Package:** ModelAdvisor

Identify root node

## Description

The ModelAdvisor.Root class returns the root object.

## Construction

ModelAdvisor.Root

Identify root node

## Methods

publish	Publish object in Model Advisor root
register	Register object in Model Advisor root

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.Root

**Class:** ModelAdvisor.Root

**Package:** ModelAdvisor

Identify root node

### Syntax

```
root_obj = ModelAdvisor.Root
```

### Description

`root_obj = ModelAdvisor.Root` creates a handle to the root object, `root_obj`.

### Examples

```
mdladvRoot = ModelAdvisor.Root;
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

# ModelAdvisor.run

**Package:** ModelAdvisor

Run Model Advisor checks on systems

## Syntax

```
SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList,Name,Value)  
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',  
FileName,Name,Value)
```

## Description

`SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList,Name,Value)` runs the Model Advisor on the systems provided by `SysList` with additional options specified by one or more optional `Name,Value` pair arguments. `CheckIDList` contains cell array of check IDs to run.

`SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',FileName,Name,Value)` runs the Model Advisor on the systems provided by `SysList`. The list of checks to run is specified using a Model Advisor configuration file, specified by `FileName`.

## Input Arguments

### SysList

Cell array of systems to run.

### CheckIDList

Cell array of check IDs to run. For details on how to find check IDs, see “Find Model Advisor Check IDs” (Simulink).

`CheckIDList` optionally can include input parameters for specific checks using the following syntax; `{'CheckID','InputParam',{'IP','IPV'}}`, where `IP` is the input

parameter name and IPV is the corresponding input parameter value. You can specify several input parameter name and value pair arguments in any order as IP1, IPV1, ..., IPN, IPVN.

## **FileName**

Name of the Model Advisor configuration file. For details on creating a configuration file, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”.

## **Name-Value Pair Arguments**

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

## **DisplayResults**

Setting DisplayResults to 'Summary' displays a summary of the system results in the Command Window. Setting DisplayResults to 'Details' displays the following in the Command Window:

- Which system the Model Advisor is checking while the run is in progress.
- For each system, the pass and fail results of each check.
- A summary of the system results.

Setting DisplayResults to 'None' displays no information in the Command Window.

**Default:** 'Summary'

## **Force**

Setting Force to 'On' removes existing modeladvisor/system folders. Setting Force to 'Off' prompts you before removing existing modeladvisor/system folders.

**Default:** 'Off'

## **ParallelMode**

Setting ParallelMode to 'On' runs the Model Advisor in parallel mode if you have a Parallel Computing Toolbox license and a multicore machine. The Parallel Computing



Toolbox does not support 32-bit Windows® machines. Each parallel process runs checks on one model at a time. In parallel mode, load the model data from the model workspace or data dictionary. The Model Advisor in parallel mode does not support model data in the base workspace. For an example, see “Create a Function for Checking Multiple Systems in Parallel”.

**Default:** 'Off'

### **TempDir**

Setting TempDir to 'On' runs the Model Advisor from a temporary working folder, to avoid concurrency issues when running using a parallel pool. For more information, see “Resolving Data Concurrency Issues” (Simulink). Setting TempDir to 'Off' runs the Model Advisor in the current working folder.

**Default:** 'Off'

### **ShowExclusions**

Setting ShowExclusions to 'On' lists Model Advisor check exclusions in the report. Setting ShowExclusions to 'Off' does not list Model Advisor check exclusion in the report.

**Default:** 'On'

## **Output Arguments**

### **SysResultObjArray**

Cell array of ModelAdvisor.SystemResult objects, one for each model specified in SysList. Each ModelAdvisor.SystemResult object contains an array of CheckResultObj objects. Save SysResultObjArray to review results at a later time without having to rerun the Model Advisor (see “Save and Load Process for Objects” (MATLAB)).

### **CheckResultObj**

Array of ModelAdvisor.CheckResult objects, one for each check that runs.

## Examples

Runs the Model Advisor checks **Check model diagnostic parameters** and **Check for fully defined interface** on the `sldemo_auto_climatecontrol/Heater Control` and `sldemo_auto_climatecontrol/AC Control` subsystems:

```
% Create list of checks and models to run.
CheckIDList ={'mathworks.maab.jc_0021',...
             'mathworks.iec61508.RootLevelImports'};
SysList={'sldemo_auto_climatecontrol/Heater Control',...
        'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,CheckIDList);
```

Runs the Model Advisor configuration file `slvndemo_mdldv_config.mat` on the `sldemo_auto_climatecontrol/Heater Control` and `sldemo_auto_climatecontrol/AC Control` subsystems:

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvndemo_mdldv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
        'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);
```

## Tips

- If you have a Parallel Computing Toolbox™ license and a multicore machine, Model Advisor can run on multiple systems in parallel. You can run the Model Advisor in parallel mode by using `ModelAdvisor.run` with `'ParallelMode'` set to `'On'`. By default, `'ParallelMode'` is set to `'Off'`. When you use `ModelAdvisor.run` with `'ParallelMode'` set to `'On'`, MATLAB automatically creates a parallel pool.

## Alternatives

- Use the Model Advisor GUI to run each system, one at a time.
- Create a script or function using the `Simulink.ModelAdvisor` class to run each system, one at a time.

## Extended Capabilities

### Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set 'ParallelMode' to 'On'.

### See Also

`ModelAdvisor.lookupCheckID` | `ModelAdvisor.summaryReport` | `view` | `viewReport`

### Topics

*"Checking Systems Programmatically"*

*"Create a Function for Checking Multiple Systems in Parallel"*

*"Automate Model Advisor Check Execution"*

*"Organize Checks and Folders Using the Model Advisor Configuration Editor"*

*"Save and Load Process for Objects" (MATLAB)*

*"Find Model Advisor Check IDs" (Simulink)*

### Introduced in R2010b

## ModelAdvisor.summaryReport

**Package:** ModelAdvisor

Open Model Advisor Command-Line Summary report

### Syntax

```
ModelAdvisor.summaryReport(SysResultObjArray)
```

### Description

`ModelAdvisor.summaryReport(SysResultObjArray)` opens the Model Advisor Command-Line Summary report in a web browser. `SysResultObjArray` is a cell array of `ModelAdvisor.SystemResult` objects returned by `ModelAdvisor.run`.

### Input Arguments

#### `SysResultObjArray`

Cell array of `ModelAdvisor.SystemResult` objects returned by `ModelAdvisor.run`.

### Examples

Opens the Model Advisor Command-Line Summary report after running the Model Advisor:

```
% Identify Model Advisor configuration file.  
% Create list of models to run.  
fileName = 'slvndemo_mdldv_config.mat';  
SysList={'sldemo_auto_climatecontrol/Heater Control',...  
        'sldemo_auto_climatecontrol/AC Control'};  
  
% Run the Model Advisor.  
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);
```

`% Open the Model Advisor Command-Line Summary report.  
ModelAdvisor.summaryReport(SysResultObjArray)`

## Alternatives

“View Results in Model Advisor Command-Line Summary Report”

## See Also

`ModelAdvisor.run | view | viewReport`

## Topics

“Checking Systems Programmatically”

“Create a Function for Checking Multiple Systems in Parallel”

“Automate Model Advisor Check Execution”

“Archive and View Model Advisor Run Results”

## Introduced in R2010b

## **ModelAdvisor.Table class**

**Package:** ModelAdvisor

Create table

### **Description**

Instances of the `ModelAdvisor.Table` class create and format a table. Specify the number of rows and columns in a table, excluding the table title and table heading row.

### **Construction**

`ModelAdvisor.Table`

Create table

## Methods

getEntry	Get table cell contents
setColHeading	Specify table column title
setColHeadingAlign	Specify column title alignment
setColHeadingValign	Specify column title vertical alignment
setColWidth	Specify column widths
setEntries	Set contents of table
setEntry	Add cell to table
setEntryAlign	Specify table cell alignment
setEntryValign	Specify table cell vertical alignment
setHeading	Specify table title
setHeadingAlign	Specify table title alignment
setRowHeading	Specify table row title
setRowHeadingAlign	Specify table row title alignment
setRowHeadingValign	Specify table row title vertical alignment

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## ModelAdvisor.Table

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Create table

### Syntax

```
table = ModelAdvisor.Table(row, column)
```

### Description

`table = ModelAdvisor.Table(row, column)` creates a table object (`table`). The Model Advisor displays the table object containing the number of rows (`row`) and columns (`column`) that you specify.

### Examples

#### Create two table objects

Create two table objects, `table1` and `table2`. The Model Advisor displays `table1` in the results as a table with one row and one column. The Model Advisor display `table2` in the results as a table with two rows and three columns.

```
table1 = ModelAdvisor.Table(1,1);  
table2 = ModelAdvisor.Table(2,3);
```

#### Create table with five rows and five columns

Create a table with five rows and five columns containing randomly generated numbers.

Use the following MATLAB code in a callback function. The Model Advisor displays `table1` in the results.



---

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	81472.3686	9754.0405	15761.3082	14188.6339	65574.0699
Row 2	90579.1937	27849.8219	97059.2782	42176.1283	3571.1679
Row 3	12698.6816	54688.1519	Example Text	91573.5525	84912.9306
Row 4	91337.5856	95750.6835	48537.5649	79220.733	93399.3248
Row 5	63235.9246	96488.8535	80028.0469	95949.2426	67873.5155

## See Also

ModelAdvisor.Text | setColHeading | setColHeadingAlign | setColWidth | setEntry | setEntryAlign | setRowHeading

## Topics

“Create Callback Functions and Results”

## ModelAdvisor.Task class

**Package:** ModelAdvisor

Define custom tasks

### Description

The `ModelAdvisor.Task` class is a wrapper for a check so that you can access the check with the Model Advisor.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** is displayed in the **By Product > Simulink** folder and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

### Construction

`ModelAdvisor.Task`

Define custom tasks

### Methods

`setCheck`

Specify check used in task

## Properties

Description	Description of task
DisplayName	Name of task
Enable	Indicate if user can enable and disable task
ID	Identifier for task
LicenseName	Product license names required to display and run task
MAObj	Model Advisor object
Value	Status of task
Visible	Indicate to display or hide task

## Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## ModelAdvisor.Task

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Define custom tasks

### Syntax

```
task_obj = ModelAdvisor.Task(task_ID)
```

### Description

`task_obj = ModelAdvisor.Task(task_ID)` creates a task object, `task_obj`, with a unique identifier, `task_ID`. `task_ID` must remain constant. If you do not specify `task_ID`, the Model Advisor assigns a random `task_ID` to the task object.

You can use one `ModelAdvisor.Check` object in multiple `ModelAdvisor.Task` objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution appears** in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for `Visible` and `LicenseName`.

### Examples

In the following example, you create three task objects, `MAT1`, `MAT2`, and `MAT3`.

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');
```

### See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## ModelAdvisor.Text class

**Package:** ModelAdvisor

Create Model Advisor text output

### Description

Instances of ModelAdvisor.Text class create formatted text for the Model Advisor output.

### Construction

ModelAdvisor.Text

Create Model Advisor text output

### Methods

setBold

Specify bold text

setColor

Specify text color

setHyperlink

Specify hyperlinked text

setItalic

Italicize text

setRetainSpaceReturn

Retain spacing and returns in text

setSubscript

Specify subscripted text

setSuperscript

Specify superscripted text

setUnderlined

Underline text

### Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

## Examples

```
t1 = ModelAdvisor.Text('This is some text');
```

## See Also

[“Model Advisor Customization”](#)

## Topics

[“Create Model Advisor Checks”](#)

[“Format Check Results”](#)

## ModelAdvisor.Text

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Create Model Advisor text output

### Syntax

```
text = ModelAdvisor.Text(content, {attribute})
```

### Description

`text = ModelAdvisor.Text(content, {attribute})` creates a text object for the Model Advisor output.

### Input Arguments

*content*                      Optional character vector specifying the content of the text object. If *content* is empty, empty text is output.



- attribute* Optional cell array of character vectors specifying the formatting of the content. If no attribute is specified, the output text has default coloring with no formatting. Possible formatting options include:
- 'normal' (default) — Text is default color and style.
  - 'bold' — Text is bold.
  - 'italic' — Text is italicized.
  - 'underline' — Text is underlined.
  - 'pass' — Text is green.
  - 'warn' — Text is yellow.
  - 'fail' — Text is red.
  - 'keyword' — Text is blue.
  - 'subscript' — Text is subscripted.
  - 'superscript' — Text is superscripted.

## Output Arguments

text	The text object you create
------	----------------------------

## Examples

```
text = ModelAdvisor.Text('Sub entry 1', {'pass','bold'})
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## publish

**Class:** ModelAdvisor.Root

**Package:** ModelAdvisor

Publish object in Model Advisor root

## Syntax

```
publish(root_obj, check_obj, location)
publish(root_obj, group_obj)
publish(root_obj, procedure_obj)
publish(root_obj, fg_obj)
```

## Description

`publish(root_obj, check_obj, location)` specifies where the Model Advisor places the check in the Model Advisor tree. `location` is either one of the subfolders in the **By Product** folder, or the name of a new subfolder to put in the **By Product** folder. Use a pipe-delimited character vector to indicate multiple subfolders. For example, to add a check to the **Simulink Check > Modeling Standards** folder, use the following: `'Simulink Check|Modeling Standards'`.

If the **By Product** is not displayed in the Model Advisor window, select **Show By Product Folder** from the **Settings > Preferences** dialog box.

`publish(root_obj, group_obj)` specifies the `ModelAdvisor.Group` object to publish as a folder in the **Model Advisor Task Manager** folder.

`publish(root_obj, procedure_obj)` specifies the `ModelAdvisor.Procedure` object to publish.

`publish(root_obj, fg_obj)` specifies the `ModelAdvisor.FactoryGroup` object to publish as a subfolder in the **By Task** folder.

## Examples

```
% publish check into By Product > Demo group.  
mdladvRoot.publish(rec, 'Demo');
```

## See Also

### Topics

“Define Where Custom Checks Appear”

“Define Where Tasks Appear”

“Define Where Custom Folders Appear”

## refresh\_customizations

**Class:** Advisor.Manager

**Package:** Advisor

Refresh Model Advisor check information cache

### Syntax

```
Advisor.Manager.refresh_customizations()
```

### Description

`Advisor.Manager.refresh_customizations()` refreshes the Model Advisor check information cache.

### Alternatives

To refresh the cache from Model Advisor, select **Settings > Preferences**. Click **Update check information cache**, then click **OK**. To see updates, close and reopen model, then start Model Advisor.

### See Also

#### Topics

“Create and Add Custom Checks - Basic Examples”

“Create Check for Model Configuration Parameters”

**Introduced in R2016b**

# register

**Class:** ModelAdvisor.Root

**Package:** ModelAdvisor

Register object in Model Advisor root

## Syntax

```
register(MAobj, obj)
```

## Description

`register(MAobj, obj)` registers the object, *obj*, in the root object *MAobj*.

In the Model Advisor memory, the `register` method registers the following types of objects:

- ModelAdvisor.Check
- ModelAdvisor.FactoryGroup
- ModelAdvisor.Group
- ModelAdvisor.Procedure
- ModelAdvisor.Task

The `register` method places objects in the Model Advisor memory that you use in other functions. The `register` method does not place objects in the Model Advisor tree.

## Examples

```
mdladvRoot = ModelAdvisor.Root;  
  
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.DisplayName='Example task with input parameter and auto-fix ability';  
MAT1.setCheck('com.mathworks.sample.Check1');  
mdladvRoot.register(MAT1);
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT2.DisplayName='Example task 2';  
MAT2.setCheck('com.mathworks.sample.Check2');  
mdladvRoot.register(MAT2);  
  
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');  
MAT3.DisplayName='Example task 3';  
MAT3.setCheck('com.mathworks.sample.Check3');  
mdladvRoot.register(MAT3)
```

---

## run

**Class:** Advisor.Application

**Package:** Advisor

Run Model Advisor analysis on model components

## Syntax

```
run(app)
```

## Description

run(app) runs a Model Advisor analysis, as specified by the Application object.

## Examples

This example shows how to create an Application object, set root analysis to RootModel, and run a Model Advisor analysis.

```
% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Run Model Advisor analysis
run(app);
```

## Input Arguments

**app** — Application

Advisor.Application object

Advisor.Application object, created by Advisor.Manager.createApplication

## **See Also**

`Advisor.Manager.createApplication | setAnalysisRoot`

**Introduced in R2015b**



# selectCheckInstances

**Class:** `Advisor.Application`

**Package:** `Advisor`

Select check instances to use in Model Advisor analysis

## Syntax

```
selectCheckInstances(app)
selectCheckInstances(app,Name,Value)
```

## Description

You can select check instances to use in a Model Advisor analysis. A check instance is an instantiation of a `ModelAdvisor.Check` object in the Model Advisor configuration. When you change the Model Advisor configuration, the check instance ID might change. To obtain the check instance ID, use the `getCheckInstanceIDs` method.

`selectCheckInstances(app)` selects all check instances to use for Model Advisor analysis.

`selectCheckInstances(app,Name,Value)` selects check instances specified by `Name,Value` pair arguments to use for Model Advisor analysis.

## Input Arguments

**app** — **Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

## **IDs — Check instance IDs**

cell array

Select check instances to use in Model Advisor analysis, as specified as a cell array of IDs

Data Types: cell

## **Examples**

### **Select All Check Instances to Use in Model Advisor Analysis**

This example shows how to set the root model, create an `Application` object, set root analysis, and select all check instances for Model Advisor analysis.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Select all checks
selectCheckInstances(app);
```

### **Select Check Instance for Model Advisor Analysis Using Instance ID**

This example shows how to set the root model, create an `Application` object, set root analysis, and select a check using instance ID.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);
```

```
% Select "Identify unconnected lines, input ports, and output  
% ports" check using check instance ID  
instanceID = getCheckInstanceIDs(app,'mathworks.design.UnconnectedLinesPorts');  
checkinstanceID = instanceID(1);  
selectCheckInstances(app,'IDs',checkinstanceID);
```

## See Also

Advisor.Manager.createApplication | deselectCheckInstances |  
getCheckInstanceIDs | setAnalysisRoot

**Introduced in R2015b**

## selectComponents

**Class:** `Advisor.Application`

**Package:** `Advisor`

Select model components for Model Advisor analysis

### Syntax

```
selectComponents(app)
selectComponents(app,Name,Value)
```

### Description

You can select model components for Model Advisor analysis. A model component is a model in the system hierarchy. Models that the root model references and that `Advisor.Application.setAnalysisRoot` specifies are model components. By default, all components are selected.

`selectComponents(app)` includes all components for Model Advisor analysis.

`selectComponents(app,Name,Value)` includes model components specified by `Name,Value` pair arguments for Model Advisor analysis.

### Input Arguments

**app — Application**

`Advisor.Application` object

`Advisor.Application` object, created by `Advisor.Manager.createApplication`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

### **IDs — Component IDs**

cell array

Components to select for Model Advisor analysis, as specified by a cell array of IDs

Data Types: cell

### **HierarchicalSelection — Select component and component children**

false (default) | true

Select components specified by IDs and component children from Model Advisor analysis.

Data Types: logical

## **Examples**

### **Include All Components in Model Advisor Analysis**

This example shows how to set the root model, create an `Application` object, set root analysis, and include model components in Model Advisor analysis.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';
```

```
% Create an Application object
app = Advisor.Manager.createApplication();
```

```
% Set the Application object root analysis
setAnalysisRoot(app,'Root',RootModel);
```

```
% Select all components
selectComponents(app);
```

### **Select Components for Model Advisor Analysis Using IDs**

This example shows how to set the root model, create an `Application` object, set root analysis, and include model components using IDs.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);

% Select component using IDs
selectComponents(app, 'IDs', RootModel);
```

## See Also

Advisor.Manager.createApplication | deselectComponents |  
setAnalysisRoot

**Introduced in R2015b**

## setAction

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Specify action for check

## Syntax

```
setAction(check_obj, action_obj)
```

## Description

setAction(check\_obj, action\_obj) returns the action object action\_obj to use in the check check\_obj. The setAction method identifies the action you want to use in a check.

## See Also

“Model Advisor Customization” | ModelAdvisor.Action

## Topics

“Create Model Advisor Checks”

## setAlign

**Class:** ModelAdvisor.Paragraph

**Package:** ModelAdvisor

Specify paragraph alignment

## Syntax

```
setAlign(paragraph, alignment)
```

## Description

setAlign(paragraph, alignment) specifies the alignment of text. Possible values are:

- 'left' (default)
- 'right'
- 'center'

## Examples

```
report_paragraph = ModelAdvisor.Paragraph;  
setAlign(report_paragraph, 'center');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”



# setAnalysisRoot

**Class:** Advisor.Application

**Package:** Advisor

Specify model hierarchy for Model Advisor analysis

## Syntax

```
setAnalysisRoot(app, 'Root', root)
setAnalysisRoot(app, 'Root', root, Name, Value)
```

## Description

Specify the model hierarchy for an Application object analysis.

`setAnalysisRoot(app, 'Root', root)` specifies the analysis root.

`setAnalysisRoot(app, 'Root', root, Name, Value)` specifies the analysis root using `Name, Value` options.

## Input Arguments

**app — Application**

Advisor.Application object

Advisor.Application object, created by `Advisor.Manager.createApplication`

**'Root', root — Name, Value argument specifying model or subsystem path**

character vector

Comma-separated `Name, Value` argument specifying model or subsystem path

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name, Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

## **RootType — Analysis root**

Model (default) | Subsystem

## **Examples**

### **Specify Root Model as Analysis Root**

This example shows how to set the root model, create an Application object, and set the root analysis.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', RootModel);
```

### **Specify Subsystem as Analysis Root**

This example shows how to set the root model, create an Application object, and specify a subsystem as the analysis root.

```
% Set root model to sldemo_mdref_basic model
RootModel='sldemo_mdref_basic';

% Create an Application object
app = Advisor.Manager.createApplication();

% Set the Application object root analysis
setAnalysisRoot(app, 'Root', 'sldemo_mdref_basic/CounterA', 'RootType', 'Subsystem');
```

## **See Also**

Advisor.Manager.createApplication

**Introduced in R2015b**

## setAnalysisRoot

**Class:** slmetric.Engine

**Package:** slmetric

Specify model or subsystem for metric analysis

### Syntax

```
setAnalysisRoot(metric_engine, 'Root', root)
setAnalysisRoot(metric_engine, 'Root', root, Name, Value)
```

### Description

Specify the model or subsystem for slmetric.Engine metric object analysis.

setAnalysisRoot(metric\_engine, 'Root', root) specifies the metric analysis root.

For an slmetric.Engine object, before you set the ModelReferencesSimulationMode and AnalyzeLibraries properties, specify the setAnalysisRoot method. The setAnalysisRoot method persists between model runs. If you change these property values and then apply the setAnalysisRoot method, the original values override the new values. For example, for one run, suppose you set these values:

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', 'vdp');
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = true;
```

For the next run, if you want to change the ModelReferencesSimulationMode or the AnalyzeLibraries properties, be sure to first specify the setAnalysisRoot method:

```
metric_engine = slmetric.Engine();
setAnalysisRoot(metric_engine, 'Root', 'vdp');
metric_engine.AnalyzeModelReferences = false;
metric_engine.AnalyzeLibraries = false;
```

`setAnalysisRoot(metric_engine, 'Root', root, Name, Value)` specifies the metric analysis root by using `Name, Value` pairs.

## Input Arguments

### **metric\_engine** — Collects and accesses metric data

`slmetric.Engine` object

When you call `execute`, `metric_engine` collects metric data for all MathWorks metrics or for the specified `MetricIDs`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

### **'Root'** — Name, Value argument specifying model or subsystem path

character vector

Comma-separated `Name, Value` argument specifying model or subsystem path. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes.

### **RootType** — Type of model component for metric analysis

`Model` (default) | `Subsystem`

Comma-separated `Name, Value` argument specifying the `RootType`

## Examples

### **Specify Model for Metric Analysis**

This example shows how to set the root model, create an `slmetric.Engine` object, and specify the model for metric analysis.

```
% Set root model to vdp model
RootModel='vdp';

% Create an slmetric.Engine object
metric_engine = slmetric.Engine();
```

```
% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', RootModel);
```

## Specify Subsystem for Metric Analysis

This example shows how to set the root model, create an `slmetric.Engine` object, and specify a subsystem for metric analysis.

```
% Set subsystem to CounterA
Subsys = 'sf_car/Engine';

% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Set a subsystem for metric analysis
setAnalysisRoot(metric_engine, 'Root', Subsys, 'RootType', 'Subsystem');
```

## See Also

`slmetric.metric.Metric` | `slmetric.metric.ResultCollection` |  
`slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”  
“Model Metrics” on page 2-419

**Introduced in R2016a**

## setBold

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Specify bold text

## Syntax

```
setBold(text, mode)
```

## Description

`setBold(text, mode)` specifies whether text should be formatted in bold font.

## Input Arguments

<code>text</code>	Instantiation of the ModelAdvisor.Text class
<code>mode</code>	A Boolean value indicating bold formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Format the text in bold font.</li><li>• <code>false</code> — Do not format the text in bold font.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('This is some text');  
setBold(t1, 'true');
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”



# setCallbackFcn

**Class:** ModelAdvisor.Action

**Package:** ModelAdvisor

Specify action callback function

## Syntax

```
setCallbackFcn(action_obj, @handle)
```

## Description

setCallbackFcn(action\_obj, @handle) specifies the handle to the callback function, handle, to use with the action object, action\_obj.

## Examples

---

**Note** The following example is a fragment of code from the `sl_customization.m` file for the example model, `slvnvdemo_mdldv`. The example does not execute as shown without the additional content found in the `sl_customization.m` file.

---

```
rec = ModelAdvisor.Check('mathworks.example.optimizationSettings');  
% Define an automatic fix action for this check  
modifyAction = ModelAdvisor.Action;  
modifyAction.setCallbackFcn(@modifyOptimizationSetting);  
modifyAction.Name = 'Modify Settings';  
modifyAction.Description = ['Modify model configuration optimization' ...  
    ' settings that can impact safety'];  
modifyAction.Enable = true;  
rec.setAction(modifyAction);
```

## See Also

“Model Advisor Customization”

## **Topics**

“Define Check Actions”

“Create Model Advisor Checks”

setActionEnable

## setCallbackFcn

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Specify callback function for check

### Syntax

```
setCallbackFcn(check_obj, @handle, context, style)
```

### Description

setCallbackFcn(check\_obj, @handle, context, style) specifies the callback function to use with the check, check\_obj.

### Input Arguments

check_obj	Instantiation of the ModelAdvisor.Check class
handle	Handle to a check callback function
context	Context for checking the model or subsystem: <ul style="list-style-type: none"><li>• 'None' — No special requirements.</li><li>• 'PostCompile' — The model must be compiled.</li></ul>

style

Type of callback function:

- 'StyleOne' — Simple check callback function, for formatting results using template
- 'StyleTwo' — Detailed check callback function
- 'StyleThree' — Check callback functions with hyperlinked results
- 'DetailStyle' — Check callback function for result collections. This style is recommended for authoring Model Advisor checks.

## Examples

This example illustrates the definition for a check using a callback function whose style is defined as `DetailStyle`.

```
% This is the recommended style to author checks.
function defineModelAdvisorChecks
mdladvRoot = ModelAdvisor.Root;
rec = ModelAdvisor.Check('com.mathworks.sample.Check0');
rec.Title = 'Check whether block names appear below blocks (recommended check style)';
rec.TitleTips = 'Example new style callback (recommended check style)';
rec.setCallbackFcn(@SampleNewCheckStyleCallback,'None','DetailStyle');
% set fix operation
myAction0 = ModelAdvisor.Action;
myAction0.setCallbackFcn(@sampleActionCB0);
myAction0.Name='Make block names appear below blocks';
myAction0.Description='Click the button to place block names below blocks';
rec.setAction(myAction0);
mdladvRoot.publish(rec, 'Demo'); % publish check into Demo group.
```

## See Also

“Model Advisor Customization”

## Topics

“Create Callback Functions and Results”

“Create Model Advisor Checks”

# setCheck

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Specify check used in task

## Syntax

```
setCheck(task, check_ID)
```

## Description

setCheck(task, check\_ID) specifies the check to use in the task.

You can use one ModelAdvisor.Check object in multiple ModelAdvisor.Task objects, allowing you to place the same check in multiple locations in the Model Advisor tree. For example, **Check for implicit signal resolution** appears in the **By Product > Simulink folder** and in the **By Task > Model Referencing** folder in the Model Advisor tree.

When adding checks as tasks, the Model Advisor uses the task properties instead of the check properties, except for Visible and LicenseName.

## Input Arguments

task	Instantiation of the ModelAdvisor.Task class
check_ID	A unique identifier for the check to use in the task

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
setCheck(MAT1, 'com.mathworks.sample.Check1');
```

## setCheckedText

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add description of check to result

### Syntax

```
setCheckedText(ft_obj, text)
```

### Description

`setCheckedText(ft_obj, text)` is an optional method that adds text or a model advisor template object as the first item in the report. Use this method to add information describing the overall check.

### Input Arguments

**ft\_obj**

A handle to a template object.

**text**

A character vector or a handle to a formatting object.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

*text* appears as the first line in the analysis result.

## Examples

Create a list object, `ft`, and add a line of text to the result:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setCheckText(ft, ['Identify unconnected lines, input ports,...  
    'and output ports in the model']);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## setColHeading

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table column title

### Syntax

```
setColHeading(table, column, heading)
```

### Description

`setColHeading(table, column, heading)` specifies that the column header of column is set to heading.

### Input Arguments

<code>table</code>	Instantiation of the <code>ModelAdvisor.Table</code> class
<code>column</code>	An integer specifying the column number
<code>heading</code>	A character vector, element object, or object array specifying the table column title

### Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setColHeading(table1, 1, 'Header 1');  
setColHeading(table1, 2, 'Header 2');  
setColHeading(table1, 3, 'Header 3');
```

### See Also

“Model Advisor Customization”



## **Topics**

“Create Model Advisor Checks”

## setColHeadingAlign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify column title alignment

### Syntax

```
setColHeadingAlign(table, column, alignment)
```

### Description

`setColHeadingAlign(table, column, alignment)` specifies the alignment of the column heading.

### Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>column</code>	An integer specifying the column number
<code><i>alignment</i></code>	Alignment of the column heading. <i>alignment</i> can have one of the following values: <ul style="list-style-type: none"><li>• left (default)</li><li>• right</li><li>• center</li></ul>

### Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setColHeading(table1, 1, 'Header 1');  
setColHeadingAlign(table1, 1, 'center');  
setColHeading(table1, 2, 'Header 2');
```

```
setColHeadingAlign(table1, 2, 'center');  
setColHeading(table1, 3, 'Header 3');  
setColHeadingAlign(table1, 3, 'center');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## setColHeadingValign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify column title vertical alignment

### Syntax

```
setColHeadingValign(table, column, alignment)
```

### Description

`setColHeadingValign(table, column, alignment)` specifies the vertical alignment of the column heading.

### Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>column</code>	An integer specifying the column number
<code>alignment</code>	Vertical alignment of the column heading. <i>alignment</i> can have one of the following values: <ul style="list-style-type: none"><li>• top (default)</li><li>• middle</li><li>• bottom</li></ul>

### Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setColHeading(table1, 1, 'Header 1');  
setColHeadingValign(table1, 1, 'middle');  
setColHeading(table1, 2, 'Header 2');
```

```
setColHeadingValign(table1, 2, 'middle');  
setColHeading(table1, 3, 'Header 3');  
setColHeadingValign(table1, 3, 'middle');
```

## **See Also**

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## setColor

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Specify text color

## Syntax

```
setColor(text, color)
```

## Description

`setColor(text, color)` sets the text color to *color*.

## Input Arguments

<i>text</i>	Instantiation of the ModelAdvisor.Text class
<i>color</i>	Color of the text, specified as one of the following formatting options: <ul style="list-style-type: none"><li>• 'normal' (default) — Text is default color.</li><li>• 'pass' — Text is green.</li><li>• 'warn' — Text is yellow.</li><li>• 'fail' — Text is red.</li><li>• 'keyword' — Text is blue.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('This is a warning');  
setColor(t1, 'warn');
```

## setColSpan

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Specify number of columns for input parameter

### Syntax

```
setColSpan(input_param, [start_col end_col])
```

### Description

setColSpan(input\_param, [start\_col end\_col]) specifies the number of columns that the parameter occupies. Use the `setColSpan` method to specify where you want an input parameter located in the layout grid when there are multiple input parameters.

### Input Arguments

input_param	Instantiation of the ModelAdvisor.InputParameter class
start_col	A positive integer representing the first column that the input parameter occupies in the layout grid
end_col	A positive integer representing the last column that the input parameter occupies in the layout grid

### Examples

```
inputParam2 = ModelAdvisor.InputParameter;  
inputParam2.Name = 'Standard font size';  
inputParam2.Value='12';  
inputParam2.Type='String';  
inputParam2.Description='sample tooltip';
```

```
inputParam2.setRowSpan([2 2]);  
inputParam2.setColSpan([1 1]);
```



# setColTitles

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add column titles to table

## Syntax

```
setColTitles(ft_obj, {col_title_1, col_title_2, ...})
```

## Description

`setColTitles(ft_obj, {col_title_1, col_title_2, ...})` is method you must use when you create a template object that is a table type. Use it to specify the titles of the columns in the table.

---

**Note** Before adding data to a table, you must specify column titles.

---

## Input Arguments

***ft\_obj***

A handle to a template object.

***col\_title\_N***

A cell of character vectors or handles to formatting objects, specifying the column titles.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The order of the `col_title_N` inputs determines which column the title is in. If you do not add data to the table, the Model Advisor does not display the table in the result.

## Examples

Create a table object, `ft`, and specify two column titles:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');  
setColTitles(ft, {'Index', 'Block Name'});
```

## See Also

["Model Advisor Customization"](#)

## Topics

["Create Model Advisor Checks"](#)

["Format Check Results"](#)

## setColWidth

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify column widths

### Syntax

```
setColWidth(table, column, width)
```

### Description

`setColWidth(table, column, width)` specifies the column.

The `setColWidth` method specifies the table column widths relative to the entire table width. If column widths are [1 2 3], the second column is twice the width of the first column, and the third column is three times the width of the first column. Unspecified columns have a default width of 1. For example:

```
setColWidth(1, 1);  
setColWidth(3, 2);
```

specifies [1 1 2] column widths.

### Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>column</code>	An integer specifying column number
<code>width</code>	An integer or array of integers specifying the column widths, relative to the entire table width

## Examples

```
table1 = ModelAdvisor.Table(2, 3)
setColWidth(table1, 1, 1);
setColWidth(table1, 3, 2);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

# setEntries

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Set contents of table

## Syntax

```
setEntries(content)
```

## Description

setEntries(content) sets content of the table.

## Input Arguments

content	A 2-D cell array containing the contents of the table. Each item of the cell array must be either a character vector or an instance of ModelAdvisor.Element. The size of the cell array must be equal to the size of the table specified in the ModelAdvisor.Table constructor.
---------	---

## Examples

```
table = ModelAdvisor.Table(4,3);
contents = cell(4,3); % 4 by 3 table
for k=1:4
    for m=1:3
        contents{k,m} = ['Contents for row-' num2str(k) ' column-' num2str(m)];
    end
end
table.setEntries(contents);
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”

# setEntry

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Add cell to table

## Syntax

```
setEntry(table, row, column, string)
setEntry(table, row, column, content)
```

## Description

setEntry(table, row, column, string) adds a character vector to a cell in a table.

setEntry(table, row, column, content) adds an object specified by content to a cell in a table.

## Input Arguments

table	Instantiation of the ModelAdvisor.Table class
row	An integer specifying the row
column	An integer specifying the column
string	A character vector representing the contents of the entry
content	An element object or object array specifying the content of the table entries

## Examples

Create two tables and insert table2 into the first cell of table1:

```
table1 = ModelAdvisor.Table(1, 1);  
table2 = ModelAdvisor.Table(2, 3);  
.  
.  
.  
setEntry(table1, 1, 1, table2);
```

### **See Also**

“Model Advisor Customization”

### **Topics**

“Create Model Advisor Checks”



## setEntryAlign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table cell alignment

### Syntax

```
setEntryAlign(table, row, column, alignment)
```

### Description

`setEntryAlign(table, row, column, alignment)` specifies the cell alignment of the designated cell.

### Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>row</code>	An integer specifying row number
<code>column</code>	An integer specifying column number
<code><i>alignment</i></code>	Cell alignment, specified as one of the following: <ul style="list-style-type: none"><li>• 'left' (default)</li><li>• 'right'</li><li>• 'center'</li></ul>

### Examples

```
table1 = ModelAdvisor.Table(2,3);  
setHeading(table1, 'New Table');
```

.

```
.  
.setEntry(table1, 1, 1, 'First Entry');  
.setEntryAlign(table1, 1, 1, 'center');
```

### **See Also**

“Model Advisor Customization”

### **Topics**

“Create Model Advisor Checks”

# setEntryValign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table cell vertical alignment

## Syntax

```
setEntryValign(table, row, column, alignment)
```

## Description

`setEntryValign(table, row, column, alignment)` specifies the cell alignment of the designated cell.

## Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>row</code>	An integer specifying row number
<code>column</code>	An integer specifying column number
<code><i>alignment</i></code>	Cell vertical alignment, specified as one of the following: <ul style="list-style-type: none"><li>• 'top' (default)</li><li>• 'middle'</li><li>• 'bottom'</li></ul>

## Examples

```
table1 = ModelAdvisor.Table(2,3);  
setHeading(table1, 'New Table');
```

.

```
.  
.setEntry(table1, 1, 1, 'First Entry');  
.setEntryValign(table1, 1, 1, 'middle');
```

## **See Also**

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

# setHeading

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table title

## Syntax

```
setHeading(table, title)
```

## Description

setHeading(table, title) specifies the table title.

## Input Arguments

table	Instantiation of the ModelAdvisor.Table class
title	A character vector, element object, or object array that specifies the table title

## Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setHeading(table1, 'New Table');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## setHeadingAlign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table title alignment

### Syntax

```
setHeadingAlign(table, alignment)
```

### Description

`setHeadingAlign(table, alignment)` specifies the alignment for the table title.

### Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>alignment</code>	Table title alignment, specified as one of the following: <ul style="list-style-type: none"><li>• 'left' (default)</li><li>• 'right'</li><li>• 'center'</li></ul>

### Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setHeading(table1, 'New Table');  
setHeadingAlign(table1, 'center');
```

### See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## setHyperlink

**Class:** ModelAdvisor.Image

**Package:** ModelAdvisor

Specify hyperlink location

### Syntax

```
setHyperlink(image, url)
```

### Description

`setHyperlink(image, url)` specifies the target location of the hyperlink associated with `image`.

### Input Arguments

<code>image</code>	Instantiation of the <code>ModelAdvisor.Image</code> class
<code>url</code>	The target URL

### Examples

```
matlab_logo=ModelAdvisor.Image;  
setHyperlink(matlab_logo, 'https://www.mathworks.com');
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”



# setHyperlink

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Specify hyperlinked text

## Syntax

```
setHyperlink(text, url)
```

## Description

setHyperlink(text, url) creates a hyperlink from the text to the specified URL.

## Input Arguments

text	Instantiation of the ModelAdvisor.Text class
url	The target location of the URL

## Examples

```
t1 = ModelAdvisor.Text('MathWorks home page');  
setHyperlink(t1, 'https://www.mathworks.com');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

## setImageSource

**Class:** ModelAdvisor.Image

**Package:** ModelAdvisor

Specify image location

### Syntax

```
setImageSource(image_obj, source)
```

### Description

setImageSource(image\_obj, source) specifies the location of the image.

### Input Arguments

image_obj	Instantiation of the ModelAdvisor.Image class
source	The location of the image file

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

# setInformation

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add description of subcheck to result

## Syntax

```
setInformation(ft_obj, text)
```

## Description

`setInformation(ft_obj, text)` is an optional method that adds text as the first item after the subcheck title. Use this method to add information describing the subcheck.

## Input Arguments

### **ft\_obj**

A handle to a template object.

### **text**

A character vector or a handle to a formatting object, that describes the subcheck.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The Model Advisor displays *text* after the title of the subcheck.

## Examples

Create a list object, `ft`, and specify a subcheck title and description:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubTitle(ft, ['Check for constructs in the model '...
    'that are not supported when generating code']);
setInformation(ft, ['Identify blocks that should not '...
    'be used for code generation.']);
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

“Format Check Results”

# setInputParameters

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Specify input parameters for check

## Syntax

```
setInputParameters(check_obj, params)
```

## Description

setInputParameters(check\_obj, params) specifies ModelAdvisor.InputParameter objects (params) to be used as input parameters to a check (check\_obj).

## Input Arguments

check_obj	Instantiation of the ModelAdvisor.Check class
params	A cell array of ModelAdvisor.InputParameters objects

## Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');  
inputParam1 = ModelAdvisor.InputParameter;  
inputParam2 = ModelAdvisor.InputParameter;  
inputParam3 = ModelAdvisor.InputParameter;  
setInputParameters(rec, {inputParam1,inputParam2,inputParam3});
```

## See Also

“Model Advisor Customization” | ModelAdvisor.InputParameter

**Topics**

“Create Model Advisor Checks”

# setInputParametersLayoutGrid

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Specify layout grid for input parameters

## Syntax

```
setInputParametersLayoutGrid(check_obj, [row col])
```

## Description

`setInputParametersLayoutGrid(check_obj, [row col])` specifies the layout grid for input parameters in the Model Advisor. Use the `setInputParametersLayoutGrid` method when there are multiple input parameters.

## Input Arguments

<code>check_obj</code>	Instantiation of the ModelAdvisor.Check class
<code>row</code>	Number of rows in the layout grid
<code>col</code>	Number of columns in the layout grid

## Examples

```
% --- sample check 1
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.Title = 'Check Simulink block font';
rec.TitleTips = 'Example style three callback';
rec.setCallbackFcn(@SampleStyleThreeCallback, 'None', 'StyleThree');
rec.setInputParametersLayoutGrid([3 2]);
```

## See Also

“Model Advisor Customization” | ModelAdvisor.InputParameter

**Topics**

“Create Model Advisor Checks”



## setItalic

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Italicize text

## Syntax

```
setItalic(text, mode)
```

## Description

`setItalic(text, mode)` specifies whether text should be italicized.

## Input Arguments

<code>text</code>	Instantiation of the ModelAdvisor.Text class
<code>mode</code>	A Boolean value indicating italic formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Italicize the text.</li><li>• <code>false</code> — Do not italicize the text.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('This is some text');  
setItalic(t1, 'true');
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”

## setListObj

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add list of hyperlinks to model objects

## Syntax

```
setListObj(ft_obj, {model_obj})
```

## Description

`setListObj(ft_obj, {model_obj})` is an optional method that generates a bulleted list of hyperlinks to model objects. *ft\_obj* is a handle to a list template object. *model\_obj* is a cell array of handles or full paths to blocks, or model objects that the Model Advisor displays as a bulleted list of hyperlinks in the report.

## Examples

Create a list object, `ft`, and add a list of the blocks found in the model:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
  
% Find all the blocks in the system  
allBlocks = find_system(system);  
  
% Add the blocks to a list  
setListObj(ft, allBlocks);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

# setRecAction

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add Recommended Action section and text

## Syntax

```
setRecAction(ft_obj, {text})
```

## Description

setRecAction(*ft\_obj*, {*text*}) is an optional method that adds a Recommended Action section to the report. Use this method to describe how to fix the check.

## Input Arguments

### **ft\_obj**

A handle to a template object.

### **text**

A cell array of character vectors or handles to formatting objects, that describes the recommended action to fix the issues reported by the check.

Valid formatting objects are: ModelAdvisor.Image, ModelAdvisor.LineBreak, ModelAdvisor.List, ModelAdvisor.Paragraph, ModelAdvisor.Table, and ModelAdvisor.Text.

The Model Advisor displays the recommended action as a separate section below the list or table in the report.

## Examples

Create a list object, ft, find Gain blocks in the model, and recommend changing them:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
% Find all Gain blocks
gainBlocks = find_system(gcs, 'BlockType','Gain');

% Find Gain blocks
for idx = 1:length(gainBlocks)
    gainObj = get_param(gainBlocks(idx), 'Object');

    setRecAction(ft, {'If you are using these blocks '...
        'as buffers, you should replace them with '...
        'Signal Conversion blocks'});
end
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

# setRefLink

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add See Also section and links

## Syntax

```
setRefLink(ft_obj, {'standard'})  
setRefLink(ft_obj, {'url', 'standard'})
```

## Description

`setRefLink(ft_obj, {'standard'})` is an optional method that adds a See Also section above the table or list in the result. Use this method to add references to standards. `ft_obj` is a handle to a template object. `standard` is a cell array of character vectors that you want to display in the result. If you include more than one cell, the Model Advisor displays the character vectors in a bulleted list.

`setRefLink(ft_obj, {'url', 'standard'})` generates a list of links in the See Also section. `url` indicates the location to link to. You must provide the full link including the protocol. For example, `https:\\www.mathworks.com` is a valid link, while `www.mathworks.com` is not a valid link. You can create a link to a protocol that is valid URL, such as a web site address, a full path to a file, or a relative path to a file.

---

**Note** `setRefLink` expects a cell array of cell arrays for the second input.

---

## Examples

Create a list object, `ft`, and add a related standard:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setRefLink(ft, {'IEC 61508-3, Table A.3 (3) 'Language subset'});
```

Create a list object, `ft`, and add a list of related standards:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setRefLink(ft, {
    {'IEC 61508-3, Table A.3 (2) ''Strongly typed programming language''},...
    {'IEC 61508-3, Table A.3 (3) ''Language subset''}});
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

“Format Check Results”



# setRetainSpaceReturn

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Retain spacing and returns in text

## Syntax

```
setRetainSpaceReturn(text, mode)
```

## Description

`setRetainSpaceReturn(text, mode)` specifies whether the text must retain the spaces and carriage returns.

## Input Arguments

<code>text</code>	Instantiation of the <code>ModelAdvisor.Text</code> class
<code>mode</code>	A Boolean value indicating whether to preserve spaces and carriage returns in the text: <ul style="list-style-type: none"><li><code>true</code> (default) — Preserve spaces and carriage returns.</li><li><code>false</code> — Do not preserve spaces and carriage returns.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('MathWorks home page');  
setRetainSpaceReturn(t1, 'true');
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”

# setRowHeading

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table row title

## Syntax

```
setRowHeading(table, row, heading)
```

## Description

setRowHeading(table, row, heading) specifies a title for the designated table row.

## Input Arguments

table	Instantiation of the ModelAdvisor.Table class
row	An integer specifying row number
heading	A character vector, element object, or object array specifying the table row title

## Examples

```
table1 = ModelAdvisor.Table(2,3);  
setRowHeading(table1, 1, 'Row 1 Title');  
setRowHeading(table1, 2, 'Row 2 Title');
```

## See Also

“Model Advisor Customization”

**Topics**

“Create Model Advisor Checks”

# setRowHeadingAlign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table row title alignment

## Syntax

```
setRowHeadingAlign(table, row, alignment)
```

## Description

`setRowHeadingAlign(table, row, alignment)` specifies the alignment for the designated table row.

## Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>row</code>	An integer specifying row number.
<code>alignment</code>	Cell alignment, specified as one of the following: <ul style="list-style-type: none"><li>• 'left' (default)</li><li>• 'right'</li><li>• 'center'</li></ul>

## Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setRowHeading(table1, 1, 'Row 1 Title');  
setRowHeadingAlign(table1, 1, 'center');  
setRowHeading(table1, 2, 'Row 2 Title');  
setRowHeadingAlign(table1, 2, 'center');
```

## **See Also**

*“Model Advisor Customization”*

## **Topics**

*“Create Model Advisor Checks”*

# setRowHeadingValign

**Class:** ModelAdvisor.Table

**Package:** ModelAdvisor

Specify table row title vertical alignment

## Syntax

```
setRowHeadingValign(table, row, alignment)
```

## Description

`setRowHeadingValign(table, row, alignment)` specifies the vertical alignment for the designated table row.

## Input Arguments

<code>table</code>	Instantiation of the ModelAdvisor.Table class
<code>row</code>	An integer specifying row number.
<code>alignment</code>	Cell vertical alignment, specified as one of the following: <ul style="list-style-type: none"><li>• 'top' (default)</li><li>• 'middle'</li><li>• 'bottom'</li></ul>

## Examples

```
table1 = ModelAdvisor.Table(2, 3);  
setRowHeading(table1, 1, 'Row 1 Title');  
setRowHeadingValign(table1, 1, 'middle');  
setRowHeading(table1, 2, 'Row 2 Title');  
setRowHeadingValign(table1, 2, 'middle');
```

## **See Also**

*“Model Advisor Customization”*

## **Topics**

*“Create Model Advisor Checks”*



# setRowSpan

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Specify rows for input parameter

## Syntax

```
setRowSpan(input_param, [start_row end_row])
```

## Description

`setRowSpan(input_param, [start_row end_row])` specifies the number of rows that the parameter occupies. Specify where you want an input parameter located in the layout grid when there are multiple input parameters.

## Input Arguments

<code>input_param</code>	The input parameter object
<code>start_row</code>	A positive integer representing the first row that the input parameter occupies in the layout grid
<code>end_row</code>	A positive integer representing the last row that the input parameter occupies in the layout grid

## Examples

```
inputParam2 = ModelAdvisor.InputParameter;  
inputParam2.Name = 'Standard font size';  
inputParam2.Value='12';  
inputParam2.Type='String';  
inputParam2.Description='sample tooltip';
```

```
inputParam2.setRowSpan([2 2]);  
inputParam2.setColSpan([1 1]);
```

## setSubBar

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add line between subcheck results

## Syntax

```
setSubBar(ft_obj, value)
```

## Description

`setSubBar(ft_obj, value)` is an optional method that adds lines between results for subchecks. *ft\_obj* is a handle to a template object. *value* is a boolean value that specifies when the Model Advisor includes a line between subchecks in the check results. By default, the value is `true`, and the Model Advisor displays the bar. The Model Advisor does not display the bar when you set the value to `false`.

## Examples

Create a list object, `ft`, turn off the subbar:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setSubBar(ft, false);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## setSubResultStatus

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add status to the check or subcheck result

### Syntax

```
setSubResultStatus(ft_obj, 'status')
```

### Description

`setSubResultStatus(ft_obj, 'status')` is an optional method that displays the status in the result. Use this method to display the status of the check or subcheck in the result. *ft\_obj* is a handle to a template object. *status* is a character vector identifying the status of the check:

Pass: Check did not identify issues.

D-Pass: Dependent on configuration parameter. Check did not identify issues.

Warn: Check has identified issues.

Fail: Check fails to execute.

### Examples

Create a list object, `ft`, and add a passing status:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setSubResultStatus(ft, 'Pass');
```

### See Also

“Model Advisor Customization”

### Topics

“Create Model Advisor Checks”

“Format Check Results”

## setSubResultStatusText

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add text below status in result

### Syntax

```
setSubResultStatusText(ft_obj, message)
```

### Description

`setSubResultStatusText(ft_obj, message)` is an optional method that displays text below the status in the result. Use this method to describe the status.

### Input Arguments

**ft\_obj**

A handle to a template object.

**message**

A character vector or a handle to a formatting object that the Model Advisor displays below the status in the report.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

### Examples

Create a list object, `ft`, add a passing status and a description of why the check passed:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');
setSubResultStatus(ft, 'Pass');
setSubResultStatusText(ft, ['Constructs that are not supported when '...
    'generating code were not found in the model or subsystem']);
```

## See Also

“Model Advisor Customization”

## Topics

“Model Advisor Customization”

“Format Check Results”

## setSubscript

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Specify subscripted text

## Syntax

```
setSubscript(text, mode)
```

## Description

setSubscript(text, mode) indicates whether to make text subscript.

## Input Arguments

text	Instantiation of the ModelAdvisor.Text class
mode	A Boolean value indicating subscripted formatting of text: <ul style="list-style-type: none"><li>• true — Make the text subscript.</li><li>• false — Do not make the text subscript.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('This is some text');  
setSubscript(t1, 'true');
```

## See Also

“Model Advisor Customization”



## **Topics**

“Create Model Advisor Checks”

## setSuperscript

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Specify superscripted text

### Syntax

```
setSuperscript(text, mode)
```

### Description

setSuperscript(text, mode) indicates whether to make text superscript.

### Input Arguments

text	Instantiation of the ModelAdvisor.Text class
mode	A Boolean value indicating superscripted formatting of text: <ul style="list-style-type: none"><li>• true — Make the text superscript.</li><li>• false — Do not make the text superscript.</li></ul>

### Examples

```
t1 = ModelAdvisor.Text('This is some text');  
setSuperscript(t1, 'true');
```

### See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## setSubTitle

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add title for subcheck in result

## Syntax

```
setSubTitle(ft_obj, title)
```

## Description

`setSubTitle(ft_obj, title)` is an optional method that adds a subcheck result title. Use this method when you create subchecks to distinguish between them in the result.

## Input Arguments

**ft\_obj**

A handle to a template object.

**title**

A character vector or a handle to a formatting object specifying the title of the subcheck.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

## Examples

Create a list object, `ft`, and add a subcheck title:

```
ft = ModelAdvisor.FormatTemplate('ListTemplate');  
setSubTitle(ft, ['Check for constructs in the model '...  
    'that are not supported when generating code']);
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## setTableInfo

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add data to table

### Syntax

```
setTableInfo(ft_obj, {data})
```

### Description

`setTableInfo(ft_obj, {data})` is an optional method that creates a table. *ft\_obj* is a handle to a table template object. *data* is a cell array of character vectors or objects specifying the information in the body of the table. The Model Advisor creates hyperlinks to objects. If you do not add data to the table, the Model Advisor does not display the table in the result.

---

**Note** Before creating a table, you must specify column titles using the `setColTitle` method.

---

### Examples

Create a table object, `ft`, add column titles, and add data to the table:

```
ft = ModelAdvisor.FormatTemplate('TableTemplate');  
setColTitle(ft, {'Index', 'Block Name'});  
setTableInfo(ft, {'1', 'Gain'});
```

### See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

“Format Check Results”

## setTableTitle

**Class:** ModelAdvisor.FormatTemplate

**Package:** ModelAdvisor

Add title to table

### Syntax

```
setTableTitle(ft_obj, title)
```

### Description

`setTableTitle(ft_obj, title)` is an optional method that adds a title to a table.

### Input Arguments

#### **ft\_obj**

A handle to a template object.

#### **title**

A character vector or a handle to a formatting object specifying the title of the table.

Valid formatting objects are: `ModelAdvisor.Image`, `ModelAdvisor.LineBreak`, `ModelAdvisor.List`, `ModelAdvisor.Paragraph`, `ModelAdvisor.Table`, and `ModelAdvisor.Text`.

The title appears above the table. If you do not add data to the table, the Model Advisor does not display the table and title in the result.

### Examples

Create a table object, `ft`, and add a table title:



```
ft = ModelAdvisor.FormatTemplate('TableTemplate');  
setTableTitle(ft, 'Table of fonts and styles used in model');
```

## See Also

“Model Advisor Customization”

## Topics

“Create Model Advisor Checks”

“Format Check Results”

## setType

**Class:** ModelAdvisor.List

**Package:** ModelAdvisor

Specify list type

## Syntax

```
setType(list_obj, listType)
```

## Description

`setType(list_obj, listType)` specifies the type of list the `ModelAdvisor.List` constructor creates.

## Input Arguments

<code>list_obj</code>	Instantiation of the <code>ModelAdvisor.List</code> class
<code>listType</code>	Specifies the list type: <ul style="list-style-type: none"><li>• numbered</li><li>• bulleted</li></ul>

## Examples

```
subList = ModelAdvisor.List();
subList.setType('numbered')
subList.addItem(ModelAdvisor.Text('Sub entry 1', {'pass','bold'}));
subList.addItem(ModelAdvisor.Text('Sub entry 2', {'pass','bold'}));
```

## See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## setUnderlined

**Class:** ModelAdvisor.Text

**Package:** ModelAdvisor

Underline text

## Syntax

```
setUnderlined(text, mode)
```

## Description

`setUnderlined(text, mode)` indicates whether to underline text.

## Input Arguments

<code>text</code>	Instantiation of the ModelAdvisor.Text class
<code>mode</code>	A Boolean value indicating underlined formatting of text: <ul style="list-style-type: none"><li>• <code>true</code> — Underline the text.</li><li>• <code>false</code> — Do not underline the text.</li></ul>

## Examples

```
t1 = ModelAdvisor.Text('This is some text');  
setUnderlined(t1, 'true');
```

## See Also

“Model Advisor Customization”

## **Topics**

“Create Model Advisor Checks”

## slmetric.Engine class

**Package:** slmetric

Collect metric data on models or model components

### Description

Use a `slmetric.Engine` object to collect metric data on models by calling `execute`. Use `getMetrics` to access the metric data and return an array of `slmetric.metric.ResultCollection` objects. This metric data is persistent in the simulation cache folder. Future instantiations of the `slmetric.Engine` object for the same model can access the cached metric data without regenerating the metric data.

### Construction

`metric_engine = slmetric.Engine()` creates a metric engine object.

### Properties

**AnalysisRoot** — Name of root model or subsystem on which to collect metric data

character vector

Name of root model or subsystem on which to collect metric data, as specified by the `slmetric.Engine.setAnalysisRoot` method. This property is read-only.

**AnalyzeLibraries** — Collect metric data on library linked subsystems in the model

1 (default)

Specify if the metric engine analyzes library-linked subsystems in the root model, including libraries inside referenced models under the root. Metric analysis does not include linked blocks to Simulink built-in libraries. Set this parameter to `false` or `0` to not include libraries in the metric analysis.

Data Types: logical

### **ModelReferencesSimulationMode – Collect metric data on referenced models under the root model**

AllModes (default) | NormalModeOnly | None

Specify if the metric engine analyzes referenced models in your root model. Choose from these values:

<b>Value</b>	<b>Description</b>
None	Metric engine does not collect metric data for referenced models.
NormalModeOnly	Metric engine collects metric data only for referenced models running in normal simulation mode.
AllModes	Metric engine collects metric data for referenced models running in normal and accelerated simulation modes.

Data Types: char

## **Methods**

execute	Collect metric data
getAnalysisRootMetric	Get metric data for one metric for analysis root only
getErrorLog	Get error log
getMetricDistribution	Get metric distribution
getMetrics	Access model metric data
getStatistics	Get statistics on metric data
setAnalysisRoot	Specify model or subsystem for metric analysis
exportMetrics	Export model metrics
getMetricMetaInformation	Obtain metric metadata

## Examples

### Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis,
%     these properties are on by default
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Display the results for the `mathworks.metrics.ExplicitIOCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ', result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
            disp([' Value: ', num2str(result(m).Value)]);
            disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
            disp([' Measures: ', num2str(result(m).Measures)]);
            disp([' AggregatedMeasures: ', num2str(result(m).AggregatedMeasures)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

Here are the results:

```
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic
```



```

Value: 3
AggregatedValue: 4
Measures: 0 3
AggregatedMeasures: 3 3
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic/More Info
Value: 0
AggregatedValue: 0
Measures: 0 0
AggregatedMeasures: 0 0
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_counter
Value: 4
AggregatedValue: 4
Measures: 3 1
AggregatedMeasures: 3 1

```

For the ComponentPath: `sldemo_mdref_basic`, the value is 3 because there are 3 outputs. The three outputs are in the second element of the Measures array. The `slmetric.metric.AggregationMode` is Max, so the `AggregatedValue` is 4 which is the number of inputs and outputs to `sldemo_mdref_counter`. The `AggregatedMeasures` array contains the maximum number of inputs and outputs for a component or subcomponent.

## See Also

`slmetric.metric.Result` | `slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2016a**

## **slmetric.metric.getAvailableMetrics**

**Package:** slmetric.metric

Obtain available metrics

### **Syntax**

```
IDs = slmetric.metric.getAvailableMetrics()  
[IDs,props] = slmetric.metric.getAvailableMetrics()
```

### **Description**

IDs = slmetric.metric.getAvailableMetrics() get metric identifiers for available metrics in the metric engine.

[IDs,props] = slmetric.metric.getAvailableMetrics() get metric identifiers and properties.

### **Examples**

#### **Obtain Available Metric IDs for Model**

This example shows how to obtain the available model metric IDs.

```
ID = slmetric.metric.getAvailableMetrics()
```

```
ID =
```

```
26×1 cell array
```

```
    {'mathworks.metrics.CloneContent'           }  
    {'mathworks.metrics.CloneDetection'        }  
    {'mathworks.metrics.CyclomaticComplexity'   }  
    {'mathworks.metrics.DescriptiveBlockNames'  }  
    {'mathworks.metrics.DiagnosticWarningsCount' }
```

```

{'mathworks.metrics.ExplicitIOCount'           }
{'mathworks.metrics.FileCount'                 }
{'mathworks.metrics.IOCount'                  }
{'mathworks.metrics.LayerSeparation'          }
{'mathworks.metrics.LibraryContent'           }
{'mathworks.metrics.LibraryLinkCount'         }
{'mathworks.metrics.MatlabCodeAnalyzerWarnings' }
{'mathworks.metrics.MatlabFunctionCount'      }
{'mathworks.metrics.MatlabLOCCount'           }
{'mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178' }
{'mathworks.metrics.ModelAdvisorCheckCompliance.maab' }
{'mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178' }
{'mathworks.metrics.ModelAdvisorCheckIssues.maab' }
{'mathworks.metrics.ModelFileCount'           }
{'mathworks.metrics.ParameterCount'           }
{'mathworks.metrics.SimulinkBlockCount'        }
{'mathworks.metrics.StateflowChartCount'      }
{'mathworks.metrics.StateflowChartObjectCount' }
{'mathworks.metrics.StateflowLOCCount'        }
{'mathworks.metrics.SubSystemCount'           }
{'mathworks.metrics.SubSystemDepth'           }

```

### Obtain Available Metric IDs and Metric Properties

This example shows how to obtain the available model metric properties.

```
[ID,PROPS]=slmetric.metric.getAvailableMetrics()
```

ID =

26×1 cell array

```

{'mathworks.metrics.CloneContent'             }
{'mathworks.metrics.CloneDetection'           }
{'mathworks.metrics.CyclomaticComplexity'     }
{'mathworks.metrics.DescriptiveBlockNames'    }
{'mathworks.metrics.DiagnosticWarningsCount'  }
{'mathworks.metrics.ExplicitIOCount'         }
{'mathworks.metrics.FileCount'               }
{'mathworks.metrics.IOCount'                 }
{'mathworks.metrics.LayerSeparation'          }
{'mathworks.metrics.LibraryContent'           }
{'mathworks.metrics.LibraryLinkCount'        }

```

```
{'mathworks.metrics.MatlabCodeAnalyzerWarnings'      }  
{'mathworks.metrics.MatlabFunctionCount'             }  
{'mathworks.metrics.MatlabLOCCount'                 }  
{'mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178'}  
{'mathworks.metrics.ModelAdvisorCheckCompliance.maab' }  
{'mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178'}  
{'mathworks.metrics.ModelAdvisorCheckIssues.maab'   }  
{'mathworks.metrics.ModelFileCount'                 }  
{'mathworks.metrics.ParameterCount'                 }  
{'mathworks.metrics.SimulinkBlockCount'             }  
{'mathworks.metrics.StateflowChartCount'           }  
{'mathworks.metrics.StateflowChartObjectCount'     }  
{'mathworks.metrics.StateflowLOCCount'             }  
{'mathworks.metrics.SubSystemCount'                 }  
{'mathworks.metrics.SubSystemDepth'                 }
```

PROPS =

1×26 struct array with fields:

```
Name  
Description  
IsBuiltIn  
Version
```

## Output Arguments

### IDs — Metric identifiers

cell array of character vectors

Metric identifiers in the metric engine.

### props — Metric properties

structure array

Metric properties, returned as a structure array with the following fields:

Name	Name of the metric algorithm.
Description	Description of the metric algorithm.

IsBuiltIn                    Boolean indicating if the metric is included with Simulink Check.

Version                     Metric algorithm version.

Data Types: struct

## See Also

slmetric.Engine | slmetric.metric.Result |  
slmetric.metric.ResultCollection

**Introduced in R2016a**

## **slmetric.metric.Result class**

**Package:** `slmetric.metric`

Metric data for specified model component and metric algorithm

### **Description**

Instances of `slmetric.metric.Result` contain the metric data for a specified model component and metric algorithm.

### **Construction**

`metric_result = slmetric.metric.Result` creates a handle to a metric results object.

### **Properties**

#### **ID — Numeric identifier**

`integer`

Unique numeric identifier for the metric result object. This property is read-only.

Data Types: `uint64`

#### **ComponentID — Component ID**

`character vector`

Unique identifier of the component object for which the metric is calculated. Use `ComponentID` to trace the generated result object to the analyzed component. Set the `ComponentID` or `ComponentPath` properties by using the `algorithm` method.

This property is read/write.

Data Types: `char`

**ComponentPath — Component path**

character vector

Component path for which metric is calculated. Use `ComponentPath` as an alternative to setting the `ComponentID` property. The metric engine converts the `ComponentPath` to a `ComponentID`. Set the `ComponentID` or `ComponentPath` properties by using the `algorithm` method.

This property is read/write.

Data Types: `char`

**MetricID — Metric identifier**

character vector

Metric identifier for “Model Metrics” on page 2-419 or custom model metrics that you create. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

This property is read/write.

Data Types: `char`

**Value — Metric value**

double (default)

Metric scalar value, generated by the algorithm for the metric specified by `MetricID` and the component specified by `ComponentID`.

If the algorithm does not specify a metric scalar value, the default value is `NaN`. For example, suppose you collect metric data for a model that contains a Stateflow Chart. For the `StateflowChartObjectCount` metric, the `Value` property of the model `slmetric.metric.Result` object is `NaN` because the model itself cannot have Stateflow objects. The `AggregatedValue` property of the model `slmetric.metric.Result` object contains the total number of Stateflow objects in the chart.

This property is read/write.

Data Types: `double`

**AggregatedValue — Aggregated metric value**

double (default)

Metric value aggregated across the model hierarchy. The metric engine implicitly aggregates the metric values based on the `AggregationMode`. Do not set this property. If the metric scalar value is NaN for all components, the `AggregatedValue` is zero.

This property is read-only.

Data Types: `double`

### **Measures — Metric measures**

`double array`

Metric measures, optionally specified by the metric algorithm. Metric measures contain detailed information about the metric value. For example, for a metric that counts the number of blocks per subsystem, you can specify measures that contain the number of virtual and nonvirtual blocks. The metric value is the sum of the virtual and nonvirtual block count.

Set the property by using the `slmetric.metric.Metric.algorithm` method. This property is read/write.

Data Types: `double`

### **AggregatedMeasures — Aggregated metric measures**

`double array`

Metric measures value aggregated across the model hierarchy. The metric engine implicitly aggregates the metric measure values based on the `AggregationMode`. Do not set this property.

This property is read-only.

Data Types: `double`

### **Details — Metric result details**

array of `slmetric.metric.ResultDetail` objects

Details about what the metric engine counts for the `Value` property

This property is read/write.

### **Category — Metric data category based on thresholding criteria**

'Compliant' | 'NonCompliant' | 'Warning' | 'Uncategorized'

Metric data values fall into one of these four categories:



- Compliant—Metric data that is in an acceptable range.
- Warning—Metric data that requires review.
- NonCompliant—Metric data that requires you to modify your model.
- Uncategorized—Metric data that does not have threshold values set.

This property is read-only.

### **Classifications — Metric data category and thresholding criteria**

`slmetric.config.ResultClassification` object

Access the metric data category and the ranges that correspond to each category. This property is empty if no threshold values are set.

### **UserData — User data**

character vector

User data optionally provided by the metric algorithm.

This property is read/write.

Data Types: char

## **Examples**

### **Collect and Access Metric Data for One Metric**

Collect and access model metric data for the model `sldemo_mdhref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis,
% these properties are on by default
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdhref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Display the results for the `mathworks.metrics.ExplicitIOCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ', result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
            disp([' Value: ', num2str(result(m).Value)]);
            disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
            disp([' Measures: ', num2str(result(m).Measures)]);
            disp([' AggregatedMeasures: ', num2str(result(m).AggregatedMeasures)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

Here are the results:

```
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic
Value: 3
AggregatedValue: 4
Measures: 0 3
AggregatedMeasures: 3 3
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic/More Info
Value: 0
AggregatedValue: 0
Measures: 0 0
AggregatedMeasures: 0 0
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_counter
Value: 4
AggregatedValue: 4
Measures: 3 1
AggregatedMeasures: 3 1
```

For the `ComponentPath: sldemo_mdref_basic`, the value is 3 because there are 3 outputs. The three outputs are in the second element of the `Measures` array. The

slmetric.metric.AggregationMode is Max, so the AggregatedValue is 4 which is the number of inputs and outputs to sldemo\_mdhref\_counter. The AggregatedMeasures array contains the maximum number of inputs and outputs for a component or subcomponent.

## See Also

slmetric.Engine | slmetric.metric.Metric |  
slmetric.metric.ResultCollection

## Topics

“Collect Model Metrics Programmatically”  
“Model Metrics” on page 2-419

**Introduced in R2016a**

## slmetric.metric.ResultCollection class

**Package:** slmetric.metric

Metric data for specified model metric

### Description

Instances of `slmetric.metric.ResultCollection` contain the metric data for a specific model metric.

### Construction

`metricRC = slmetric.metric.ResultCollection` creates a handle to a metric result collection object.

### Properties

#### **MetricID — Metric identifier**

character vector

Metric identifier for a MathWorks metric or a custom metric. You can get metric identifiers by calling `slmetric.metric.getAvailableMetrics`.

#### **Status — Unique identifier**

integer

Status code of metric execution. This property is read-only.

Integer	Status
1	No result. Metric algorithm is not applicable to the analyzed system. Components analyzed by the metric not found, or metric with compile requirement cannot be executed on library model.
0	Result collected.

Integer	Status
-1	No result. Error executing metric.
-2	No result available from previous run.
-3	No result. Compilation error.
-4	Empty result. Missing prerequisite.

### Category — Metric data category based on thresholding criteria

'Compliant' | 'NonCompliant' | 'Warning' | 'Uncategorized'

Metric data values fall into one of these four categories:

- **Compliant**—Metric data that is in an acceptable range.
- **Warning**—Metric data that requires review.
- **NonCompliant**—Metric data that requires you to modify your model.
- **Uncategorized**—Metric data that has no threshold values.

If at least one component is **NonCompliant**, this property returns **NonCompliant**. If at least one component is **Warning** and no components are **NonCompliant**, this property returns **Warning**. If all components are **Compliant**, this category returns **Compliant**.

This property is read-only.

### Outdated — Determine if metric data is current

logical

If `true`, the metric data is out-of-date because the model or source files have changed. This property is read-only.

### Results — Metric data collected for executing one or more metrics

array of `slmetric.metric.Result` objects

Metric data collected when you call the `execute` method for one or more metrics. This property is read-only.

## Examples

### Collect and Access Metric Data for One Metric

Collect and access model metric data for the model `sldemo_mdref_basic`.

Create an `slmetric.Engine` object and set the root in the model for analysis.

```
metric_engine = slmetric.Engine();

% Include referenced models and libraries in the analysis,
%     these properties are on by default
metric_engine.ModelReferencesSimulationMode = 'AllModes';
metric_engine.AnalyzeLibraries = 1;

setAnalysisRoot(metric_engine, 'Root', 'sldemo_mdref_basic');
```

Collect model metric data

```
execute(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Get the model metric data that returns an array of `slmetric.metric.ResultCollection` objects, `res_col`.

```
res_col = getMetrics(metric_engine, 'mathworks.metrics.ExplicitIOCount');
```

Display the results for the `mathworks.metrics.ExplicitIOCount` metric.

```
for n=1:length(res_col)
    if res_col(n).Status == 0
        result = res_col(n).Results;

        for m=1:length(result)
            disp(['MetricID: ', result(m).MetricID]);
            disp([' ComponentPath: ', result(m).ComponentPath]);
            disp([' Value: ', num2str(result(m).Value)]);
            disp([' AggregatedValue: ', num2str(result(m).AggregatedValue)]);
            disp([' Measures: ', num2str(result(m).Measures)]);
            disp([' AggregatedMeasures: ', num2str(result(m).AggregatedMeasures)]);
        end
    else
        disp(['No results for:', result(n).MetricID]);
    end
    disp(' ');
end
```

Here are the results:

```
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic
```

```
Value: 3
AggregatedValue: 4
Measures: 0 3
AggregatedMeasures: 3 3
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_basic/More Info
Value: 0
AggregatedValue: 0
Measures: 0 0
AggregatedMeasures: 0 0
MetricID: mathworks.metrics.ExplicitIOCount
ComponentPath: sldemo_mdref_counter
Value: 4
AggregatedValue: 4
Measures: 3 1
AggregatedMeasures: 3 1
```

For the ComponentPath: `sldemo_mdref_basic`, the value is 3 because there are 3 outputs. The three outputs are in the second element of the Measures array. The `slmetric.metric.AggregationMode` is Max, so the AggregatedValue is 4 which is the number of inputs and outputs to `sldemo_mdref_counter`. The AggregatedMeasures array contains the maximum number of inputs and outputs for a component or subcomponent.

## See Also

`slmetric.Engine` | `slmetric.metric.Result` |  
`slmetric.metric.getAvailableMetrics`

**Introduced in R2016a**

## Attributes property

**Class:** ModelAdvisor.ListViewParameter

**Package:** ModelAdvisor

Attributes to display in Model Advisor Report Explorer

## Values

Cell array

**Default:** {} (empty cell array)

## Description

The `Attributes` property specifies the attributes to display in the center pane of the Model Advisor Results Explorer.

## Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
myLVParam.Attributes = {'FontName'}; % name is default property
```



## CallbackContext property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Specify when to run check

### Values

'PostCompileForCodegen'

'PostCompile'

'None' (default)

### Description

The CallbackContext property specifies the context for checking the model or subsystem.

'None'	No special requirements for the model before checking.
'Postcompile'	The model is compiled and simulated. Use this option for checks that analyze simulated models.
'PostCompileForCodegen'	The model is compiled for code generation, but is not simulated. Use this option to ensure code generation readiness of the model and to analyze both active and inactive variant paths.

## CallbackHandle property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Callback function handle for check

### Values

Function handle.

An empty handle [ ] is the default.

### Description

The `CallbackHandle` property specifies the handle to the check callback function.

# CallbackStyle property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Callback function type

## Values

'StyleOne' (default)

'StyleTwo'

'StyleThree'

'DetailStyle'

## Description

The CallbackStyle property specifies the type of the callback function.

'StyleOne'	Simple check callback function.
'StyleTwo'	Detailed check callback function
'StyleThree'	Check callback function with hyperlinked results
'DetailStyle'	Check callback function for detailed result collections. This style is recommended for authoring Model Advisor checks.

## EmitInputParametersToReport property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Display check input parameters in the Model Advisor report

### Values

'true' (default)

'false'

### Description

The EmitInputParametersToReport property specifies the display of check input parameters in the Model Advisor report.

'true'	Display check input parameters in the Model Advisor report
'false'	Do not display check input parameters in the Model Advisor report

## Data property

**Class:** ModelAdvisor.ListViewParameter

**Package:** ModelAdvisor

Objects in Model Advisor Result Explorer

## Values

Array of Simulink objects

**Default:** [] (empty array)

## Description

The Data property specifies the objects displayed in the Model Advisor Result Explorer.

## Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
myLVParam.Data = get_param(searchResult,'object');
```

## Description property

**Class:** ModelAdvisor.Action

**Package:** ModelAdvisor

Message in **Action** box

## Values

Character vector

**Default:** ' ' (empty character vector)

## Description

The Description property specifies the message displayed in the Action box.

## Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
%Specify a callback function for the action
myAction.setCallbackFcn(@sampleActionCB);
myAction.Name='Fix block fonts';
myAction.Description=...
    'Click the button to update all blocks with specified font';
```

## Description property

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Description of folder

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The `Description` property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

## Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.Description='Sample Factory Group';
```

## Description property

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Description of folder

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The `Description` property provides information about the folder. Details about the folder are displayed in the right pane of the Model Advisor.

## Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
MAG.Description='This is my group';
```



## Description property

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Description of input parameter

## Values

Character vector.

**Default:** '' (empty character vector)

## Description

The `Description` property specifies a description of the input parameter. Details about the check are displayed in the right pane of the Model Advisor.

## Examples

```
% define input parameters
inputParam2 = ModelAdvisor.InputParameter;
inputParam2.Name = 'Standard font size';
inputParam2.Value='12';
inputParam2.Type='String';
inputParam2.Description='sample tooltip';
```

## Description property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Description of task

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The `Description` property is a description of the task that the Model Advisor displays in the **Analysis** box.

When adding checks as tasks, the Model Advisor uses the task `Description` property instead of the check `TitleTips` property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.DisplayName='Example task 1';  
MAT1.Description='This is the first example task.'
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT2.DisplayName='Example task 2';  
MAT2.Description='This is the second example task.'
```

```
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');  
MAT3.DisplayName='Example task 3';  
MAT3.Description='This is the third example task.'
```

# DisplayName property

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Name of folder

## Values

Character vector

**Default:** ' ' (empty character vector)

## Description

The `DisplayName` specifies the name of the folder that is displayed in the Model Advisor.

## Examples

```
% --- sample factory group
rec = ModelAdvisor.FactoryGroup('com.mathworks.sample.factorygroup');
rec.DisplayName='Sample Factory Group';
```

## DisplayName property

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Name of folder

### Values

Character vector

**Default:** ' ' (empty character vector)

### Description

The `DisplayName` specifies the name of the folder that is displayed in the Model Advisor.

### Examples

```
MAG = ModelAdvisor.Group('com.mathworks.sample.GroupSample');  
MAG.DisplayName='My Group';
```

# DisplayName property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Name of task

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The `DisplayName` property specifies the name of the task. The Model Advisor displays each custom task in the tree using the name of the task. Therefore, you should specify a unique name for each task. When you specify the same name for multiple tasks, the Model Advisor generates a warning.

When adding checks as tasks, the Model Advisor uses the task `DisplayName` property instead of the check `Title` property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.DisplayName='Example task with input parameter and auto-fix ability';
```

```
MAT2 = ModelAdvisor.Task('com.mathworks.sample.TaskSample2');  
MAT2.DisplayName='Example task 2';
```

```
MAT3 = ModelAdvisor.Task('com.mathworks.sample.TaskSample3');  
MAT3.DisplayName='Example task 3';
```

## Enable property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Indicate whether user can enable or disable check

### Values

true (default)

false

### Description

The Enable property specifies whether the user can enable or disable the check.

true	Display the check box control
------	-------------------------------

false	Hide the check box control
-------	----------------------------

# Enable property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Indicate if user can enable and disable task

## Values

true (default)

false

## Description

The Enable property specifies whether the user can enable or disable a task.

true (default)

Display the check box control for task

false

Hide the check box control for task

When adding checks as tasks, the Model Advisor uses the task Enable property instead of the check Enable property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Enable = false;
```

## Entries property

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Drop-down list entries

## Values

Depends on the value of the Type property.

## Description

The Entries property is valid only when the Type property is one of the following:

- Enum
- ComboBox
- PushButton

## Examples

```
inputParam3 = ModelAdvisor.InputParameter;  
inputParam3.Name='Valid font';  
inputParam3.Type='Combobox';  
inputParam3.Description='sample tooltip';  
inputParam3.Entries={'Arial', 'Arial Black'};
```



## ID property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Identifier for check

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The ID property specifies a permanent, unique identifier for the check. Note the following about the ID property:

- You must specify this property.
- The value of ID must remain constant.
- The Model Advisor generates an error if ID is not unique.
- Tasks and factory group definitions must refer to checks by ID.

## **ID property**

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Identifier for folder

## **Values**

Character vector

## **Description**

The ID property specifies a permanent, unique identifier for the folder.

---

### **Note**

- You must specify this field.
  - The value of **ID** must remain constant.
  - The Model Advisor generates an error if **ID** is not unique.
  - Group definitions must refer to other groups by **ID**.
-

## ID property

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Identifier for folder

## Values

Character vector

## Description

The ID property specifies a permanent, unique identifier for the folder.

---

### Note

- You must specify this field.
  - The value of **ID** must remain constant.
  - The Model Advisor generates an error if **ID** is not unique.
  - Group definitions must refer to other groups by **ID**.
-

## ID property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Identifier for task

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The ID property specifies a permanent, unique identifier for the task.

---

### Note

- The Model Advisor automatically assigns a unique identifier to ID if you do not specify it.
  - The value of ID must remain constant.
  - The Model Advisor generates an error if ID is not unique.
  - Group definitions must refer to tasks using ID.
- 

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.ID='Task_ID_1234';
```

## LicenseName property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Product license names required to display and run check

### Values

Cell array of product license names

{ }(empty cell array) (default)

### Description

The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

---

**Tip** To find the text for license strings, type `help license` at the MATLAB command line.

---

## LicenseName property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Product license names required to display and run task

### Values

Cell array of product license names

**Default:** {} (empty cell array)

### Description

The `LicenseName` property specifies a cell array of names for product licenses required to display and run the check.

When the Model Advisor starts, it tests whether the product license exists. If you do not meet the license requirements, the Model Advisor does not display the check.

The Model Advisor performs a checkout of the product licenses when you run the custom check. If you do not have the product licenses available, you see an error message that the required license is not available.

If you specify `ModelAdvisor.Check.LicenseName`, the Model Advisor displays the check when the union of both properties is true.

---

**Tip** To find the text for license strings, type `help license` at the MATLAB command line.

---

## ListViewVisible property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Status of **Explore Result** button

### Values

false (default)

true

### Description

The `ListViewVisible` property is a Boolean value that sets the status of the **Explore Result** button.

true	Display the <b>Explore Result</b> button.
false	Hide the <b>Explore Result</b> button.

### Examples

```
% add 'Explore Result' button  
rec.ListViewVisible = true;
```

## **MAObj property**

**Class:** ModelAdvisor.FactoryGroup

**Package:** ModelAdvisor

Model Advisor object

### **Values**

Handle to a Simulink.ModelAdvisor object

### **Description**

The MAObj property specifies a handle to the current Model Advisor object.



## MAObj property

**Class:** ModelAdvisor.Group

**Package:** ModelAdvisor

Model Advisor object

### Values

Handle to Simulink.ModelAdvisor object

### Description

The MAObj property specifies a handle to the current Model Advisor object.

## **MAObj property**

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Model Advisor object

### **Values**

Handle to a Simulink.ModelAdvisor object

### **Description**

The MAObj property specifies the current Model Advisor object.

When adding checks as tasks, the Model Advisor uses the task MAObj property instead of the check MAObj property.

## Name property

**Class:** ModelAdvisor.Action

**Package:** ModelAdvisor

Action button label

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The Name property specifies the label for the action button. This property is required.

## Examples

```
% define action (fix) operation
myAction = ModelAdvisor.Action;
%Specify a callback function for the action
myAction.setCallbackFcn(@sampleActionCB);
myAction.Name='Fix block fonts';
```

## Name property

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Input parameter name

## Values

Character vector.

**Default:** '' (empty character vector)

## Description

The Name property specifies the name of the input parameter in the custom check.

## Examples

```
inputParam2 = ModelAdvisor.InputParameter;  
inputParam2.Name = 'Standard font size';  
inputParam2.Value='12';  
inputParam2.Type='String';  
inputParam2.Description='sample tooltip';
```

## Name property

**Class:** ModelAdvisor.ListViewParameter

**Package:** ModelAdvisor

Drop-down list entry

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The Name property specifies an entry in the **Show** drop-down list in the Model Advisor Result Explorer.

## Examples

```
% define list view parameters
myLVParam = ModelAdvisor.ListViewParameter;
myLVParam.Name = 'Invalid font blocks'; % the name appeared at pull down filter
```

## Result property

**Class:** `ModelAdvisor.Check`

**Package:** `ModelAdvisor`

Results cell array

## Values

Cell array

**Default:** `{}` (empty cell array)

## Description

The `Result` property specifies the cell array for storing the results that are returned by the callback function specified in `CallbackHandle`.

---

**Tip** To set the icon associated with the check, use the `Simulink.ModelAdvisor.setCheckResultStatus` and `setCheckErrorSeverity` methods.

---

# SupportExclusion property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Set to support exclusions

## Values

Boolean value specifying that the check supports exclusions.

true The check supports exclusions.

false (default). The check does not support exclusions.

## Description

The SupportExclusion property specifies whether the check supports exclusions.

'true'	Check supports exclusions.
'false'	Check does not support exclusions.

## Examples

```
% specify that a check supports exclusions
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.SupportExclusion = true;
```

## SupportLibrary property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Set to support library models

### Values

Boolean value specifying that the check supports library models.

true. The check supports library models.

false (default). The check does not support library models.

### Description

The SupportLibrary property specifies whether the check supports library models.

'true'	Check supports library models.
'false'	Check does not support library models.

### Examples

```
% specify that a check supports library models
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');
rec.SupportLibrary = true;
```



## Title property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Name of check

## Values

Character vector

**Default:** '' (empty character vector)

## Description

The `Title` property specifies the name of the check in the Model Advisor. The Model Advisor displays each custom check in the tree using the title of the check. Therefore, you should specify a unique title for each check. When you specify the same title for multiple checks, the Model Advisor generates a warning.

## Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');  
rec.Title = 'Check Simulink block font';
```

## TitleTips property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Description of check

### Values

Character vector

**Default:** '' (empty character vector)

### Description

The `TitleTips` property specifies a description of the check. Details about the check are displayed in the right pane of the Model Advisor.

### Examples

```
rec = ModelAdvisor.Check('com.mathworks.sample.Check1');  
rec.Title = 'Check Simulink block font';  
rec.TitleTips = 'Example style three callback';
```

## Type property

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Input parameter type

## Values

character vector

**Default:** ''

## Description

The Type property specifies the type of input parameter.

Use the Type property with the Value and Entries properties to define input parameters.

Valid values are listed in the following table.

Type	Data Type	Default Value	Description
Bool	Boolean	false	A check box
ComboBox	Cell array	First entry in the list	A drop-down menu <ul style="list-style-type: none"><li>• Use Entries to define the entries in the list.</li><li>• Use Value to indicate a specific entry in the menu or to enter a value not in the list.</li></ul>

Type	Data Type	Default Value	Description
Enum	Cell array	First entry in the list	A drop-down menu <ul style="list-style-type: none"><li>• Use <code>Entries</code> to define the entries in the list.</li><li>• Use <code>Value</code> to indicate a specific entry in the list.</li></ul>
PushButton	N/A	N/A	A button  When you click the button, the callback function specified by <code>Entries</code> is called.
String	Character vector	' '	A text box

## Examples

```
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
```

# Advisor.authoring.DataFile.validate

**Class:** Advisor.authoring.DataFile

**Package:** Advisor.authoring

Validate XML data file used for model configuration check

## Syntax

```
msg = Advisor.authoring.DataFile.validate(dataFile)
```

## Description

`msg = Advisor.authoring.DataFile.validate(dataFile)` validates the syntax of the XML data file used for model configuration checks.

## Input Arguments

`dataFile`            XML data file name (character vector)

## Examples

```
dataFile = 'myDataFile.xml';  
msg = Advisor.authoring.DataFile.validate(dataFile);  
  
if isempty(msg)  
    disp('Data file passed the XSD schema validation.');
```

else  
 disp(msg);  
end

## **See Also**

Advisor.authoring.CustomCheck |

Advisor.authoring.generateConfigurationParameterDataFile

## **Topics**

“Create Check for Model Configuration Parameters”

# Advisor.authoring.CompositeConstraint class

**Package:** Advisor.authoring

Create a Model Advisor constraint that checks for multiple constraints

## Description

Instances of `Advisor.authoring.CompositeConstraint` class contain multiple constraints. Depending on the instance definition, the Model Advisor reports a violation if a model does not meet one or all of the constraints.

## Construction

`cc = Advisor.authoring.CompositeConstraint()` creates an instance of this class

## Properties

### **ConstraintID — IDs of constraints**

character vector | cell array of character vectors

IDs of constraints that compose an `Advisor.authoring.CompositeConstraint` object. This property is read-only. Use the `addConstraintID` method to create a `CompositeConstraint`.

### **CompositeOperator — Operator for specifying whether the Model Advisor reports a violation**

character vector

Use `and` operator to specify that the Model Advisor reports a violation if a model does not meet all of the check constraints. Use `or` operator to specify that the Model Advisor reports a violation if a model does not meet at least one of the check constraints. This property is read/write.

## Methods

`addConstraintID`                      Add constraint to composite constraint

## Examples

### Specify a Composite Constraint

These commands specify a composite constraint for Multi-Port Switch blocks.

Create three `PositiveBlockParameter` constraint objects.

```
c1 = Advisor.authoring.PositiveBlockParameterConstraint();
c1.ID = 'ID_A2';
c1.BlockType = 'MultiPortSwitch';
c1.ParameterName = 'DataPortOrder';
c1.SupportedParameterValues = {'Specify indices'};
c1.ValueOperator = 'eq';

c2 = Advisor.authoring.PositiveBlockParameterConstraint();
c2.ID = 'ID_A3';
c2.BlockType = 'MultiPortSwitch';
c2.ParameterName = 'DataPortForDefault';
c2.SupportedParameterValues = {'Additional data port'};
c2.ValueOperator = 'eq';

c3 = Advisor.authoring.PositiveBlockParameterConstraint();
c3.ID = 'ID_A4';
c3.BlockType = 'MultiPortSwitch';
c3.ParameterName = 'DiagnosticForDefault';
c3.SupportedParameterValues = {'None'};
c3.ValueOperator = 'eq';
```

Use the `addPreRequisiteConstraintID` method to make `c1` a prerequisite to checking constraints `c2` and `c3`.

```
c2.addPreRequisiteConstraintID('ID_A2');
c3.addPreRequisiteConstraintID('ID_A2');
```

Create a composite constraint that specifies that if a Rate Transition block does not meet both constraints `c2` and `c3`, the block is in violation of this check.

```
CC = Advisor.authoring.CompositeConstraint();
CC.addConstraintID('ID_A3');
```



```
CC.addConstraintID('ID_A4');  
CC.CompositeOperator = 'and';
```

## See Also

NegativeBlockParameterConstraint | NegativeBlockTypeConstraint |  
NegativeModelParameterConstraint | PositiveBlockParameterConstraint |  
PositiveBlockTypeConstraint | PositiveModelParameterConstraint

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

# Advisor.authoring.PositiveBlockTypeConstraint class

**Package:** Advisor.authoring

Create a Model Advisor constraint to check for supported block types

## Description

Instances of `Advisor.authoring.PositiveBlockTypeConstraint` class define the only blocks that a model can contain.

## Construction

`constraint = Advisor.authoring.PositiveBlockTypeConstraint()` creates an instance of this class.

## Properties

### ID — Unique identifier

character vector

Unique identifier for the positive block type constraint. This property is read/write.

### SupportedBlockTypes — Supported block types

structure of character vectors

Structure consisting of these fields:

- 'BlockType'
- 'MaskType'

List of supported block types. For more information on the **MaskType** field, see “Mask Editor Overview” (Simulink) and “Mask Parameters” (Simulink). This property is read/write.

## PreRequisiteConstraintIDs — IDs of prerequisite constraints

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read-only.

## Examples

### Specify Supported Block Types

These commands specify that a model contain only Inport, Outport, and Gain blocks and Constant blocks that have a specified mask:

```
c1=Advisor.authoring.PositiveBlockTypeConstraint;  
c1.ID='ID_1';  
s1=struct('BlockType','Inport','MaskType','');  
s2=struct('BlockType','Outport','MaskType','');  
s3=struct('BlockType','Gain','MaskType','');  
s4=struct('BlockType','Constant','MaskType','Stateflow');  
c1.SupportedBlockTypes={s1;s2;s3;s4};
```

## See Also

[CompositeConstraint](#) | [NegativeBlockParameterConstraint](#) | [NegativeBlockTypeConstraint](#) | [NegativeModelParameterConstraint](#) | [PositiveBlockParameterConstraint](#) | [PositiveModelParameterConstraint](#)

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

# Advisor.authoring.NegativeModelParameterConstraint class

**Package:** `Advisor.authoring`

Create a Model Advisor constraint to check for unsupported model parameter values

## Description

Instances of `Advisor.authoring.NegativeModelParameterConstraint` class define unsupported values for specified model parameters.

## Construction

`constraint = Advisor.authoring.NegativeModelParameterConstraint` creates an instance of this class.

## Properties

### **ID — Unique identifier**

character vector

Unique identifier for the negative model parameter constraint. This property is read/write.

### **ParameterName — Name of model parameter**

character vector

Model parameter for which you are specifying a constraint. This property is read/write.

### **UnsupportedParameterValues — Unsupported model parameter values**

cell array of character vectors | cell array of structs | cell array of array of character vectors

List of unsupported values for the model parameter specified by the `ParameterName` field. This property is read/write.

### **PreRequisiteConstraintIDs — IDs of prerequisite constraints**

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read-only.

## **Examples**

### **Specify Unsupported Model Parameter Value**

These commands specify that the **MaxType** parameter does not support a value of zero:

```
c1=Advisor.authoring.NegativeModelParameterConstraint;  
c1.ID='ID_1';  
c1.ParameterName='MaxStep';  
c1.UnsupportedParameterValues={'0'};
```

## **See Also**

[CompositeConstraint](#) | [NegativeBlockParameterConstraint](#) | [NegativeBlockTypeConstraint](#) | [PositiveBlockParameterConstraint](#) | [PositiveBlockTypeConstraint](#) | [PositiveModelParameterConstraint](#)

## **Topics**

“Define Checks for Supported or Unsupported Blocks and Parameters”

### **Introduced in R2018a**

# Advisor.authoring.PositiveModelParameterConstraint class

**Package:** `Advisor.authoring`

Create a Model Advisor constraint to check for supported model parameter values

## Description

Instances of `Advisor.authoring.PositiveModelParameterConstraint` class define supported values for specified model parameters.

## Construction

`constraint = Advisor.authoring.PositiveModelParameterConstraint()`  
creates an instance of this class.

## Properties

### **ID — Unique identifier**

character vector

Unique identifier for the positive model parameter constraint. This property is read/write.

### **ParameterName — Name of model parameter**

character vector

Model parameter for which you are specifying a constraint. This property is read/write.

### **SupportedParameterValues — Supported model parameter values**

cell array of character vectors | cell array of structs | cell array of array of character vectors

List of supported values for the model parameter specified by the `ParameterName` field. This property is read/write.

## PreRequisiteConstraintIDs — IDs of prerequisite constraints

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read-only.

## Examples

### Specify Supported Model Parameter Values

These commands specify that the Solver **Type** model parameter must have a value of `Variable-step`:

```
c1=Advisor.authoring.PositiveModelParameterConstraint;
c1.ID='ID_1';
c1.ParameterName='SolverType';
c1.SupportedParameterValues={'Variable-step'};
```

These commands specify that the **Stop time** model parameter must have a value of `10` or `15`:

```
c1=Advisor.authoring.PositiveModelParameterConstraint;
c1.ID='ID_1';
c1.ParameterName='StopTime';
c1.SupportedParameterValues={'10','15'};
```

For the **ReplacementTypes** model parameter (Embedded Coder Users), these commands specify two sets of supported values for the double and single data types:

```
c1 = Advisor.authoring.PositiveModelParameterConstraint();
c1.ID='ID_2';
c1.ParameterName = 'ReplacementTypes';
s1 = struct('double', 'a', 'single', 'b');
s2 = struct('double', 'c', 'single', 'b');
c1.SupportedParameterValues = {s1, s2};
```

## See Also

[CompositeConstraint](#) | [NegativeBlockParameterConstraint](#) | [NegativeBlockTypeConstraint](#) | [NegativeModelParameterConstraint](#) | [PositiveBlockParameterConstraint](#) | [PositiveBlockTypeConstraint](#)

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**



# Advisor.authoring.NegativeBlockParameterConstraint class

**Package:** Advisor.authoring

Create a Model Advisor constraint to check for unsupported block parameter values

## Description

Instances of `Advisor.authoring.NegativeBlockParameterConstraint` class define unsupported values for specified block parameters.

## Construction

`constraint = Advisor.authoring.NegativeBlockParameterConstraint()`  
creates an instance of this class.

## Properties

### **ID — Unique identifier**

character vector

Unique identifier for the negative block constraint. This property is read/write.

### **BlockType — Block type**

character vector

Block that contains the parameter for which you are specifying a constraint. For a list of block types, see “Block-Specific Parameters” (Simulink). This property is read/write.

### **ParameterName — Name of block parameter**

character vector

Block parameter for which you are specifying a constraint. For a list of block parameters, see “Block-Specific Parameters” (Simulink). This property is read/write.

## **UnsupportedParameterValues — Unsupported block parameter values**

cell array of character vectors | cell array of structs | cell array of array of character vectors

List of unsupported values for the block parameter specified by the `BlockType` and `ParameterName` fields. This property is read/write.

## **ValueOperator — Operator for specifying unsupported parameter values**

character vector

To specify one or more unsupported values, use these operators:

- 'eq'
- 'or'
- 'lt'
- 'gt'
- 'ge'
- 'le'
- 'range'
- 'regex'

This property is read/write. For more information on the regex operator, see `regexp`.

## **PreRequisiteConstraintIDs — IDs of prerequisite constraints**

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read/write.

## **Examples**

### **Specify Unsupported Block Parameter Values**

For a Constant block, these commands specify that one or four values are unsupported for the **Value** parameter:

```
c1=Advisor.authoring.NegativeBlockParameterConstraint;  
c1.ID='ID_1';  
c1.BlockType='Constant';  
c1.ParameterName='Value';  
c1.UnsupportedParameterValues={'1','4'};  
c1.ValueOperator='or';
```

## See Also

CompositeConstraint | NegativeBlockTypeConstraint |  
NegativeModelParameterConstraint | NegativeModelParameterConstraint |  
PositiveBlockParameterConstraint | PositiveBlockTypeConstraint |  
PositiveModelParameterConstraint

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

## Introduced in R2018a

# Advisor.authoring.PositiveBlockParameterConstraint class

**Package:** Advisor.authoring

Create a Model Advisor constraint to check for supported block parameter values

## Description

Instances of `Advisor.authoring.PositiveBlockParameterConstraint` class define supported values for a specified block parameter.

## Construction

`constraint = Advisor.authoring.PositiveBlockParameterConstraint` creates an instance of this class.

## Properties

### **ID — Unique identifier**

character vector

Unique identifier for the positive block parameter constraint. This property is read/write.

### **BlockType — Block type**

character vector

Block that contains the parameter for which you are specifying a constraint. For a list of block types, see “Block-Specific Parameters” (Simulink). This property is read/write.

### **ParameterName — Name of block parameter**

character vector

Block parameter for which you are specifying a constraint. For a list of block parameters, see “Block-Specific Parameters” (Simulink). This property is read/write.

**SupportedParameterValues — Supported block parameter values**

cell array of character vectors | cell array of structs | cell array of array of character vectors

List of supported values for the block parameter specified by the `BlockType` and `ParameterName` fields. This property is read/write.

**ValueOperator — Operator for specifying supported parameter values**

character vector

Use these operators to specify one or more supported values:

- 'eq'
- 'or'
- 'lt'
- 'gt'
- 'ge'
- 'le'
- 'range'
- 'regex'

This property is read/write. For more information on the `regex` operator, see `regexp`.

**PreRequisiteConstraintIDs — IDs of prerequisite constraints**

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read-only.

## Example

### Specify Supported Block Parameter Values

For a Constant block, these commands specify that the **Value** parameter must have values of 2 and 5.

```
c1=Advisor.authoring.PositiveBlockParameterConstraint;  
c1.ID='ID_1';
```

```
c1.BlockType='Constant';  
c1.ParameterName='Value';  
c1.SupportedParameterValues={'[2,5]'};  
c1.ValueOperator='eq';
```

For a Constant block, these commands specify that the **Value** parameter must have a value between 1 and 4.

```
c1=Advisor.authoring.PositiveBlockParameterConstraint;  
c1.ID='ID_1';  
c1.BlockType='Constant';  
c1.ParameterName='Value';  
c1.SupportedParameterValues={'1','4'};  
c1.ValueOperator='range';
```

## See Also

[CompositeConstraint](#) | [NegativeBlockParameterConstraint](#) | [NegativeBlockTypeConstraint](#) | [NegativeModelParameterConstraint](#) | [PositiveBlockTypeConstraint](#) | [PositiveModelParameterConstraint](#)

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

## addPreRequisiteConstraintID

**Class:** `Advisor.authoring.PositiveBlockParameterConstraint`,  
`Advisor.authoring.NegativeBlockParameterConstraint`,  
`Advisor.authoring.PositiveModelParameterConstraint`,  
`Advisor.authoring.NegativeModelParameterConstraint`,  
`Advisor.authoring.PositiveBlockTypeConstraint`,  
`Advisor.authoring.NegativeBlockTypeConstraint`

**Package:** `Advisor.authoring`

Check a prerequisite constraint object before the actual constraint object

### Syntax

```
addPreRequisiteConstraintID(ID_1)
```

### Description

Specify a constraint as a prerequisite to a constraint object. The Model Advisor checks the prerequisite constraint before checking the actual constraint object.

`addPreRequisiteConstraintID(ID_1)` specifies a prerequisite constraint ID `ID_1` that the Model Advisor checks before checking the actual constraint object.

### Input Arguments

**ID\_1 — ID of constraint object**

character vector

To create constraint objects that you can specify as prerequisite constraints, use these classes:

- `Advisor.authoring.PositiveBlockParameterConstraint`
- `Advisor.authoring.NegativeBlockParameterConstraint`

- `Advisor.authoring.PositiveModelParameterConstraint`
- `Advisor.authoring.NegativeModelParameterConstraint`
- `Advisor.authoring.PositiveBlockTypeConstraint`
- `Advisor.authoring.NegativeBlockTypeConstraint`

## Examples

### Specify a Prerequisite Constraint

Specify a constraint on a Gain block. Specify this constraint as a prerequisite for a constraint on a Constant block.

Use the `PositiveBlockParameterConstraint` class to create a constraint on the **Gain** parameter of a Gain block.

```
c1=Advisor.authoring.PositiveBlockParameterConstraint;
c1.ID='ID_1';
c1.BlockType='Gain';
c1.ParameterName='Gain';
c1.SupportedParameterValues={'0','5'};
c1.ValueOperator='range';
```

Use the `NegativeBlockParameterConstraint` class to create a negative constraint on the **Value** parameter of a Constant block.

```
c2=Advisor.authoring.NegativeBlockParameterConstraint;
c2.ID='ID_2';
c2.BlockType='Constant';
c2.ParameterName='Value';
c2.UnsupportedParameterValues={'5'};
c2.ValueOperator='lt';
```

Use the `AddPreRequisiteConstraintID` method to specify the Gain block constraint as a prerequisite to the Constant block constraint.

```
c2.addPreRequisiteConstraintID('ID_1');
```



The Model Advisor does not check the Constant block constraint unless the **Gain** parameter has a value between 0 and 5.

## See Also

NegativeBlockParameterConstraint | NegativeBlockTypeConstraint |  
NegativeModelParameterConstraint | PositiveBlockParameterConstraint |  
PositiveBlockTypeConstraint | PositiveModelParameterConstraint

## Topics

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

## addConstraintID

**Class:** `Advisor.authoring.CompositeConstraint`

**Package:** `Advisor.authoring`

Add constraint to composite constraint

### Syntax

```
addConstraintID(ID_1)
```

### Description

Specify a constraint ID to add to a composite constraint.

`addConstraintID(ID_1)` specifies a constraint ID `ID_1` that the Model Advisor checks as part of a `CompositeConstraint` object.

### Input Arguments

**ID\_1 — ID of constraint object**

character vector

To create root constraint objects that you can specify as part of a composite constraint, use these classes:

- `Advisor.authoring.PositiveBlockParameterConstraint`
- `Advisor.authoring.NegativeBlockParameterConstraint`
- `Advisor.authoring.PositiveBlockTypeConstraint`
- `Advisor.authoring.NegativeBlockTypeConstraint`

### Examples

## Specify a Composite Constraint

These commands specify a composite constraint for Multi-Port Switch blocks:

Create three PositiveBlockParameter constraint objects.

```
c1 = Advisor.authoring.PositiveBlockParameterConstraint();
c1.ID = 'ID_A1';
c1.BlockType = 'MultiPortSwitch';
c1.ParameterName = 'DataPortOrder';
c1.SupportedParameterValues = {'Specify indices'};
c1.ValueOperator = 'eq';

c2 = Advisor.authoring.PositiveBlockParameterConstraint();
c2.ID = 'ID_A2';
c2.BlockType = 'MultiPortSwitch';
c2.ParameterName = 'DataPortForDefault';
c2.SupportedParameterValues = {'Additional data port'};
c2.ValueOperator = 'eq';

c3 = Advisor.authoring.PositiveBlockParameterConstraint();
c3.ID = 'ID_A3';
c3.BlockType = 'MultiPortSwitch';
c3.ParameterName = 'DiagnosticForDefault';
c3.SupportedParameterValues = {'None'};
c3.ValueOperator = 'eq';
```

Use the addPreRequisiteConstraintID method to make c1 a prerequisite to checking constraints c2 and c3.

```
c2.addPreRequisiteConstraintID('ID_1');
c3.addPreRequisiteConstraintID('ID_2');
```

Create a composite constraint that specifies that if a Rate Transition block does not meet both constraints c2 and c3, the block is in violation of this check.

```
CC = Advisor.authoring.CompositeConstraint();
CC.addConstraintID('ID_A2');
CC.addConstraintID('ID_A3');
CC.CompositeOperator = 'and';
```

## See Also

CompositeConstraint | NegativeBlockParameterConstraint |  
 NegativeBlockTypeConstraint | NegativeModelParameterConstraint |  
 PositiveBlockParameterConstraint | PositiveBlockTypeConstraint |  
 PositiveModelParameterConstraint

## **Topics**

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

# Advisor.authoring.NegativeBlockTypeConstraint class

**Package:** Advisor.authoring

Create a Model Advisor constraint to check for unsupported blocks

## Description

Instances of `Advisor.authoring.NegativeBlockTypeConstraint` class define blocks that a model must not contain.

## Construction

`constraint = Advisor.authoring.NegativeBlockTypeConstraint` creates an instance of this class.

## Properties

### **ID — Unique identifier**

character vector

Unique identifier for the block type constraint. This property is read/write.

### **UnsupportedBlockTypes — Unsupported block types**

structure of character vectors

Structure consisting of these fields:

- 'BlockType'
- 'MaskType'

List of unsupported block types. This property is read/write. For more information on the **MaskType** field, see “Mask Editor Overview” (Simulink) and “Mask Parameters” (Simulink).

## **PreRequisiteConstraintIDs — IDs of prerequisite constraints**

cell array of character vectors

IDs of constraints that you specify as prerequisites by using the `addPreRequisiteConstraintID` method. If a prerequisite is not satisfied, the Model Advisor does not check the constraint that has the prerequisite. This property is read-only.

## **Examples**

### **Specify Unsupported Block Types**

These commands specify that a model cannot contain Rate Transition and Integrator blocks and Constant blocks with a specified mask:

```
c1=Advisor.authoring.NegativeBlockTypeConstraint;  
c1.ID='ID_1';  
s1=struct('BlockType','Integrator','MaskType','');  
s2=struct('BlockType','RateTransition','MaskType','');  
s3=struct('BlockType','Constant','MaskType','Stateflow');  
c1.UnsupportedBlockTypes={s1;s2};
```

## **See Also**

[CompositeConstraint](#) | [NegativeBlockParameterConstraint](#) | [NegativeModelParameterConstraint](#) | [PositiveBlockParameterConstraint](#) | [PositiveBlockTypeConstraint](#) | [PositiveModelParameterConstraint](#)

## **Topics**

“Define Checks for Supported or Unsupported Blocks and Parameters”

**Introduced in R2018a**

## Value property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Status of check

## Values

'true' (default)

'false'

## Description

The Value property specifies the initial status of the check. When you use the Value property to specify the initial status of the check, you enable or disable **Run This Check** in the Model Advisor window.

If you want to specify the initial status of a check in the **By Product** folder, before starting Model Advisor, make sure `ModelAdvisor.Preferences.DeselectByProduct` is false.

'true'	Check is enabled
'false'	Check is disabled

## Examples

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```

## **See Also**

ModelAdvisor.Preferences



## Value property

**Class:** ModelAdvisor.InputParameter

**Package:** ModelAdvisor

Value of input parameter

## Values

Depends on the Type property.

## Description

The Value property specifies the initial value of the input parameter. This property is valid only when the Type property is one of the following:

- 'Bool'
- 'String'
- 'Enum'
- 'ComboBox'

## Examples

```
% define input parameters
inputParam1 = ModelAdvisor.InputParameter;
inputParam1.Name = 'Skip font checks.';
inputParam1.Type = 'Bool';
inputParam1.Value = false;
```

## Value property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Status of task

## Values

'true' (default) — Initial status of task is enabled

'false' — Initial status of task is disabled

## Description

The Value property indicates the initial status of a task—whether it is enabled or disabled.

When adding checks as tasks, the Model Advisor uses the task Value property instead of the check Value property.

## Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Value = 'false';
```

## view

View Model Advisor run results for checks

## Syntax

```
view(CheckResultObj)
```

## Description

`view(CheckResultObj)` opens a web browser and displays the results of the check specified by `CheckResultObj`. `CheckResultObj` is a `ModelAdvisor.CheckResult` object returned by `ModelAdvisor.run`.

## Input Arguments

### CheckResultObj

`ModelAdvisor.CheckResult` object which is a part of a `ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

## Examples

View the Model Advisor run results for the first check in the `slvndemo_mdldv_config` configuration file:

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvndemo_mdldv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
         'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% View the 'Identify unconnected...' check result.
view(SysResultObjArray{1}.CheckResultObjs(1))
```

## Alternatives

“View Model Advisor Report”

## See Also

`ModelAdvisor.run` | `ModelAdvisor.summaryReport` | `viewReport`

## Topics

“Checking Systems Programmatically”

“Create a Function for Checking Multiple Systems in Parallel”

“Automate Model Advisor Check Execution”

“Archive and View Model Advisor Run Results”

**Introduced in R2010b**

# viewReport

View Model Advisor run results for systems

## Syntax

```
viewReport(SysResultObjArray{})  
viewReport(SysResultObjArray, 'MA')  
viewReport(SysResultObjArray, 'Cmd')
```

## Description

`viewReport(SysResultObjArray{})` opens the Model Advisor Report for the system specified by `SysResultObjArray{}`. `SysResultObjArray{}` is a `ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

`viewReport(SysResultObjArray, 'MA')` opens the Model Advisor and displays the results of the run for the system specified by `SysResultObjArray`.

`viewReport(SysResultObjArray, 'Cmd')` displays the Model Advisor run summary in the Command Window for the systems specified by `SysResultObjArray`.

## Input Arguments

### `SysResultObjArray{}`

`ModelAdvisor.SystemResult` object returned by `ModelAdvisor.run`.

**Default:**

## Examples

Open the Model Advisor report for `sldemo_auto_climatecontrol/Heater Control`.

```
% Identify Model Advisor configuration file.  
% Create list of models to run.
```

```
fileName = 'slvndemo_mdldv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
        'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Open the Model Advisor report.
viewReport(SysResultObjArray{1})
```

Open Model Advisor and display results for sldemo\_auto\_climatecontrol/Heater Control.

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvndemo_mdldv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
        'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Open the Model Advisor and display results.
viewReport(SysResultObjArray{1}, 'MA')
```

Display results in the Command Window for sldemo\_auto\_climatecontrol/Heater Control.

```
% Identify Model Advisor configuration file.
% Create list of models to run.
fileName = 'slvndemo_mdldv_config.mat';
SysList={'sldemo_auto_climatecontrol/Heater Control',...
        'sldemo_auto_climatecontrol/AC Control'};

% Run the Model Advisor.
SysResultObjArray = ModelAdvisor.run(SysList,'Configuration',fileName);

% Display results in the Command Window.
viewReport(SysResultObjArray{1}, 'Cmd')
```

## Alternatives

- “View Model Advisor Report”
- “View Results in Model Advisor GUI”
- “View Results in Command Window”

## See Also

`ModelAdvisor.run` | `ModelAdvisor.summaryReport` | `view`

## Topics

*"Checking Systems Programmatically"*

*"Create a Function for Checking Multiple Systems in Parallel"*

*"Automate Model Advisor Check Execution"*

*"Archive and View Model Advisor Run Results"*

**Introduced in R2010b**

## Visible property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Indicate to display or hide check

## Values

'true' (default)

'false'

## Description

The Visible property specifies whether the Model Advisor displays the check.

'true'                      Display the check

'false'                     Hide the check

## Examples

```
% hide all checks that do not belong to Demo group
if ~(strcmp(checkCellArray{i}.Group, 'Demo'))
    checkCellArray{i}.Visible = false;
    checkCellArray{i}.Value = false;
end
```



## Visible property

**Class:** ModelAdvisor.Task

**Package:** ModelAdvisor

Indicate to display or hide task

### Values

'true' (default) — Display task in the Model Advisor

'false' — Hide task

### Description

The `Visible` property specifies whether the Model Advisor displays the task.

---

**Caution** When adding checks as tasks, you cannot specify both the task and check `Visible` properties, you must specify one or the other. If you specify both properties, the Model Advisor generates an error when the check `Visible` property is `false`.

---

### Examples

```
MAT1 = ModelAdvisor.Task('com.mathworks.sample.TaskSample1');  
MAT1.Visible = 'false';
```

## slmetric.metric.registerMetric

**Package:** slmetric.metric

Register a custom model metric with the model metric repository

### Syntax

```
[MetricID,err_msg] = slmetric.metric.registerMetric(classname)
```

### Description

[MetricID,err\_msg] = slmetric.metric.registerMetric(classname) register a custom model metric with the model metric repository. The new metric class must be on the MATLAB search path and derived from slmetric.metric.Metric.

### Examples

#### Register a Custom Model Metric with the Model Metric Repository

This example shows how to register a custom model metric.

Create a new metric class, derived from slmetric.metric.Metric, called my\_metric.

```
slmetric.metric.createNewMetricClass('my_metric')
```

Finish the custom model metric implementation and testing.

Register the new custom metric in the model metric repository.

```
[MetricID, err_msg] = slmetric.metric.registerMetric('my_metric');
```

## Input Arguments

### **classname** — Metric class name

character vector

New metric class name.

Data Types: char

## Output Arguments

### **MetricID** — Metric ID

character vector

Unique metric identifier.

Data Types: char

### **err\_msg** — Error message

character vector

If you cannot register a new class, the function returns an error message.

Data Types: char

## See Also

[slmetric.metric.Metric](#) | [slmetric.metric.createNewMetricClass](#) | [slmetric.metric.refresh](#) | [slmetric.metric.unregisterMetric](#)

**Introduced in R2016a**

## **slmetric.metric.unregisterMetric**

**Package:** slmetric.metric

Unregister a custom model metric from the model metric repository

### **Syntax**

```
slmetric.metric.unregisterMetric(MetricID)
```

### **Description**

`slmetric.metric.unregisterMetric(MetricID)` unregister a custom model metric from the model metric repository.

### **Input Arguments**

**MetricID** — Unique metric identifier

character vector

Metric identifier for a custom model metric that you created.

### **See Also**

`slmetric.metric.Metric` | `slmetric.metric.createNewMetricClass` |  
`slmetric.metric.refresh` | `slmetric.metric.registerMetric`

**Introduced in R2016a**

# slmetric.metric.refresh

**Package:** slmetric.metric

Update available model metrics

## Syntax

```
slmetric.metric.refresh()
```

## Description

`slmetric.metric.refresh()` updates available metrics after manual updates to the metric registration file.

## See Also

```
slmetric.metric.Metric | slmetric.metric.createNewMetricClass |  
slmetric.metric.registerMetric | slmetric.metric.unregisterMetric
```

**Introduced in R2016a**

## **slmetric.metric.createNewMetricClass**

**Package:** `slmetric.metric`

Create new metric class for a custom model metric

### **Syntax**

```
slmetric.metric.createNewMetricClass(class_name)
```

### **Description**

`slmetric.metric.createNewMetricClass(class_name)` creates a `slmetric.metric.Metric` class in the current working folder. The new metric class is used to define a custom model metric and supports the following `Advisor.component.Types`:

- `Model`
- `SubSystem`
- `ModelBlock`
- `Chart`
- `MATLABFunction`

### **Examples**

#### **Create a Custom Model Metric Class**

This example shows how to create a new metric class `my_metric`.

Call the function and provide a name for the new metric class:

```
slmetric.metric.createNewMetricClass('my_metric')
```

The function creates a `my_metric.m` file in the current working folder.

```
slmetric.metric.createNewMetricClass('my_metric')
```

The file contains the class definition for `my_metric`, which includes the constructor and an empty metric algorithm method.

```
classdef my_metric < slmetric.metric.Metric
    % my_metric Summary of this metric class goes here
    % Detailed explanation goes here
    properties
    end

    methods
        function this = my_metric()
            this.ID = 'my_metric';
            this.Description = '';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.CompileContext = 'None';
            this.Version = 1;
        end

        function res = algorithm(this, component)
            res = slmetric.metric.Result();
            res.ComponentID = component.ID;
            res.MetricID = this.ID;
            res.Value = 0;
        end
    end
end
```

Write your custom metric algorithm in `algorithm`.

When your custom metric class is working and tested, register your metric using `slmetric.metric.registerMetric`.

## Input Arguments

**class\_name** — Name of the new metric class

character vector

Name of the new metric class you are creating for a custom metric.

Data Types: char

## **See Also**

`Advisor.component.Types` | `slmetric.metric.Metric` |  
`slmetric.metric.registerMetric` | `slmetric.metric.unregisterMetric`

**Introduced in R2016a**



# exportMetrics

**Class:** slmetric.Engine

**Package:** slmetric

Export model metrics

## Syntax

```
exportMetrics(metric_engine,filename)
exportMetrics(metric_engine,filename,filelocation)
```

## Description

Export model metric data to an XML file.

`exportMetrics(metric_engine,filename)` exports an XML filename containing metric data to your current folder.

`exportMetrics(metric_engine,filename,filelocation)` exports an XML filename containing metric data to filelocation.

## Input Arguments

**metric\_engine** — Collects and accesses metric data

slmetric.Engine object

When you call `execute`, `metric_engine` collects metric data for available metrics or for the specified `MetricIDs`. Calling `getMetrics` accesses the collected metric data in `metric_engine`.

**filename** — XML file name

character vector

Name of XML file.

Example: 'MyMetrics.xml'

## **filelocation — File path**

character vector

Path to XML file

Example: 'C:/mywork'

## **Examples**

### **Export Metrics to Current Folder**

This example shows how to export metrics for model vdp to XML file MyMetrics.xml in your current folder.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'vdp', 'RootType', 'Model');

% Generate and collect model metrics
execute(metric_engine);
rc = getMetrics(metric_engine);

% Export metrics to XML file myMetrics.xml
exportMetrics(metric_engine, 'MyMetrics.xml');
```

### **Export Metrics to Specified Location**

This example shows how to export metrics for model vdp to XML file MyMetrics.xml in a specified folder, C:/work.

```
% Create an slmetric.Engine object
metric_engine = slmetric.Engine();

% Specify model for metric analysis
setAnalysisRoot(metric_engine, 'Root', 'vdp', 'RootType', 'Model');
```

```
% Collect model metrics
execute(metric_engine);
rc = getMetrics(metric_engine);

% Export metrics to XML file myMetrics.xml
exportMetrics(metric_engine, 'MyMetrics.xml', 'C:/work');
```

## See Also

`slmetric.metric.ResultCollection` | `slmetric.metric.getAvailableMetrics`

## Topics

“Collect Model Metrics Programmatically”

“Model Metrics” on page 2-419

**Introduced in R2016a**

## clonedetection

Open Clone Detector app

### Syntax

```
clonedetection(model)
```

### Description

`clonedetection(model)` opens the Clone Detector App for a model specified by `model`. If the specified model is not open, this command opens it.

### Examples

#### Open Identify Modeling Clones tool for a model

Open the Clone Detector App for `rtwdemo_preprocessor_subsys` example model:

```
clonedetection('rtwdemo_preprocessor_subsys')
```

### Input Arguments

#### **model** — Model name

character vector

Model name or handle, specified as a character vector.

Data Types: `char`

### See Also

“Enable Component Reuse by Using Clone Detection”

**Introduced in R2017a**

## **slmetric.dashboard.Configuration class**

**Package:** `slmetric.dashboard`

Object containing information on Metrics Dashboard layout and widgets

### **Description**

Instances of `slmetric.dashboard.Configuration` contain information on the layout and types of widgets in the Metric Dashboard.

### **Construction**

Use the `slmetric.dashboard.Configuration` class to specify the layout and types of widgets in the Metrics Dashboard. To create an `slmetric.dashboard.Configuration` object, use the `new` method. Each `slmetric.dashboard.Configuration` object contains one `slmetric.dashboard.Layout` object. Use the methods and properties of the `slmetric.dashboard.Layout` class to customize the widgets and layout of the Metrics Dashboard.

You can modify an existing Metrics Dashboard layout, such as the shipped Metrics Dashboard layout, by using the `getDashboardLayout` method.

### **Properties**

**Name — Configuration object name**

character vector | string scalar

Name of configuration object that you use to specify the Metrics Dashboard layout. This property is read/write.

Data Types: char

**FileName — XML file name**

character vector | string scalar

XML file name that contains information on the current Metrics Dashboard layout. This property is read/write.

Data Types: char

#### **Location — XML file location**

character vector | string scalar

Location of XML file that contains Metrics Dashboard layout. This property is optional and read/write.

Data Types: char

## **Methods**

getDashboardLayout	Create Metrics Dashboard layout object in base workspace
new	Create configuration object for customizing Metrics Dashboard layout
open	Create <code>slmetric.dashboard.Configuration</code> object associated with XML configuration file in the base workspace
openDefaultConfiguration	Return shipping Metrics Dashboard configuration object in base workspace
save	Save contents of <code>slmetric.dashboard.Configuration</code> object to XML file

## **Examples**

### **Create a Configuration Object**

Use the `new` method to create an `slmetric.dashboard.Configuration` object. As an input, specify the name of the XML file that is to contain information on a custom metrics dashboard layout. After you add this information to the configuration object, use the `save` method to save the file.

```
CONF = slmetric.dashboard.Configuration.new('Name','default')
```

```
CONF =
```

Configuration with properties:

```
Name: 'default'  
FileName: ''  
Location: ''
```

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



# slmetric.dashboard.Container class

**Package:** slmetric.dashboard

Widget for holding slmetric.dashboard.Widget and slmetric.dashboard.CustomWidget objects in Metrics Dashboard

## Description

An slmetric.dashboard.Container object holds slmetric.dashboard.Widget and slmetric.dashboard.CustomWidget objects. You can use the slmetric.dashboard.Container methods to specify the container size and border.

For example, the image is of the default Metrics Dashboard layout. This portion of the Metrics Dashboard contains an slmetric.dashboard.Group widget with the title Size. This group contains three slmetric.dashboard.Container widgets. The containers on the left and right each contain one slmetric.dashboard.Widget object. The middle container contains two slmetric.dashboard.Widget objects



## Construction

container = slmetric.dashboard.Container creates a handle to an slmetric.dashboard.Container object.

## Properties

**ShowBorder** — Display a border around the slmetric.dashboard.Container object

0 (default) | logical

If `true`, the `slmetric.dashboard.Container` object has a border around it in the Metrics Dashboard. This property is read/write.

## Type — Widget type

`Container` (default)

This widget type is a container. This property is read-only.

## Methods

<code>addWidget</code>	Add widget to <code>slmetric.dashboard.Container</code> object
<code>getSeparators</code>	Determine whether there are lines on sides of Metrics Dashboard container
<code>getWidgets</code>	Obtain a list of widgets in an <code>slmetric.dashboard.Container</code> object
<code>getWidths</code>	Obtain widths of Metrics Dashboard container
<code>removeWidget</code>	Remove widget from <code>slmetric.dashboard.Container</code> object
<code>setSeparators</code>	Specify lines on Metrics Dashboard container sides
<code>setWidths</code>	Specify multiple widths for Metrics Dashboard container
<code>setMargin</code>	Specify distance from container edge to its contents
<code>getMargin</code>	Obtain distance from container edge to its contents
<code>getPosition</code>	Obtain container position within Metrics Dashboard
<code>setPosition</code>	Set container position within Metrics Dashboard

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);
complianceContainers = getWidgets(complianceGroup);
complianceContainerWidgets = getWidgets(complianceContainers(1));
complianceContainers(1).removeWidget(complianceContainerWidgets(1));
setMetricIDs(complianceContainerWidgets(1),...
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);
misraWidget.Title=('MISRA');
misraWidget.VisualizationType = 'RadialGauge';
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For a model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the play button and run metrics.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## **slmetric.dashboard.CustomWidget class**

**Package:** `slmetric.dashboard`

Object for holding custom Metrics Dashboard widgets

### **Description**

For custom or shipped metrics, use the `slmetric.dashboard.CustomWidget` object to visualize metric data in the Metrics Dashboard. Choose a single value, radial gauge, bar chart, or distribution heat map approach.

### **Construction**

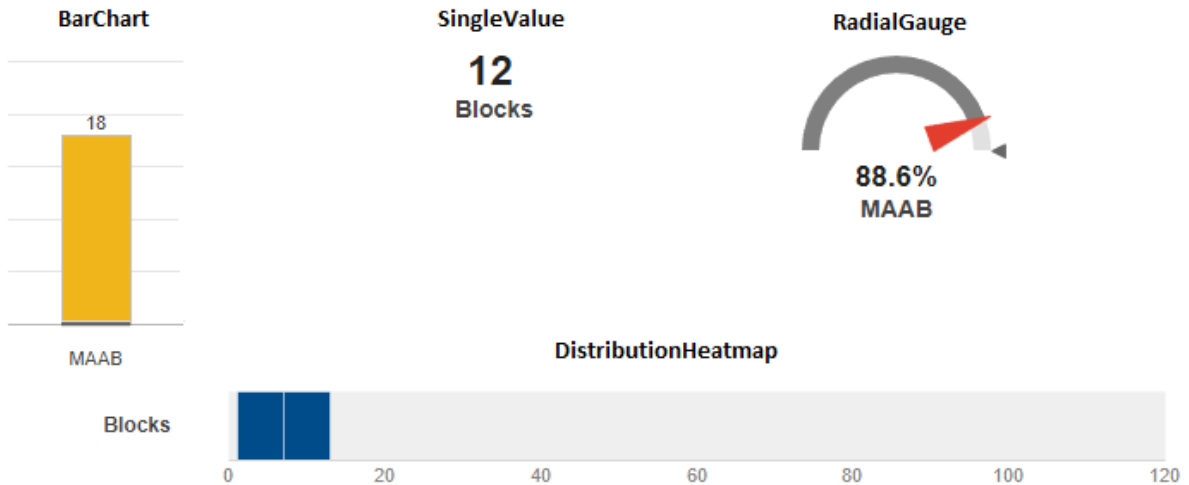
For `slmetric.dashboard.Layout`, `slmetric.dashboard.Container`, or `slmetric.dashboard.Group` objects, use the `addWidget` or `removeWidget` methods to add or remove `slmetric.dashboard.CustomWidget` objects from the Metrics Dashboard. Use `slmetric.dashboard.CustomWidget` methods to specify the widget size.

### **Properties**

#### **VisualizationType — Type of `slmetric.dashboard.CustomWidget` object**

`SingleValue` (default) | `RadialGauge` | `BarChart` | `DistributionHeatmap`

Type of `slmetric.dashboard.CustomWidget` object that you want to add, remove, or modify in the Metrics Dashboard. This property is read/write. Choose from these widget types:



Data Types: char

### Labels — Add labels to custom widget

character vector | string scalar

Add labels to custom widget. This property is only for the BarChart VisualizationType property, so you can add apply labels to each individual bar. This property is read/write.

Data Types: char

### Title — Title of slmetric.dashboard.CustomWidget object

character vector | string scalar

Specify a title for the custom widget. For a radial gauge, there is a 16 character limit. This property is read/write.

Data Types: char

### Type — Type of slmetric.dashboard.CustomWidget object

'Custom'

Type of slmetric.dashboard.CustomWidget object. This property is read-only.

Data Types: char

## Methods

- `getSeparators` Determine whether there are lines on sides of Metrics Dashboard custom widget
- `getWidths` Obtain widths of Metrics Dashboard custom widget
- `setSeparators` Specify lines on Metrics Dashboard custom widget sides
- `setWidths` Specify multiples widths for Metrics Dashboard custom widget
- `setMetricIDs` Set metric identifier for custom Metrics Dashboard widget
- `getHeight` Obtain height of Metrics Dashboard custom widget
- `getMetricIDs` Obtain metric identifier for custom Metrics Dashboard widget
- `getPosition` Obtain custom widget position within Metrics Dashboard
- `setPosition` Set custom widget position within Metrics Dashboard
- `setHeight` Specify height of Metrics Dashboard custom widget

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties
```



```

        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
end

methods
function this = nonvirtualblockcount()
    this.ID = 'nonvirtualblockcount';
    this.Name = 'Nonvirtual Block Count';
    this.Version = 1;
    this.CompileContext = 'None';
    this.Description = 'Algorithm that counts nonvirtual blocks per level.';
    this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
    this.ValueName = 'Nonvirtual Blocks'
    this.ComponentScope = [Advisor.component.Types.Model, ...
        Advisor.component.Types.SubSystem];
    this.AggregationMode = slmetric.AggregationMode.Sum;
    this.AggregateComponentDetails = true;
    this.ResultChecksumCoverage = true;
    this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...

```

```
        'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            case 'Trigger'
                % Virtual when the output port is not present.
                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                    isNonVirtual(n) = false;
                end
            case 'Enable'
```

```

% Virtual unless connected directly to an Output block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Output')
        isNonVirtual(n) = true;
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For a model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

`slmetric.dashboard.getActiveConfiguration` |

`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## slmetric.dashboard.Group class

**Package:** slmetric.dashboard

Widget for holding slmetric.dashboard.Container, slmetric.dashboard.Widget and slmetric.dashboard.CustomButton objects on Metrics Dashboard

### Description

An slmetric.dashboard.Group object can hold slmetric.dashboard.Container, slmetric.dashboard.Widget and slmetric.dashboard.CustomButton objects. You can use the slmetric.dashboard.Group methods and properties to specify the group size, width, and title.

For example, the image is of the default Metrics Dashboard layout. This portion of the Metrics Dashboard contains an slmetric.dashboard.Group widget with the title Size. This group contains three slmetric.dashboard.Container widgets. The containers on the left and right each contain one slmetric.dashboard.Widget object. The middle container contains two slmetric.dashboard.Widget objects.



### Construction

`group = slmetric.dashboard.Group` creates a handle to an slmetric.dashboard.Group object.

### Properties

**Title** — Specify title of metrics group

Character vectorString scalar

Specify title for a group of `slmetric.dashboard.Widget` and `slmetric.dashboard.CustomWidget` objects. The title must summarize the types of widgets in the group. For example, a group with the title `Size` contains widgets pertaining to the size of the model. This property is read/write.

### Type — Widget type

Group (default)

This widget type is a group. This property is read-only.

### ShowBorder — Display a border around the `slmetric.dashboard.Group` object

0 (default) | logical

If `true`, the `slmetric.dashboard.Group` object has a border around it in the Metrics Dashboard. This property is read/write.

## Methods

<code>addWidget</code>	Add widget to <code>slmetric.dashboard.Group</code> object
<code>getSeparators</code>	Determine whether there are lines on sides of Metrics Dashboard group
<code>getWidgets</code>	Obtain a list of widgets in an <code>slmetric.dashboard.Group</code> object
<code>getWidths</code>	Obtain widths of Metrics Dashboard group
<code>removeWidget</code>	Remove widget from <code>slmetric.dashboard.Group</code> object
<code>setSeparators</code>	Specify lines on Metrics Dashboard group sides
<code>setWidths</code>	Specify multiple widths for Metrics Dashboard group
<code>getMargin</code>	Obtain distance from group edge to contents
<code>getPosition</code>	Obtain group position within Metrics Dashboard
<code>setMargin</code>	Specify distance from group edge to its contents
<code>setPosition</code>	Set group position within Metrics Dashboard

## Examples

## Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end  
  
    methods  
        function this = nonvirtualblockcount()  
            this.ID = 'nonvirtualblockcount';  
            this.Name = 'Nonvirtual Block Count';  
            this.Version = 1;  
            this.CompileContext = 'None';  
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';  
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'  
            this.ValueName = 'Nonvirtual Blocks'  
            this.ComponentScope = [Advisor.component.Types.Model, ...  
                Advisor.component.Types.SubSystem];  
            this.AggregationMode = slmetric.AggregationMode.Sum;  
            this.AggregateComponentDetails = true;  
            this.ResultChecksumCoverage = true;  
            this.SupportsResultDetails = true;  
  
        end  
  
        function res = algorithm(this, component)  
            % create a result object for this component  
            res = slmetric.metric.Result();  
  
            % set the component and metric ID
```



```
res.ComponentID = component.ID;
res.MetricID = this.ID;

% Practice
D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
D1.Value=0;
D1.setGroup('Group1','Group1Name');
D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
D2.Value=1;
D2.setGroup('Group1','Group1Name');

% use find_system to get blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
```

```
% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
    'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
    isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
    isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outputport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Outputport')
        isNonVirtual(n) = true;
    end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end
```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

metricsdashboard sf\_car

Click the **All Metrics** button and run all metrics.

### See Also

slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration

### Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.dashboard.Layout class

**Package:** `slmetric.dashboard`

Create object for holding Metrics Dashboard customizations

## Description

Object that holds an array of widget objects. The size, types, and locations of widgets in an `slmetric.dashboard.Layout` object determine the Metrics Dashboard appearance. These are the widget objects:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.Widget`
- `slmetric.dashboard.CustomWidget`

## Construction

For an `slmetric.dashboard.Configuration` object, use the `getDashboardLayout` method to create an `slmetric.dashboard.Layout` object. You can add or remove widgets from this object. You can specify the size and location of these widgets in the Metrics Dashboard. Once you complete your specification, apply the `slmetric.dashboard.Configuration.save` method to save your configuration. Use the `slmetric.dashboard.setActiveConfiguration` function to set the active configuration.

## Methods

<code>addWidget</code>	Add widget to <code>slmetric.dashboard.Layout</code> object
<code>getWidgets</code>	Obtain a list of widgets in an <code>slmetric.dashboard.Layout</code> object
<code>removeWidget</code>	Remove widget from <code>slmetric.dashboard.Layout</code> object

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);
complianceContainers = getWidgets(complianceGroup);
complianceContainerWidgets = getWidgets(complianceContainers(1));
complianceContainers(1).removeWidget(complianceContainerWidgets(1));
setMetricIDs(complianceContainerWidgets(1),...
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the complianceContainers `slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);
misraWidget.Title=('MISRA');
misraWidget.VisualizationType = 'RadialGauge';
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config,'FileName','MetricConfig.xml');
save(conf,'Filename','DashboardConfig.xml');
```

Set the active configurations.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For a model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

### See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

### Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



# slmetric.dashboard.Widget class

**Package:** `slmetric.dashboard`

Object for holding Actual/Potential Reuse, System Interface, or System Info widgets

## Description

You can add, remove, or modify `slmetric.dashboard.Widget` objects in the Metrics Dashboard. The types of `slmetric.dashboard.Widget` objects are **Actual Reuse/Potential Reuse**, **System Interface**, or **System Info** widgets.

## Construction

For `slmetric.dashboard.Layout`, `slmetric.dashboard.Container`, or `slmetric.dashboard.Group` objects, use the `addWidget` or `removeWidget` methods to add or remove `slmetric.dashboard.Widget` objects from the Metrics Dashboard. Use the `slmetric.dashboard.Widget` methods to specify widget size.

## Properties

### Title — Title of `slmetric.dashboard.Widget` object

character vector | string scalar

By default, the `LibraryReuse` widget title is `Library Reuse`, the `SystemInfo` widget title is blank, and the `GlocalInterface` widget title is `System Interface`. This property is read/write.

Data Types: char

### Type — Type of `slmetric.dashboard.Widget` object

`LibraryReuse` | `SystemInfo` | `GlocalInterface`

Type of `slmetric.dashboard.Widget` object that you want to add, remove, or modify in the Metrics Dashboard. This property is read-only.

Data Types: char

## Methods

<code>getSeparators</code>	Determine whether there are lines on sides of Metrics Dashboard widget
<code>getWidths</code>	Obtain widths of Metrics Dashboard widget
<code>setSeparators</code>	Specify lines on Metrics Dashboard widget sides
<code>setWidths</code>	Specify multiple widths for Metrics Dashboard widget
<code>getMetricIDs</code>	Obtain metric identifier for Metrics Dashboard widget
<code>getHeight</code>	Obtain height of Metrics Dashboard widget
<code>getPosition</code>	Obtain widget position within Metrics Dashboard
<code>setHeight</code>	Specify height of Metrics Dashboard widget
<code>setPosition</code>	Set widget position within Metrics Dashboard

## Examples

### Create Metrics Dashboard with Three Widget Objects

Create a Metrics Dashboard with the three types of `slmetric.dashboard.Widget` objects.

To begin, create a new `slmetric.dashboard.Configuration` object.

```
config=slmetric.dashboard.Configuration.new('Name', 'default');
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout=getDashboardLayout(config);
```

Add the three `slmetric.dashboard.Widget` objects to the `slmetric.dashboard.Layout` object.

```
addWidget(layout, 'LibraryReuse');  
addWidget(layout, 'SystemInfo');  
addWidget(layout, 'GlocalInterface');
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(config, 'FileName', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

The Metrics Dashboard contains the three `slmetric.dashboard.Widget` objects.

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getDashboardLayout

**Class:** `slmetric.dashboard.Configuration`

**Package:** `slmetric.dashboard`

Create Metrics Dashboard layout object in base workspace

### Syntax

```
Layout = getDashboardLayout(conf)
```

### Description

`Layout = getDashboardLayout(conf)` creates an `slmetric.dashboard.Layout` object in the base workspace. Use this object to specify the location, size, and types of widgets that are in the Metrics Dashboard.

### Input Arguments

**conf** — Metrics Dashboard configuration object

`slmetric.dashboard.Configuration` object

`slmetric.dashboard.Configuration` object for which to create a custom Metrics Dashboard configuration. By default, an `slmetric.dashboard.Configuration` object holds an empty `slmetric.dashboard.Layout` object.

### Output Arguments

**Layout** — Metrics Dashboard layout object

`slmetric.dashboard.Layout` object

`slmetric.dashboard.Layout` object for which to specify the location, size, and types of widgets in the Metrics Dashboard.

## Examples

### Obtain an `slmetric.dashboard.Layout` Object

Use the new method to create an `slmetric.dashboard.Configuration` object. As an input, specify the name of the XML file that is to contain information on a custom metrics dashboard layout. After you add this information to the configuration object, use the `slmetric.dashboard.Configuration.save` method to save the file.

```
CONF = slmetric.dashboard.Configuration.new('Name', 'default')
```

```
CONF =
```

```
Configuration with properties:
```

```
    Name: 'default'  
  FileName: ''  
  Location: ''
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(CONF);
```

## See Also

```
slmetric.dashboard.Configuration |  
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## **slmetric.dashboard.Configuration.new**

**Class:** `slmetric.dashboard.Configuration`

**Package:** `slmetric.dashboard`

Create configuration object for customizing Metrics Dashboard layout

### **Syntax**

```
Co = slmetric.dashboard.Configuration.new('Name', 'Default')
```

### **Description**

Create an `slmetric.dashboard.Configuration` object for holding customizations pertaining to the Metrics Dashboard layout. Use the `save` command to create and store an associated XML configuration file.

`Co = slmetric.dashboard.Configuration.new('Name', 'Default')` creates a configuration object.

### **Input Arguments**

**Name — Name of configuration object that is tagged in XML file**

character vector | string scalar

Name of configuration object in XML file that contains customizations pertaining to the layout and types of widgets on the Metrics Dashboard.

Data Types: `char`

### **Output Arguments**

**Co — Configuration object**

character vector | string scalar

Name of `slmetric.dashboard.Configuration` object that contains customizations pertaining to the layout and types of widgets on the Metrics Dashboard.

Data Types: `char`

## Examples

### Create a Configuration Object

Use the new method to create an `slmetric.dashboard.Configuration` object. As an input, specify a configuration object name. This name is then associated with a tag in the configuration object XML file. After adding information to the configuration object, use the `save` method to create and store an associated XML file.

```
CONF = slmetric.dashboard.Configuration.new('Name', 'default')
```

```
CONF =
```

```
Configuration with properties:
```

```
    Name: 'default'  
  FileName: ''  
    Location: ''
```

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

**Introduced in R2018b**

## open

**Class:** `slmetric.dashboard.Configuration`

**Package:** `slmetric.dashboard`

Create `slmetric.dashboard.Configuration` object associated with XML configuration file in the base workspace

## Syntax

```
Co = slmetric.dashboard.Configuration.open(  
'FileName','myConfig.xml',... 'Location', pwd,'locale', 'ja_JP')
```

## Description

Reads the contents of the XML file containing the Metrics Dashboard layout into memory and returns the corresponding configuration object. If you modify the contents of the configuration object, invoke the `save` method to write to the XML file.

```
Co = slmetric.dashboard.Configuration.open(  
'FileName','myConfig.xml',... 'Location', pwd,'locale', 'ja_JP')
```

 reads a configuration file.

---

**Note** If you do not supply an input argument, the `slmetric.dashboard.Configuration.open` command reads the contents of the default Metrics Dashboard layout XML file into memory and returns the corresponding `slmetric.dashboard.Configuration` object.

---

## Input Arguments

**FileName — Name of XML file**

character vector | string scalar

Name of XML file containing custom Metrics Dashboard layout and widgets.



Data Types: char

**Location — Folder containing XML file**

character vector | string scalar

Name of folder containing XML file that contains Metrics Dashboard layout. This input argument is optional.

Data Types: char

**Locale — Name of folder containing XML file**

character vector | string scalar

Name of folder containing XML file that contains Metrics Dashboard layout. This input argument is optional.

Data Types: char

## Output Arguments

**Co — Configuration object**

character vector | string scalar

Name of `slmetric.dashboard.Configuration` object that you want to open.

Data Types: char

## Examples

**Access an Existing Configuration Object**

Use the `open` method to add an existing `slmetric.dashboard.Configuration` object to the base workspace. As an input, specify the name of the XML file that contains the information in the configuration object. If you modify the information that this configuration object contains, use the `save` method to save this information to the XML file.

```
CONF = slmetric.dashboard.Configuration.open('FileName',...  
      'myConfig.xml', 'Location', pwd(), 'locale', 'ja_JP');
```

## **See Also**

**Introduced in R2018b**

# openDefaultConfiguration

**Class:** slmetric.dashboard.Configuration

**Package:** slmetric.dashboard

Return shipping Metrics Dashboard configuration object in base workspace

## Syntax

```
DefaultLayout =  
slmetric.dashboard.Configuration.openDefaultConfiguration
```

## Description

DefaultLayout =  
slmetric.dashboard.Configuration.openDefaultConfiguration returns the slmetric.dashboard.Configuration object corresponding to the shipping Metrics Dashboard layout in the base workspace. This object contains information on the size, type, and location of all widgets that ship with the Metrics Dashboard. Use this object to add or remove widgets from the shipping Metrics Dashboard configuration.

## Output Arguments

**DefaultLayout** — **Default Metrics Dashboard configuration object**

slmetric.dashboard.Configuration object

slmetric.dashboard.Configuration object corresponding to the shipping slmetric.dashboard.Configuration object.

## Examples

### **Open shipping Metrics Dashboard Configuration Object**

Use the `openDefaultConfiguration` method to return the shipping `slmetric.dashboard.Configuration` object. If you modify the information that this configuration object contains, use the `slmetric.dashboard.Configuration.save` method to save this information to an XML file.

```
CONF = slmetric.dashboard.Configuration.openDefaultConfiguration
```

### **See Also**

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

### **Topics**

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## save

**Class:** `slmetric.dashboard.Configuration`

**Package:** `slmetric.dashboard`

Save contents of `slmetric.dashboard.Configuration` object to XML file

## Syntax

```
save(Co'FileName','myConfig.xml', ... 'Location',pwd 'locale',  
'ja_JP');
```

## Description

Save the contents of a configuration object to an XML file. The configuration object contains information on a custom Metrics Dashboard layout.

`save(Co'FileName','myConfig.xml', ... 'Location',pwd 'locale', 'ja_JP');` saves the contents of a configuration object to an XML file. The XML file applies your customizations to the Metrics Dashboard.

---

**Note** Do not manually edit the XML file.

---

## Input Arguments

**Co** — **Metrics Dashboard Configuration object**

`slmetric.dashboard.Configuration` object

`slmetric.dashboard.Configuration` object to save to an XML file.

**Filename** — **Name of XML file that contains custom Metrics Dashboard layout**

character vector | string scalar

Name of XML file that contains information on the location and types of widgets that are on the Metrics Dashboard.

Data Types: char

## **Location — Name of folder containing XML file that contains custom Metrics Dashboard layout**

character vector | string scalar

Name of XML file that contains information on the location and types of widgets in the Metrics Dashboard. This input argument is optional.

Data Types: char

## **Locale — Create folder that is to contain XML file**

character vector | string scalar

Name of new folder that is to contain the XML file that contains information on the location and types of widgets in the Metrics Dashboard. If you do not specify a value for `Locale`, Simulink creates the XML file in the folder that you specify with the `Location` property. This input argument is optional.

Data Types: char

## **Serialize a Configuration Object to XML File**

Serialize a configuration object to an XML file.

Use the `save` method to serialize an `slmetric.dashboard.Configuration` object to an XML file. If you modify the information that this configuration object contains, use the `slmetric.dashboard.Configuration.save` method to save information to this file.

```
save(CONF, 'config', 'FileName', 'Configfile.xml', 'Location', pwd)
```

Use the `slmetric.config.setActiveConfiguration` function to specify that the metric engine use this configuration.

```
slmetric.config.setActiveConfiguration('C:\temp\Configfile.xml');
```

## **See Also**

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## **Topics**

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## addWidget

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Add widget to `slmetric.dashboard.Container` object

## Syntax

```
newWidget = addWidget(container, widgetType, num)
```

## Description

`newWidget = addWidget(container, widgetType, num)` adds a widget to an `slmetric.dashboard.Container` object.

## Input Arguments

### **container — Add widget to Metrics Dashboard**

`slmetric.dashboard.Container` object

`slmetric.dashboard.Container` object for which you want to add widgets to customize the Metrics Dashboard layout. This property is read-write.

### **widgetType — Metrics Dashboard widget**

`Group` | `Container` | `SystemInfo` | `GlocalInterface` | `LibraryReuse` | `Custom`

Specify the `Type` property of an `slmetric.dashboard.Container`, `slmetric.dashboard.Widget`, `slmetric.dashboard.Group`, or `slmetric.dashboard.CustomWidget` object.

Data Types: `char`

### **num — Widget placement**

`int`



Placement of widget in container on Metrics Dashboard. Order of widgets in the container proceeds from left to right, and then down in the container.

## Output Arguments

### **newWidget** — New Metrics Dashboard widget

widget object

New widget that you are adding to an `slmetric.dashboard.Container` object on the Metrics Dashboard. You can add these widgets to a container:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomWidget`
- `slmetric.dashboard.Widget`

## Examples

### **Configure Compliance Metrics**

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor

and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);  
complianceContainerWidgets = getWidgets(complianceContainers(1));  
complianceContainers(1).removeWidget(complianceContainerWidgets(1));  
setMetricIDs(complianceContainerWidgets(1),({'mathworks.metrics.ModelAdvisorCompliance  
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers` `slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);  
misraWidget.Title=('MISRA');  
misraWidget.VisualizationType = 'RadialGauge';  
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task');  
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');  
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));  
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

## Introduced in R2018b

## getSeparators

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Determine whether there are lines on sides of Metrics Dashboard container

### Syntax

```
S = getSeparators(container)
```

### Description

`S = getSeparators(container)` returns a structure or an array of structures indicating whether there are lines on the sides of an `slmetric.dashboard.Container` object.

### Input Arguments

**container** — Container for which you want to know whether there are separators

`slmetric.dashboard.Container` object

Determine whether there are separators on the sides of an `slmetric.dashboard.Container` object.

### Output Arguments

**S** — Structure of four fields

Structure|Array of Structures

The output is a structure or an array of structures consisting of these fields:

- `S.top`

- S.bottom
- S.left
- S.right

Each field is empty or has a value of 1 or 0. An empty field indicates that you did not set a value. A value of 1 indicates that there is a line on that container side. A value of 0 indicates that there is not a line on that container side.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getWidgets

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Obtain a list of widgets in an `slmetric.dashboard.Container` object

### Syntax

```
containerList = getWidgets(container)
```

### Description

`containerList = getWidgets(container)` creates an array of objects that are in the `slmetric.dashboard.Container` object. These objects are widgets of the following types:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomWidget`
- `slmetric.dashboard.Widget`

Use the `getWidgets` method to identify widgets that you want to modify or remove from the `slmetric.dashboard.Container` object.

### Input Arguments

**container** — **Object that holds metric dashboard layout customizations**

`slmetric.dashboard.Container` object

`slmetric.dashboard.Container` object for which you want to obtain a list of widgets.

## Output Arguments

**containerList** — Array of objects in `slmetric.dashboard.Container` object  
array of objects

Array of objects in `slmetric.dashboard.Container` object.

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method.

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);  
complianceContainerWidgets = getWidgets(complianceContainers(1));  
complianceContainers(1).removeWidget(complianceContainerWidgets(1));  
setMetricIDs(complianceContainerWidgets(1),...  
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));  
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);  
misraWidget.Title='MISRA';  
misraWidget.VisualizationType = 'RadialGauge';  
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');  
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');  
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.



```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));  
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getWidths

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Obtain widths of Metrics Dashboard container

## Syntax

```
Widths = getWidths(containerName)
```

## Description

`Widths = getWidths(containerName)` returns an `slmetric.dashboard.Width` object array consisting of four enumerations. Use the `slmetric.dashboard.Container.setWidths` method to set the width sizes. You can set between one and four sizes. If you set just one size, the array contains four of the same enumerations. These are the possible enumeration values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the sizes that a container can have as the screen size changes. If the container has one value, the container always has the same size regardless of the screen size. If the container has four different values, the container size can change four times as you maximize or minimize the screen.

## Input Arguments

### **containerName — Metrics Dashboard container**

`slmetric.dashboard.Container` object

Container for which you want to obtain widths

Data Types: char

## Output Arguments

### **Widths — Container widths**

`slmetric.dashboard.Width` enumeration array

`slmetric.dashboard.Width` enumeration array consisting of between one and four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## Examples

### **Configure Compliance Metrics**

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);
```

```

complianceContainerWidgets = getWidgets(complianceContainers(1));
complianceContainers(1).removeWidget(complianceContainerWidgets(1));
setMetricIDs(complianceContainerWidgets(1),...
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));
complianceContainerWidgets(1).Labels={'MISRA'};

```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers slmetric.dashboard.Container` object.

```

misraWidget = complianceContainers(1).addWidget('Custom', 1);
misraWidget.Title=('MISRA');
misraWidget.VisualizationType = 'RadialGauge';
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');
misraWidget.setWidths(slmetric.dashboard.Width.Medium);

```

Save the configuration objects. These commands serialize the API information to XML files.

```

save(config,'FileName','MetricConfig.xml');
save(conf,'Filename','DashboardConfig.xml');

```

Set the active configurations.

```

slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));

```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```

slmetric.dashboard.getActiveConfiguration |
slmetric.dashboard.setActiveConfiguration

```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
 “Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## removeWidget

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Remove widget from `slmetric.dashboard.Container` object

### Syntax

```
removeWidget(container,widget)
```

### Description

`removeWidget(container,widget)` removes a widget from an `slmetric.dashboard.Container` object. You can remove these widgets from the Metrics Dashboard:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomWidget`
- `slmetric.dashboard.Widget`

Use the `getWidgets` method to identify widgets that you want to remove from an `slmetric.dashboard.Container` object.

### Input Arguments

**group** — Remove widget from group in Metrics Dashboard

`slmetric.dashboard.Group` object

Remove widget object from an `slmetric.dashboard.Group` object.

**widget** — Widget that you want to remove from a `slmetric.dashboard.Group` object

index of widget in array

Widget object that you want to remove from an `slmetric.dashboard.Group` object. Apply the `removeWidget` method to the array index containing the widget that you want to remove from the group in the `slmetric.dashboard.Layout` object.

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);  
complianceContainerWidgets = getWidgets(complianceContainers(1));  
complianceContainers(1).removeWidget(complianceContainerWidgets(1));  
setMetricIDs(complianceContainerWidgets(1),...  
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));  
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers` `slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);  
misraWidget.Title=('MISRA');  
misraWidget.VisualizationType = 'RadialGauge';  
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');  
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');  
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.



```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));  
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setSeparators

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Specify lines on Metrics Dashboard container sides

### Syntax

`setSeparators(S)`

### Description

`setSeparators(S)` specifies whether there are lines on the sides of an `slmetric.dashboard.Container` object.

### Input Arguments

**S — Structure of four Boolean values**

Structure | Array of Structures

The input is a structure or an array of four structures consisting of these fields:

- `S.top`
- `S.bottom`
- `S.left`
- `S.right`

Each field must be set to `1` or `0`. A value of `1` indicates that there is a line on that container side. A value of `0` indicates that there is no line on that container side. To indicate that the container sides are always the same even if the screen size changes, you can pass one structure. Passing four structures indicates that the container sides can have different separators as the screen width size changes. Use the `setWidths` method to specify up to four different widths.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## setWidths

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Specify multiple widths for Metrics Dashboard container

## Syntax

```
setWidths(containerName, widths)
```

## Description

`setWidths(containerName, widths)` specifies possible widths that an `slmetric.dashboard.Container` object can have. You can specify up to four different widths. For the input argument `widths`, pass either one value or an array of four values. You can choose from these possible values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the different sizes that a container can have as the screen size changes. If you specify one value, the container always has that value regardless of the screen size. If you specify four different values, the container size can change four times as you maximize and minimize the screen.

## Input Arguments

**containerName** — Container that is to have between one and four widths

`slmetric.dashboard.Container` object

`slmetric.dashboard.Container` object that is to have between one and four widths

**widths — Width array**

character vector | array of character vectors | string scalar | array of string scalars

Specify one or as many as four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

**See Also**

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

**Topics**

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getSeparators

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Determine whether there are lines on sides of Metrics Dashboard custom widget

### Syntax

```
S=getSeparators(customWid)
```

### Description

`S=getSeparators(customWid)` returns a structure or an array of structures indicating whether there are lines on the sides of an `slmetric.dashboard.CustomWidget` object.

### Input Arguments

**customWid** — Custom widget for which you want to know whether there are separators

`slmetric.dashboard.CustomWidget` object

Determine whether there are separators on the sides of an `slmetric.dashboard.CustomWidget` object.

### Output Arguments

**S** — Structure of four fields

Structure | Array of Structures

The output is a structure or an array of structures consisting of these fields:

- `S.top`

- S.bottom
- S.left
- S.right

Each field is empty or has a value of 1 or 0. An empty field indicates that you did not set a value. A value of 1 indicates that there is a line on that custom widget side. A value of 0 indicates that there is no line on that custom widget side.

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the nonvirtualblockcount.m file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
```

```
this.ValueName = 'Nonvirtual Blocks'
this.ComponentScope = [Advisor.component.Types.Model, ...
    Advisor.component.Types.SubSystem];
this.AggregationMode = slmetric.AggregationMode.Sum;
this.AggregateComponentDetails = true;
this.ResultChecksumCoverage = true;
this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
```



```

% or the Treat as atomic unit check box is selected.
if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
    'on')
    isNonVirtual(n) = false;
end
case 'Outport'
% Outport: Virtual when the block resides within
% SubSystem block (conditional or not), and
% does not reside in the root (top-level) Simulink window.
if component.Type ~= Advisor.component.Types.Model
    isNonVirtual(n) = false;
end
case 'Selector'
% Virtual only when Number of input dimensions
% specifies 1 and Index Option specifies Select
% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
    'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
    isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
    isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Outport')
        isNonVirtual(n) = true;
    end
end
end
end
end

```

```
        end
    end

    blocks = blocks(isNonVirtual);

    res.Value = length(blocks);
end
end
end
```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getWidths

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Obtain widths of Metrics Dashboard custom widget

## Syntax

```
Widths=getWidths(customName)
```

## Description

`Widths=getWidths(customName)` returns an `slmetric.dashboard.Width` object array consisting of four enumerations. Use the `slmetric.dashboard.CustomWidget.setWidths` method to set the width sizes. You can set between one and four sizes. If you set just one size, the array contains four of the same enumerations. These are the possible enumeration values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the sizes that a custom widget can have as the screen size changes. If the custom widget has one value, the custom widget always has the same size regardless of the screen size. If the custom widget has four different values, the custom widget size can change four times as you maximize and minimize the screen.

## Input Arguments

### **customName — Metrics Dashboard custom widget**

`slmetric.dashboard.CustomWidget` object

Custom widget for which you want to obtain widths

Data Types: char

## Output Arguments

### **Widths — Custom widget widths**

`slmetric.dashboard.Width` enumeration array

`slmetric.dashboard.Width` enumeration array consisting of between one and four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## Examples

### **Add a Custom Widget to a Group**

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
            this.ValueName = 'Nonvirtual Blocks'
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;

        end

        function res = algorithm(this, component)
            % create a result object for this component
            res = slmetric.metric.Result();

            % set the component and metric ID
            res.ComponentID = component.ID;
            res.MetricID = this.ID;

            % Practice
            D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
            D1.Value=0;
            D1.setGroup('Group1','Group1Name');
            D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
            D2.Value=1;
        end
    end
end
```

```
D2.setGroup('Group1','Group1Name');

% use find_system to get all blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            end
        end
    end
end
```

```
        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end
```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```



Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left = false;
s.right = true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |
slmetric.dashboard.setActiveConfiguration
```

## **Topics**

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# setSeparators

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Specify lines on Metrics Dashboard custom widget sides

## Syntax

```
setSeparators(S)
```

## Description

`setSeparators(S)` specifies whether there are lines on the sides of an `slmetric.dashboard.CustomWidget` object.

## Input Arguments

### **S — Structure of four Boolean values**

Structure | Array of Structures

The input is a structure or an array of structures consisting of these fields:

- `S.top`
- `S.bottom`
- `S.left`
- `S.right`

Each field must be set to 1 or 0. A value of 1 indicates that there is a line on that custom widget side. A value of 0 indicates that there is no line on that custom widget side. To indicate that the custom widget sides are always the same even if the screen size changes, you can pass one structure. Passing four structures indicates that the custom widget sides can have different separators as the screen width size changes. Use the `setWidths` method to specify up to four different widths.

Data Types: char

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end  
  
    methods  
    function this = nonvirtualblockcount()  
        this.ID = 'nonvirtualblockcount';  
        this.Name = 'Nonvirtual Block Count';  
        this.Version = 1;  
        this.CompileContext = 'None';  
        this.Description = 'Algorithm that counts nonvirtual blocks per level.';  
        this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'  
        this.ValueName = 'Nonvirtual Blocks'  
        this.ComponentScope = [Advisor.component.Types.Model, ...  
            Advisor.component.Types.SubSystem];  
        this.AggregationMode = slmetric.AggregationMode.Sum;  
        this.AggregateComponentDetails = true;  
        this.ResultChecksumCoverage = true;  
        this.SupportsResultDetails = true;
```

```

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                        'on')
                        isNonVirtual(n) = false;
                    end
                case 'Outputport'
                    % Outputport: Virtual when the block resides within
                    % SubSystem block (conditional or not), and

```

```
    % does not reside in the root (top-level) Simulink window.
    if component.Type ~= Advisor.component.Types.Model
        isNonVirtual(n) = false;
    end
case 'Selector'
% Virtual only when Number of input dimensions
% specifies 1 and Index Option specifies Select
% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
    'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
    isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
    isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Outport')
        isNonVirtual(n) = true;
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
```

```

    end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

### See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

### Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



## setWidths

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Specify multiples widths for Metrics Dashboard custom widget

## Syntax

```
setWidths(customName, widths)
```

## Description

`setWidths(customName, widths)` specifies possible widths that an `slmetric.dashboard.CustomWidget` object can have. You can specify up to four different widths. For the input argument `widths`, pass either one value or an array of four values. You can choose from these possible values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the different sizes that a custom widget can have as the screen size changes. If you specify one value, the widget always has that value regardless of the screen size. If you specify four different values, the widget size can change four times as you maximize and minimize the screen.

## Input Arguments

**customName** — Custom widget that is to have between one and four widths

`slmetric.dashboard.CustomWidget` object

`slmetric.dashboard.CustomWidget` object that is to have between one and four widths

## **widths — Width array**

`character vector | array of character vectors | string scalar | array of string scalars`

Specify one or as many as four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## **Examples**

### **Add a Custom Widget to a Group**

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
                             'GotoTagVisiblity','Mux','SignalSpecification', ...  
                             'Terminator','Inport'};  
    end
```

```

methods
function this = nonvirtualblockcount()
    this.ID = 'nonvirtualblockcount';
    this.Name = 'Nonvirtual Block Count';
    this.Version = 1;
    this.CompileContext = 'None';
    this.Description = 'Algorithm that counts nonvirtual blocks per level.';
    this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)';
    this.ValueName = 'Nonvirtual Blocks';
    this.ComponentScope = [Advisor.component.Types.Model, ...
        Advisor.component.Types.SubSystem];
    this.AggregationMode = slmetric.AggregationMode.Sum;
    this.AggregateComponentDetails = true;
    this.ResultChecksumCoverage = true;
    this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

```

```
for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            case 'Trigger'
                % Virtual when the output port is not present.
                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                    isNonVirtual(n) = false;
                end
            case 'Enable'
                % Virtual unless connected directly to an Outport block.
                isNonVirtual(n) = false;

                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
```



```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# addWidget

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Add widget to `slmetric.dashboard.Group` object

## Syntax

```
newWidget = addWidget(group,widgetType,num)
```

## Description

`newWidget = addWidget(group,widgetType,num)` adds a widget to an `slmetric.dashboard.Container` object.

## Input Arguments

### **group — Add widget to Metrics Dashboard**

`slmetric.dashboard.Group` object

`slmetric.dashboard.Group` object for which you want to add widgets to customize Metrics Dashboard layout.

### **widgetType — Metrics Dashboard widget**

`Container` | `SystemInfo` | `GlocalInterface` | `LibraryReuse` | `Custom`

Specify the Type of an `slmetric.dashboard.Container`, `slmetric.dashboard.Widget`, `slmetric.dashboard.Group`, or `slmetric.dashboard.CustomWidget` object. This property is read/write.

Data Types: `char`

### **num — Widget placement**

`int`

Placement of widget in group on Metrics Dashboard. Order of widgets in the group proceeds from left to right, and then down in the group.

## Output Arguments

### **newWidget** — New Metrics Dashboard widget

`slmetric.dashboard.Widget` object

New widget that you are adding to an `slmetric.dashboard.Group` object on the Metrics Dashboard. You can add these widgets to a group:

- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomWidget`
- `slmetric.dashboard.Widget`

## Examples

### **Add a Custom Widget to a Group**

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end
```



```
methods
function this = nonvirtualblockcount()
    this.ID = 'nonvirtualblockcount';
    this.Name = 'Nonvirtual Block Count';
    this.Version = 1;
    this.CompileContext = 'None';
    this.Description = 'Algorithm that counts nonvirtual blocks per level.';
    this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)';
    this.ValueName = 'Nonvirtual Blocks';
    this.ComponentScope = [Advisor.component.Types.Model, ...
        Advisor.component.Types.SubSystem];
    this.AggregationMode = slmetric.AggregationMode.Sum;
    this.AggregateComponentDetails = true;
    this.ResultChecksumCoverage = true;
    this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));
```

```
for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            case 'Trigger'
                % Virtual when the output port is not present.
                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                    isNonVirtual(n) = false;
                end
            case 'Enable'
                % Virtual unless connected directly to an Outport block.
                isNonVirtual(n) = false;

                if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
```

```

                                pc = get_param(blocks{n}, 'PortConnectivity');
                                if ~isempty(pc.DstBlock) && ...
                                    strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                                        'Output')
                                    isNonVirtual(n) = true;
                                end
                            end
                        end
                    end
                end
            end
        end
        blocks = blocks(isNonVirtual);
        res.Value = length(blocks);
    end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the play button and run all metrics.

## See Also

[slmetric.dashboard.getActiveConfiguration](#) |  
[slmetric.dashboard.setActiveConfiguration](#)

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

# getSeparators

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Determine whether there are lines on sides of Metrics Dashboard group

## Syntax

```
S = getSeparators(group)
```

## Description

`S = getSeparators(group)` returns a structure or an array of structures indicating whether there are lines on the sides of an `slmetric.dashboard.Group` object.

## Input Arguments

**group** — Group for which you want to know whether there are separators

`slmetric.dashboard.Group` object

Determine whether there are separators on the sides of an `slmetric.dashboard.Group` object.

## Output Arguments

**S** — Structure of four fields

Structure | Array of Structures

The output is a structure or an array of structures consisting of these fields:

- `S.top`
- `S.bottom`

- S.left
- S.right

Each field is empty or has a value of 1 or 0. An empty field indicates that you did not set a value. A value of 1 indicates that there is a line on that group side. A value of 0 indicates that there is no line on that group side.

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end  
  
    methods  
        function this = nonvirtualblockcount()  
            this.ID = 'nonvirtualblockcount';  
            this.Name = 'Nonvirtual Block Count';  
            this.Version = 1;  
            this.CompileContext = 'None';  
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';  
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'  
            this.ValueName = 'Nonvirtual Blocks'  
            this.ComponentScope = [Advisor.component.Types.Model, ...
```

```

        Advisor.component.Types.SubSystem];
this.AggregationMode = slmetric.AggregationMode.Sum;
this.AggregateComponentDetails = true;
this.ResultChecksumCoverage = true;
this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...)

```

```
        'on')
            isNonVirtual(n) = false;
        end
    case 'Outport'
        % Outport: Virtual when the block resides within
        % SubSystem block (conditional or not), and
        % does not reside in the root (top-level) Simulink window.
        if component.Type ~= Advisor.component.Types.Model
            isNonVirtual(n) = false;
        end
    case 'Selector'
        % Virtual only when Number of input dimensions
        % specifies 1 and Index Option specifies Select
        % all, Index vector (dialog), or Starting index (dialog).
        nod = get_param(blocks{n}, 'NumberOfDimensions');
        ios = get_param(blocks{n}, 'IndexOptionArray');

        ios_settings = {'Assign all', 'Index vector (dialog)', ...
            'Starting index (dialog)'};

        if nod == 1 && any(strcmp(ios_settings, ios))
            isNonVirtual(n) = false;
        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end
```



```

        blocks = blocks(isNonVirtual);
    res.Value = length(blocks);
    end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
```

```
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# getWidgets

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Obtain a list of widgets in an `slmetric.dashboard.Group` object

## Syntax

```
groupList = getWidgets(group)
```

## Description

`groupList = getWidgets(group)` creates an array of objects that are in the `slmetric.dashboard.Group` object. These objects are widgets of the following types:

- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomButton`
- `slmetric.dashboard.Widget`

Use the `getWidgets` method to identify widgets that you want to modify or remove from the `slmetric.dashboard.Group` object.

## Input Arguments

**group** — Object that holds metric dashboard layout customizations

`slmetric.dashboard.Group` object

`slmetric.dashboard.Group` object for which you want to obtain a list of widgets.

## Output Arguments

**group** — Array of objects in an `slmetric.dashboard.Group` object

array of objects

Array of widget objects in an `slmetric.dashboard.Group` object.

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end  
  
    methods  
        function this = nonvirtualblockcount()  
            this.ID = 'nonvirtualblockcount';  
            this.Name = 'Nonvirtual Block Count';  
            this.Version = 1;  
            this.CompileContext = 'None';  
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';  
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'  
            this.ValueName = 'Nonvirtual Blocks'  
            this.ComponentScope = [Advisor.component.Types.Model, ...  
                Advisor.component.Types.SubSystem];  
            this.AggregationMode = slmetric.AggregationMode.Sum;  
            this.AggregateComponentDetails = true;  
            this.ResultChecksumCoverage = true;  
            this.SupportsResultDetails = true;
```

```

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                        'on')
                        isNonVirtual(n) = false;
                    end
                case 'Outputport'
                    % Outputport: Virtual when the block resides within
                    % SubSystem block (conditional or not), and

```

```
    % does not reside in the root (top-level) Simulink window.
    if component.Type ~= Advisor.component.Types.Model
        isNonVirtual(n) = false;
    end
case 'Selector'
% Virtual only when Number of input dimensions
% specifies 1 and Index Option specifies Select
% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
    'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
    isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
    isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Outport')
        isNonVirtual(n) = true;
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
```

```

    end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

### See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

### Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**



# getWidths

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Obtain widths of Metrics Dashboard group

## Syntax

`Widths=getWidths(groupName)`

## Description

`Widths=getWidths(groupName)` returns an `slmetric.dashboard.Width` object array consisting of four enumerations. Use the `slmetric.dashboard.Group.setWidths` method to set the width sizes. You can set between one and four sizes. If you set just one size, the array contains four of the same enumerations. These are the possible enumeration values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the sizes that a group can have as the screen size changes. If the group has one value, the group always has the same size regardless of the screen size. If the group has four different values, the group size can change four times as you maximize and minimize the screen.

## Input Arguments

### **groupName — Metrics Dashboard group**

`slmetric.dashboard.Group` object

Group for which you want to obtain widths

Data Types: `char`

## Output Arguments

### **Widths — Group widths**

`slmetric.dashboard.Width` enumeration array

`slmetric.dashboard.Width` enumeration array consisting of between one and four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## Examples

### **Add a Custom Widget to a Group**

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the nonvirtualblockcount.m file.

```

classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
            this.ValueName = 'Nonvirtual Blocks'
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;

        end

        function res = algorithm(this, component)
            % create a result object for this component
            res = slmetric.metric.Result();

            % set the component and metric ID
            res.ComponentID = component.ID;
            res.MetricID = this.ID;

            % Practice
            D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
            D1.Value=0;
            D1.setGroup('Group1','Group1Name');
            D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
            D2.Value=1;
        end
    end
end

```

```
D2.setGroup('Group1','Group1Name');

% use find_system to get all blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
                % all, Index vector (dialog), or Starting index (dialog).
                nod = get_param(blocks{n}, 'NumberOfDimensions');
                ios = get_param(blocks{n}, 'IndexOptionArray');

                ios_settings = {'Assign all', 'Index vector (dialog)', ...
                    'Starting index (dialog)'};

                if nod == 1 && any(strcmp(ios_settings, ios))
                    isNonVirtual(n) = false;
                end
            end
        end
    end
end
```

```

        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left = false;  
s.right = true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## **Topics**

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## removeWidget

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Remove widget from `slmetric.dashboard.Group` object

### Syntax

```
removeWidget(group,widget)
```

### Description

`removeWidget(group,widget)` removes a widget from an `slmetric.dashboard.Group` object. You can remove these widgets from the Metrics Dashboard:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.Custom`
- `slmetric.dashboard.Widget`

Use the `getWidgets` method to identify widgets that you want to remove from an `slmetric.dashboard.Group` object.

### Input Arguments

**group** — Remove widget from group in Metrics Dashboard

`slmetric.dashboard.Group` object

Remove widget object from an `slmetric.dashboard.Group` object.

**widget** — Widget that you want to remove from a `slmetric.dashboard.Group` object

index of widget in array



Widget object that you want to remove from an `slmetric.dashboard.Group` object. Apply the `removeWidget` method to the array index containing the widget that you want to remove from the group in the `slmetric.dashboard.Layout` object.

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)';
            this.ValueName = 'Nonvirtual Blocks';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
```

```
        this.ResultChecksumCoverage = true;
        this.SupportsResultDetails = true;

    end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                        'on')
                        isNonVirtual(n) = false;
                    end
            end
        end
    end
end
```

```

case 'Outport'
    % Outport: Virtual when the block resides within
    % SubSystem block (conditional or not), and
    % does not reside in the root (top-level) Simulink window.
    if component.Type ~= Advisor.component.Types.Model
        isNonVirtual(n) = false;
    end
case 'Selector'
    % Virtual only when Number of input dimensions
    % specifies 1 and Index Option specifies Select
    % all, Index vector (dialog), or Starting index (dialog).
    nod = get_param(blocks{n}, 'NumberOfDimensions');
    ios = get_param(blocks{n}, 'IndexOptionArray');

    ios_settings = {'Assign all', 'Index vector (dialog)', ...
        'Starting index (dialog)'};

    if nod == 1 && any(strcmp(ios_settings, ios))
        isNonVirtual(n) = false;
    end
case 'Trigger'
    % Virtual when the output port is not present.
    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
        isNonVirtual(n) = false;
    end
case 'Enable'
    % Virtual unless connected directly to an Outport block.
    isNonVirtual(n) = false;

    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
        pc = get_param(blocks{n}, 'PortConnectivity');

        if ~isempty(pc.DstBlock) && ...
            strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                'Outport')
            isNonVirtual(n) = true;
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

```

```
        res.Value = length(blocks);
    end
end
end
```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setSeparators

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Specify lines on Metrics Dashboard group sides

### Syntax

`setSeparators(S)`

### Description

`setSeparators(S)` specifies whether there are lines on the sides of an `slmetric.dashboard.Group` object.

### Input Arguments

#### **S — Structure of four Boolean values**

Structure | Array of Structures

The input is a structure array consisting of these fields:

- `S.top`
- `S.bottom`
- `S.left`
- `S.right`

Each field must be set to `1` or `0`. A value of `1` indicates that there is a line on that group side. A value of `0` indicates that there is no line on that group side. To indicate that the group sides are always the same even if the screen size changes, you can pass one structure. Passing four structures indicates that the group sides can have different separators as the screen width size changes. Use the `setWidths` method to specify up to four different widths.

Data Types: char

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the nonvirtualblockcount.m file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;
        end
    end
end
```

```
function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                        'on')
                        isNonVirtual(n) = false;
                    end
                case 'Outport'
                    % Outport: Virtual when the block resides within
                    % SubSystem block (conditional or not), and
                    % does not reside in the root (top-level) Simulink window.
                    if component.Type ~= Advisor.component.Types.Model
```



```

        isNonVirtual(n) = false;
    end
    case 'Selector'
        % Virtual only when Number of input dimensions
        % specifies 1 and Index Option specifies Select
        % all, Index vector (dialog), or Starting index (dialog).
        nod = get_param(blocks{n}, 'NumberOfDimensions');
        ios = get_param(blocks{n}, 'IndexOptionArray');

        ios_settings = {'Assign all', 'Index vector (dialog)', ...
            'Starting index (dialog)'};

        if nod == 1 && any(strcmp(ios_settings, ios))
            isNonVirtual(n) = false;
        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf,'Filename','DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setWidths

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Specify multiple widths for Metrics Dashboard group

## Syntax

```
setWidths(groupName, widths)
```

## Description

`setWidths(groupName, widths)` specifies possible widths for an `slmetric.dashboard.Group` object. You can specify up to four different widths. For the input argument `widths`, pass either one value or an array of four values. You can choose from these possible values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the different sizes that a group can have as the screen size changes. If you specify one value, the group always has that value regardless of the screen size. If you specify four different values, the container size can change four times as you maximize and minimize the screen.

## Input Arguments

**groupName** — Group that is to have between one and four widths

`slmetric.dashboard.Group` object

`slmetric.dashboard.Container` object that is to have between one and four widths

### **widths — Width array**

character vector | array of character vectors | string scalar | array of string scalars

Specify one or as many as four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## **Examples**

### **Add a Custom Widget to a Group**

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisibility','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end
```

```
methods
function this = nonvirtualblockcount()
    this.ID = 'nonvirtualblockcount';
    this.Name = 'Nonvirtual Block Count';
    this.Version = 1;
    this.CompileContext = 'None';
    this.Description = 'Algorithm that counts nonvirtual blocks per level.';
    this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
    this.ValueName = 'Nonvirtual Blocks'
    this.ComponentScope = [Advisor.component.Types.Model, ...
        Advisor.component.Types.SubSystem];
    this.AggregationMode = slmetric.AggregationMode.Sum;
    this.AggregateComponentDetails = true;
    this.ResultChecksumCoverage = true;
    this.SupportsResultDetails = true;

end

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get all blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
```

```

blockType = get_param(blocks{n}, 'BlockType');

if any(strcmp(this.VirtualBlockTypes, blockType))
    isNonVirtual(n) = false;
else
    switch blockType
        case 'SubSystem'
            % Virtual unless the block is conditionally executed
            % or the Treat as atomic unit check box is selected.
            if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                'on')
                isNonVirtual(n) = false;
            end
        case 'Outport'
            % Outport: Virtual when the block resides within
            % SubSystem block (conditional or not), and
            % does not reside in the root (top-level) Simulink window.
            if component.Type ~= Advisor.component.Types.Model
                isNonVirtual(n) = false;
            end
        case 'Selector'
            % Virtual only when Number of input dimensions
            % specifies 1 and Index Option specifies Select
            % all, Index vector (dialog), or Starting index (dialog).
            nod = get_param(blocks{n}, 'NumberOfDimensions');
            ios = get_param(blocks{n}, 'IndexOptionArray');

            ios_settings = {'Assign all', 'Index vector (dialog)', ...
                'Starting index (dialog)'};

            if nod == 1 && any(strcmp(ios_settings, ios))
                isNonVirtual(n) = false;
            end
        case 'Trigger'
            % Virtual when the output port is not present.
            if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
                isNonVirtual(n) = false;
            end
        case 'Enable'
            % Virtual unless connected directly to an Outport block.
            isNonVirtual(n) = false;

            if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
                pc = get_param(blocks{n}, 'PortConnectivity');
            end
    end
end

```

```
                if ~isempty(pc.DstBlock) && ...
                    strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                        'Output')
                    isNonVirtual(n) = true;
                end
            end
        end
    end
    blocks = blocks(isNonVirtual);
    res.Value = length(blocks);
end
end
end
```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);
sizeGroupWidgets = sizeGroup.getWidgets();
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.



```
newWidget = sizeGroup.addWidget('Custom', 1);
newWidget.Title=('Nonvirtual Block Count');
newWidget.setMetricIDs('nonvirtualblockcount');
newWidget.setWidths(slmetric.dashboard.Width.Medium);
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;
s.bottom = false;
s.left= false;
s.right= true;
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## addWidget

**Class:** `slmetric.dashboard.Layout`

**Package:** `slmetric.dashboard`

Add widget to `slmetric.dashboard.Layout` object

## Syntax

```
newWidget = addWidget(dashboardLayout, widgetType, num)
```

## Description

`newWidget = addWidget(dashboardLayout, widgetType, num)` adds a widget to an `slmetric.dashboard.Layout` object.

## Input Arguments

### **dashboardLayout — Add widget to Metrics Dashboard**

`slmetric.dashboard.Layout` object

`slmetric.dashboard.Layout` object for which you want to add widgets to customize Metrics Dashboard layout.

### **widgetType — Metrics Dashboard widget**

`Group` | `Container` | `SystemInfo` | `GlocalInterface` | `LibraryReuse` | `Custom`

Specify the `Type` property of an `slmetric.dashboard.Container`, `slmetric.dashboard.Widget`, `slmetric.dashboard.Group`, or `slmetric.dashboard.CustomWidget` object.

Data Types: `char`

### **num — Widget placement**

`int`

Placement of widget on Metrics Dashboard. Order of widgets proceeds from left to right, and then down.

## Output Arguments

### **newWidget** — New Metrics Dashboard widget

`slmetric.dashboard.Widget` object

New widget that you are adding to Metrics Dashboard. Choose from one of these widgets:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.Widget`
- `slmetric.dashboard.Custom`

## Examples

### **Configure Compliance Metrics**

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);  
complianceContainerWidgets = getWidgets(complianceContainers(1));  
complianceContainers(1).removeWidget(complianceContainerWidgets(1));  
setMetricIDs(complianceContainerWidgets(1),...  
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));  
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers` `slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);  
misraWidget.Title=('MISRA');  
misraWidget.VisualizationType = 'RadialGauge';
```

```
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');  
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');  
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));  
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getWidgets

**Class:** `slmetric.dashboard.Layout`

**Package:** `slmetric.dashboard`

Obtain a list of widgets in an `slmetric.dashboard.Layout` object

## Syntax

```
Layout = getWidgets(dashboardLayout)
```

## Description

`Layout = getWidgets(dashboardLayout)` creates an array of objects that are in the `slmetric.dashboard.Layout` object. These objects are widgets of the following types:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.Widget`
- `slmetric.dashboard.CustomWidget`

Use the `getWidgets` method to identify widgets that you want to modify or remove from the `slmetric.dashboard.Layout` object.

## Input Arguments

**dashboardLayout** — **Object that holds Metrics Dashboard layout customizations**

`slmetric.dashboard.Layout` object

`slmetric.dashboard.Layout` object for which you want to obtain a list of widgets.

Data Types: `char`

## Output Arguments

### Layout — Array of objects in `slmetric.dashboard.Layout` object

array of objects

Array of objects in `slmetric.dashboard.Layout` object.

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method.

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);  
complianceContainers = getWidgets(complianceGroup);  
complianceContainerWidgets = getWidgets(complianceContainers(1));  
complianceContainers(1).removeWidget(complianceContainerWidgets(1));  
setMetricIDs(complianceContainerWidgets(1),...  
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));  
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the `complianceContainers slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);  
misraWidget.Title='MISRA';  
misraWidget.VisualizationType = 'RadialGauge';  
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');  
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config, 'FileName', 'MetricConfig.xml');  
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configurations.



```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));  
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## removeWidget

**Class:** `slmetric.dashboard.Layout`

**Package:** `slmetric.dashboard`

Remove widget from `slmetric.dashboard.Layout` object

### Syntax

```
removeWidget(dashboardLayout,widget in array)
```

### Description

`removeWidget(dashboardLayout,widget in array)` removes a widget from an `slmetric.dashboard.Layout` object. You can remove these widgets from the Metrics Dashboard:

- `slmetric.dashboard.Group`
- `slmetric.dashboard.Container`
- `slmetric.dashboard.CustomWidget`
- `slmetric.dashboard.Widget`

Use the `getWidgets` method to identify widgets that you want to remove from a `slmetric.dashboard.Layout` object.

### Input Arguments

**dashboardLayout** — Remove widget from Metrics Dashboard

`slmetric.dashboard.Layout` object

Remove widget object from an `slmetric.dashboard.Layout` object.

**widget** — Widget to remove from an `slmetrics.dashboard.Layout` object

index of widget in array

Widget object that you want to remove from an `slmetric.dashboard.layout` object. Use the `getWidgets` method to return an array of widgets in the `slmetrics.dashboard.layout` object. Apply the `removeWidget` method to the array index containing the widget that you want to remove from the Metrics Dashboard.

## Examples

### Configure Compliance Metrics

You can use the Metrics Dashboard and metric APIs to obtain compliance and issues metric data on your Model Advisor configuration. To set up your Model Advisor configuration, see “Organize Checks and Folders Using the Model Advisor Configuration Editor”. You can also use an existing check group such as the MISRA checks. After you have set up your Model Advisor configuration, follow these steps to specify the check groups for which you want to obtain compliance and issues metric data:

Open the default configuration:

```
config=slmetric.config.Configuration.open()
```

Specify a metric family ID that you associate with those check groups:

```
famParamID = 'ModelAdvisorStandard';
```

Create a cell array consisting of the Check Group IDs that correspond to the check groups. Obtain a Check Group ID by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The folder contains a **Check Group ID** parameter.

```
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
```

The previous cell array specifies MAAB, High-Integrity, and MISRA check groups. The values `maab` and `hisl_do178` correspond to a subset of MAAB and High-Integrity System checks. To include all checks, specify the value for the **Check Group ID** parameter from the Model Advisor Configuration Editor.

To set up the configuration, pass the `values` cell array into the `setMetricFamilyParameterValues` method .

```
setMetricFamilyParameterValues(config, famParamID, values);
```

Point the **High Integrity Compliance** and **High Integrity Check Issues** widgets to the MISRA check group. To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object `conf`.

```
layout = getDashboardLayout(conf);
```

Obtain the widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Obtain the compliance group from the layout. This group contains two containers. The first container contains the High Integrity and MAAB Compliance and Check Issues widgets. Remove the **High Integrity Compliance** widget.

```
complianceGroup = layoutWidget(3);
complianceContainers = getWidgets(complianceGroup);
complianceContainerWidgets = getWidgets(complianceContainers(1));
complianceContainers(1).removeWidget(complianceContainerWidgets(1));
setMetricIDs(complianceContainerWidgets(1),...
({'mathworks.metrics.ModelAdvisorCompliance._SYSTEM_By Task_misra_c'}));
complianceContainerWidgets(1).Labels={'MISRA'};
```

Add a custom widget for visualizing MISRA check issues metrics to the complianceContainers `slmetric.dashboard.Container` object.

```
misraWidget = complianceContainers(1).addWidget('Custom', 1);
misraWidget.Title=('MISRA');
misraWidget.VisualizationType = 'RadialGauge';
misraWidget.setMetricIDs('mathworks.metrics.ModelAdvisorCheckCompliance._SYSTEM_By Task_misra_c');
misraWidget.setWidths(slmetric.dashboard.Width.Medium);
```

Save the configuration objects. These commands serialize the API information to XML files.

```
save(config,'FileName','MetricConfig.xml');
save(conf,'Filename','DashboardConfig.xml');
```

Set the active configurations.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getSeparators

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Determine whether there are lines on sides of Metrics Dashboard widget

### Syntax

```
S=getSeparators(widget)
```

### Description

`S=getSeparators(widget)` returns a structure or an array of structures indicating whether there are lines on the sides of an `slmetric.dashboard.Widget` object.

### Input Arguments

**widget** — **Widget for which you want to know whether there are separators**

`slmetric.dashboard.Widget` object

Determine whether there are separators on the sides of an `slmetric.dashboard.Widget` object.

### Output Arguments

**S** — **Structure of four fields**

Structure | Array of Structures

The structure array contains these fields:

- `S.top`
- `S.bottom`

- S.left
- S.right

Each field is empty or has a value of 1 or 0. An empty field indicates that you did not set a value. A value of 1 indicates that there is a line on that widget side. A value of 0 indicates that there is no line on that widget side.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getWidths

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Obtain widths of Metrics Dashboard widget

## Syntax

`Widths=getWidths(widgetName)`

## Description

`Widths=getWidths(widgetName)` returns an `slmetric.dashboard.Width` object array consisting of four enumerations. Use the `slmetric.dashboard.Widgets.setWidths` method to set the width sizes. You can set between one and four sizes. If you set just one size, the array contains four of the same enumerations. These are the possible enumeration values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the sizes that a widget can have as the screen size changes. If the widget has one value, the widget always has the same size regardless of the screen size. If the widget has four different values, the widget size can change four times as you maximize and minimize the screen.



## Input Arguments

### **widgetName — Metrics Dashboard widget**

`slmetric.dashboard.Widget` object

Widget for which you want to obtain widths.

Data Types: `char`

## Output Arguments

### **Widths — Widget widths**

`slmetric.dashboard.Width` enumeration array

`slmetric.dashboard.Width` enumeration array consisting of between one and four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setSeparators

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Specify lines on Metrics Dashboard widget sides

### Syntax

`setSeparators(S)`

### Description

`setSeparators(S)` specifies whether there are lines on the sides of an `slmetric.dashboard.Widget` object.

### Input Arguments

#### **S — Structure of four Boolean values**

Structure array

The input is a structure array consisting of these fields:

- `S.top`
- `S.bottom`
- `S.left`
- `S.right`

Each field must be set to `1` or `0`. A value of `1` indicates that there is a line on that widget side. A value of `0` indicates that there is no line on that widget side. To indicate that the widget sides are always the same even if the screen size changes, you can pass one structure. Passing four structures indicates that the widget sides can have different separators as the screen width size changes. Use the `setWidths` method to specify up to four different widths.

Data Types: char

## **See Also**

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## **Topics**

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setWidths

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Specify multiple widths for Metrics Dashboard widget

## Syntax

```
setWidths(widgetName, widths)
```

## Description

`setWidths(widgetName, widths)` specifies possible widths that an `slmetric.dashboard.Widget` object can have. You can specify up to four different widths. For the input argument `widths`, pass either one value or an array of four values. You can choose from these possible values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

These values correspond to the different sizes that a widget can have as the screen size changes. If you specify one value, the widget always has that value regardless of the screen size. If you specify four different values, the widget size can change four times as you maximize and minimize the screen.

## Input Arguments

**widgetName** — **Widget that is to have between one and four widths**

`slmetric.dashboard.Widget` object

`slmetric.dashboard.Widget` object that is to have between one and four widths

**widths — Width array**

`character vector | array of character vectors | string scalar | array of string scalars`

Specify one or as many as four of these values:

- `slmetric.dashboard.Width.ExtraSmall`
- `slmetric.dashboard.Width.Small`
- `slmetric.dashboard.Width.Medium`
- `slmetric.dashboard.Width.Large`
- `slmetric.dashboard.Width.XLarge`
- `slmetric.dashboard.Width.XXLarge`

**See Also**

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

**Topics**

*“Collect Model Metric Data by Using the Metrics Dashboard”*

*“Customize Metrics Dashboard Layout and Functionality”*

**Introduced in R2018b**

# slmetric.dashboard.getActiveConfiguration

**Package:** slmetric.dashboard

Obtain file path and name of XML file containing active Metrics Dashboard layout

## Syntax

Path = slmetric.dashboard.getActiveConfiguration

## Description

Path = slmetric.dashboard.getActiveConfiguration returns the file path and name of the active Metrics Dashboard layout XML file. This file contains information on the location, size, and types of widgets in the Metrics Dashboard.

## Examples

### Get Default Metrics Dashboard Layout

At the MATLAB command line, enter this command to get the active metric dashboard layout:

```
slmetric.dashboard.getActiveConfiguration();
```

## Output Arguments

### Path — File path to XML file

character vector | string scalar

Full file path to folder containing XML file that contains the active Metrics Dashboard layout.

---

**Note** Passing an empty string to the `slmetric.dashboard.setActionConfiguration` function (that is, `slmetric.dashboard.setActiveConfiguration('')`), resets the configuration to the default, which is the shipping configuration. If you then enter the `slmetric.dashboard.getActiveConfiguration` method, the method returns an empty array.

---

Data Types: `char`

## See Also

`slmetric.dashboard.Configuration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# slmetric.dashboard.setActiveConfiguration

**Package:** slmetric.dashboard

Activate custom metric dashboard layout

## Syntax

```
slmetric.dashboard.setActiveConfiguration(fullfile)
```

## Description

`slmetric.dashboard.setActiveConfiguration(fullfile)` sets a custom Metrics Dashboard layout as the default configuration. When you collect metric data by using the Metrics Dashboard, the metric engine uses this custom layout.

---

**Note** Passing an empty string to this function (that is, `slmetric.dashboard.setActiveConfiguration('')`), resets the configuration to the default, shipping configuration.

---

## Examples

### Activate Custom Configuration

At the MATLAB command line, enter this command to set the active metric configuration:

```
slmetric.config.setActiveConfiguration('C:\temp\MyConfig.xml');
```

## Input Arguments

**fullfile** — File path to XML file

character vector | string scalar



Full file path to folder containing XML file that contains Metrics Dashboard custom configurations.

Example: 'C:\temp\MyConfig.xml'

Data Types: char

## See Also

[slmetric.config.Configuration](#) | [slmetric.config.getActiveConfiguration](#)

## External Websites

["Collect and Explore Metric Data by Using the Metrics Dashboard"](#)

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

**Introduced in R2018b**

## setMargin

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Specify distance from container edge to its contents

### Syntax

```
pixels = setMargin(Container,px)
```

### Description

`pixels = setMargin(Container,px)` specifies how far in pixels the edges of an `slmetric.dashboard.Container` object is from the widgets that it contains.

### Input Arguments

#### **Container — Metrics Dashboard container**

`slmetric.dashboard.Container` object

The `slmetric.dashboard.Container` object for which you are specifying margin size in pixels.

Data Types: `char`

#### **px — Container margins**

`character vector` | `string scalar`

Margin distance from container contents in pixels.

Example: `'40 px'`

Data Types: `char`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## setMetricIDs

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Set metric identifier for custom Metrics Dashboard widget

### Syntax

```
setMetricIDs(CustomWidget, metricID)
```

### Description

`setMetricIDs(CustomWidget, metricID)` assigns a metric identifier to an `slmetric.dashboard.CustomWidget` object.

### Input Arguments

**CustomWidget — Custom widget object**

`slmetric.dashboard.CustomWidget` object

`slmetric.dashboard.CustomWidget` object for which you want to assign a metric identifier. The `slmetric.dashboard.CustomWidget` object is the means of visualizing metric data for the metric identifier.

Data Types: `char`

**metricID — Metric identifier**

`character vector` `string` `scalar`

Metric identifier associated with an `slmetric.dashboard.CustomWidget` object.

### Examples

## Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
            this.ValueName = 'Nonvirtual Blocks'
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;
        end

        function res = algorithm(this, component)
            % create a result object for this component
            res = slmetric.metric.Result();

            % set the component and metric ID
```

```
res.ComponentID = component.ID;
res.MetricID = this.ID;

% Practice
D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
D1.Value=0;
D1.setGroup('Group1','Group1Name');
D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
D2.Value=1;
D2.setGroup('Group1','Group1Name');

% use find_system to get all blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            case 'Selector'
                % Virtual only when Number of input dimensions
                % specifies 1 and Index Option specifies Select
```

```

% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
    'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
    isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
    isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
    pc = get_param(blocks{n}, 'PortConnectivity');

    if ~isempty(pc.DstBlock) && ...
        strcmp(get_param(pc.DstBlock, 'BlockType'), ...
            'Outport')
        isNonVirtual(n) = true;
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For a model, open the Metrics Dashboard.



metricsdashboard sf\_car

Click the **All Metrics** button and run all metrics.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getMetricIDs

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Obtain metric identifier for Metrics Dashboard widget

### Syntax

```
metricID = getMetricIDs(widget)
```

### Description

`metricID = getMetricIDs(widget)` returns the metric identifier for an `slmetric.dashboard.Widget` object.

### Input Arguments

**widget — Widget object**

`slmetric.dashboard.Widget` object

`slmetric.dashboard.Widget` object for which you want to obtain the associated metric identifier. The `slmetric.dashboard.Widget` object is the means of visualizing metric data for the metric identifier.

Data Types: `char`

### Output Arguments

**metricID — Metric identifier**

`character vector`  
`string` `scalar`

Metric identifier associated with an `slmetric.dashboard.Widget` object.

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getMetricFamilyParameterValues

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Obtain metric family Check Group IDs

### Syntax

```
ArraysValue = getMetricFamilyParameterValues(config,...  
'ModelAdvisorStandard')
```

### Description

For an `slmetric.config.Configuration` object, use the `ArraysValue = getMetricFamilyParameterValues(config,... 'ModelAdvisorStandard')` method to obtain the metric family parameter values. These values are the Check Group IDs corresponding to the check groups for which you obtain compliance and issues metric data. Compliance metric data is the percentage of passed checks. Issues metric data is the number of check issues.

### Input Arguments

**config — Configuration object**

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object for which to obtain checks groups that have compliance and issues metric data.

**'ModelAdvisorStandard' — Required string**

character vector | string scalar

String that you must supply as an input.

## Output Arguments

### ValuesArray — Metric family parameter values

cell array of character vectors | cell array of string scalars

Cell array of metric family parameter values. For an `slmetric.config.Configuration` object, these values are the check groups for obtaining compliance and issues metric data.

## Examples

### Obtain Compliance and Issues Data for Groups of Model Advisor Checks

Obtain compliance and issues metric data on the Modeling Standards for MISRA C:2012, MAAB, and High-Integrity Systems check groups.

Open the default configuration.

```
config = slmetric.config.Configuration.open();
```

Specify the metric family parameter ID and the metric family parameter values. To obtain the MISRA value, open the Model Advisor Configuration Editor and select the MISRA folder. The **Check Group ID** parameter is in the folder.

```
famParamID = 'ModelAdvisorStandard';
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
setMetricFamilyParameterValues(config, famParamID, values);
```

The `maab` and `hisl_do178` checks include a subset of MAAB and High-Integrity System checks. To include all the checks, specify the Check Group ID from the Model Advisor Configuration Editor.

Check the metric family parameter values associated with the `slmetric.config.Configuration` object.

```
ValuesArray = getMetricFamilyParameterValues(config, famParamID);
```

This code is for the `ValuesArray` cell array:

```
ValuesArray =
```

3×1 cell array

```
{'_SYSTEM_By Task_misra_c'}  
{'hisl_dol78'                }  
{'maab'                      }
```

Save the new configuration.

```
config.save('FileName', 'MetricConfig.xml');
```

Set the active Metrics Dashboard configuration.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
```

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# isMetricFamilyParameterParameterized

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Determine whether Metrics Dashboard configuration object has metric family parameter values

## Syntax

```
ParameterizedConfig = isMetricFamilyParameterParameterized(  
config, ... 'ModelAdvisorStandard')
```

## Description

For an `slmetric.config.Configuration` object, use the `ParameterizedConfig = isMetricFamilyParameterParameterized(config, ... 'ModelAdvisorStandard')` method to determine whether an `slmetric.config.Configuration` object contains metric family parameter values. These values are the Check Group IDs corresponding to the check groups for which you obtain compliance and issues metric data. Compliance metric data is the percentage of passed checks. Issues metric data is the number of check issues.

## Input Arguments

**config** — Configuration object

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object for which to obtain checks groups that have compliance and issues metric data.

**'ModelAdvisorStandard'** — Required string

character vector | string scalar

Standard string that you must supply as an input.

## Output Arguments

### **ParameterizedConfig — Determine whether Metrics Dashboard configuration object has metric family parameter values**

boolean

Determine whether a Metrics Dashboard configuration object has metric family parameter values.

Data Types: Logical

## Examples

### **Obtain Compliance and Issues Data for Groups of Model Advisor Checks**

Obtain compliance and issues metric data on the Modeling Standards for MISRA C:2012, MAAB, and High-Integrity Systems check groups.

Open the default configuration.

```
config = slmetric.config.Configuration.open();
```

Specify the metric family parameter ID and the metric family parameter values. To obtain the MISRA value, open the Model Advisor Configuration Editor and select the MISRA folder. The **Check Group ID** parameter is in the folder.

```
famParamID = 'ModelAdvisorStandard';  
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};  
setMetricFamilyParameterValues(config, famParamID, values);
```

The maab and hisl\_do178 checks include a subset of MAAB and High-Integrity System checks. To include all the checks, specify the Check Group ID from the Model Advisor Configuration Editor.

Check that the `slmetric.config.Configuration` object has metric family parameter values.

```
PC = isMetricFamilyParameterParameterized(config, famParamID);
```

```
PC =
```



```
logical
```

```
1
```

Save the new configuration.

```
config.save('FileName', 'MetricConfig.xml');
```

Set the active Metrics Dashboard configuration.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
```

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## resetMetricFamilyParameterValues

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Clear metric family parameter values

### Syntax

```
resetMetricFamilyParameterValues(config, ... 'ModelAdvisorStandard')
```

### Description

For an `slmetric.config.Configuration` object, use the `resetMetricFamilyParameterValues(config, ... 'ModelAdvisorStandard')` method to clear the metric family parameter values. These values are the Check Group IDs corresponding to the check groups for which you obtain compliance and issues metric data. Compliance metric data is the percentage of passed checks. Issues metric data is the number of check issues.

### Input Arguments

**config** — Configuration object

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object for which to clear the metric family parameter values.

**'ModelAdvisorStandard'** — Required string

character vector | string scalar

Standard string that you must supply as an input.

## Examples

### Reset Metric Family Parameter Values

Obtain compliance and issues metric data on the Modeling Standards for MISRA C:2012, MAAB, and High-Integrity Systems check groups.

Open the default configuration.

```
config = slmetric.config.Configuration.open();
```

Specify the metric family parameter ID and the metric family parameter values. To obtain the MISRA value, open the Model Advisor Configuration Editor and select the MISRA folder. The **Check Group ID** parameter is in the folder.

```
famParamID = 'ModelAdvisorStandard';
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};
setMetricFamilyParameterValues(config, famParamID, values);
```

The maab and hisl\_do178 checks include a subset of MAAB and High-Integrity System checks. To include all the checks, specify the Check Group ID from the Model Advisor Configuration Editor.

Check the metric family parameter values associated with the `slmetric.config.Configuration` object.

```
ValuesArray = getMetricFamilyParameterValues(config, famParamID);
```

This code is for the ValuesArray cell array:

```
ValuesArray =
```

```
3×1 cell array

    {'_SYSTEM_By Task_misra_c'}
    {'hisl_do178'}
    {'maab'}
```

Reset the values.

```
resetMetricFamilyParameterValues(config, famParamID)
```

Check that the `slmetric.config.Configuration` object does have associated metric family parameter values.

```
ValuesArray = getMetricFamilyParameterValues(config, famParamID);
```

### See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

### Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”

“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

# setMetricFamilyParameterValues

**Class:** `slmetric.config.Configuration`

**Package:** `slmetric.config`

Obtain compliance and issues metric data on your Model Advisor configuration

## Syntax

```
setMetricFamilyParameterValues(config, ... 'ModelAdvisorStandard',  
values)
```

## Description

Use the Model Advisor Configuration Editor to create groups of Model Advisor checks or use a shipped check group. Then, use the `setMetricFamilyParameterValues(config, ... 'ModelAdvisorStandard', values)` method to obtain compliance and issues data for this group and any other groups that you specify as part of the `values` input. Compliance data is the percentage of passed checks. Issues data is the number of check issues. The `values` input sets the groups that are members of the family that you associate with a particular `slmetric.config.Configuration` object.

## Input Arguments

**config** — Configuration object

`slmetric.config.Configuration` object

`slmetric.config.Configuration` object to add check groups for which to obtain compliance and issues data.

**'ModelAdvisorStandard'** — Required string

character vector | string scalar

Standard string that you must supply as an input.

## values — Cell array of Check Group IDs

cell array of character vectors | cell array of string scalars

Specify Check Group IDs for each group of Model Advisor checks for which to obtain compliance and issues metric data. Obtain the Check Group IDs by opening up the Model Advisor Configuration Editor and selecting the folder that contains the group of checks. The **Check Group ID** parameter is in the folder.

## Examples

### Obtain Compliance and Issues Data for Groups of Model Advisor Checks

Obtain compliance and issues data on the Modeling Standards for MISRA C:2012, MAAB, and High-Integrity Systems check groups.

Open the default configuration.

```
config = slmetric.config.Configuration.open();
```

Specify the metric family parameter ID and the metric family parameter values. To obtain the MISRA value, open the Model Advisor Configuration Editor and select the MISRA folder. The **Check Group ID** parameter is in the folder.

```
famParamID = 'ModelAdvisorStandard';  
values = {'maab', 'hisl_do178', '_SYSTEM_By Task_misra_c'};  
setMetricFamilyParameterValues(config, famParamID, values);
```

The `maab` and `hisl_do178` checks include a subset of MAAB and High-Integrity System checks. To include all the checks, specify the Check Group ID from the Model Advisor Configuration Editor.

Save the new configuration.

```
config.save('FileName', 'MetricConfig.xml');
```

Set the active Metrics Dashboard configuration.

```
slmetric.config.setActiveConfiguration(fullfile(pwd, 'MetricConfig.xml'));
```

## See Also

`slmetric.config.Configuration` | `slmetric.config.getActiveConfiguration`  
| `slmetric.config.setActiveConfiguration`

## Topics

“Collect and Explore Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getMargin

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Obtain distance from container edge to its contents

## Syntax

```
pixels = getMargin(Container)
```

## Description

`pixels = getMargin(Container)` returns how far in pixels the edges of an `slmetric.dashboard.Container` object is from the widgets that it contains.

## Input Arguments

**Container — Metrics Dashboard container**

`slmetric.dashboard.Container` object

The `slmetric.dashboard.Container` object for which you are obtaining the margin distance.

## Output Arguments

**pixels — Container margins**

character vector | string scalar

Margin distance from container contents in pixels.

Example: `'40 px'`

Data Types: `char`



## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getPosition

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Obtain container position within Metrics Dashboard

### Syntax

```
Num = getPosition(Container)
```

### Description

`Num = getPosition(Container)` returns the position of an `slmetric.dashboard.Container` object in an array that holds Metrics Dashboard objects. These objects are in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object. The order of containers in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

### Input Arguments

**Container — Metrics Dashboard container**

`slmetric.dashboard.Container` object

Specify the `slmetric.dashboard.Container` object for which you get its position in the array.

### Output Arguments

**Num — Position of container object**

`double`

Position of `slmetric.dashboard.Container` object within an array that holds the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setPosition

**Class:** `slmetric.dashboard.Container`

**Package:** `slmetric.dashboard`

Set container position within Metrics Dashboard

## Syntax

```
setPosition(Container, num)
```

## Description

`setPosition(Container, num)` sets the position of an `slmetric.dashboard.Container` object in an array that holds Metrics Dashboard objects. This array contains the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object. The order of containers in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

## Input Arguments

**Container — Metrics Dashboard container**

`slmetric.dashboard.Container` object

Specify the `slmetric.dashboard.Container` object for which you set its position in the array.

## Output Arguments

**Num — Position of container object**

double

Position of `slmetric.dashboard.Container` object within an array that holds the Metrics Dashboard objects in either an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getHeight

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Obtain height of Metrics Dashboard custom widget

### Syntax

```
Height = getHeight(CustomWidget)
```

### Description

`Height = getHeight(CustomWidget)` returns the height of a custom widget in pixels.

### Input Arguments

**CustomWidget — Metrics Dashboard custom widget**

`slmetric.dashboard.CustomWidget` object

`slmetric.dashboard.CustomWidget` for which you want to obtain its height.

### Output Arguments

**num — Height in pixels**

integer

Height of `slmetric.dashboard.CustomWidget` object in pixels.

Example: `Height = getHeight(CustomWidget)`

Data Types: `uint32`

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
            this.ValueName = 'Nonvirtual Blocks'
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;
        end

        function res = algorithm(this, component)
```

```
% create a result object for this component
res = slmetric.metric.Result();

% set the component and metric ID
res.ComponentID = component.ID;
res.MetricID = this.ID;

% Practice
D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
D1.Value=0;
D1.setGroup('Group1','Group1Name');
D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
D2.Value=1;
D2.setGroup('Group1','Group1Name');

% use find_system to get all blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
        end
    end
end
```



```

end
case 'Selector'
% Virtual only when Number of input dimensions
% specifies 1 and Index Option specifies Select
% all, Index vector (dialog), or Starting index (dialog).
nod = get_param(blocks{n}, 'NumberOfDimensions');
ios = get_param(blocks{n}, 'IndexOptionArray');

ios_settings = {'Assign all', 'Index vector (dialog)', ...
  'Starting index (dialog)'};

if nod == 1 && any(strcmp(ios_settings, ios))
  isNonVirtual(n) = false;
end
case 'Trigger'
% Virtual when the output port is not present.
if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
  isNonVirtual(n) = false;
end
case 'Enable'
% Virtual unless connected directly to an Outport block.
isNonVirtual(n) = false;

if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
  pc = get_param(blocks{n}, 'PortConnectivity');

  if ~isempty(pc.DstBlock) && ...
    strcmp(get_param(pc.DstBlock, 'BlockType'), ...
      'Outport')
    isNonVirtual(n) = true;
  end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

[slmetric.dashboard.getActiveConfiguration](#) |  
[slmetric.dashboard.setActiveConfiguration](#)

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getMetricIDs

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Obtain metric identifier for custom Metrics Dashboard widget

### Syntax

```
metricID = getMetricIDs(CustomWidget)
```

### Description

`metricID = getMetricIDs(CustomWidget)` returns the metric identifier for an `slmetric.dashboard.CustomWidget` object.

### Input Arguments

**CustomWidget — Custom widget object**

`slmetric.dashboard.CustomWidget` object

`slmetric.dashboard.CustomWidget` object for which you want to obtain the associated metric identifier. The `slmetric.dashboard.CustomWidget` object is the means of visualizing metric data for the metric identifier.

Data Types: `char`

### Output Arguments

**metricID — Metric identifier**

`character vector`  
`string scalar`

Metric identifier associated with an `slmetric.dashboard.CustomWidget` object.

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'
            this.ValueName = 'Nonvirtual Blocks'
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;
        end

        function res = algorithm(this, component)
```

```
% create a result object for this component
res = slmetric.metric.Result();

% set the component and metric ID
res.ComponentID = component.ID;
res.MetricID = this.ID;

% Practice
D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
D1.Value=0;
D1.setGroup('Group1','Group1Name');
D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
D2.Value=1;
D2.setGroup('Group1','Group1Name');

% use find_system to get blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
            default
                % Default case: isNonVirtual(n) = true;
        end
    end
end
```

```

        end
    case 'Selector'
        % Virtual only when Number of input dimensions
        % specifies 1 and Index Option specifies Select
        % all, Index vector (dialog), or Starting index (dialog).
        nod = get_param(blocks{n}, 'NumberOfDimensions');
        ios = get_param(blocks{n}, 'IndexOptionArray');

        ios_settings = {'Assign all', 'Index vector (dialog)', ...
            'Starting index (dialog)'};

        if nod == 1 && any(strcmp(ios_settings, ios))
            isNonVirtual(n) = false;
        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.



```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

[slmetric.dashboard.getActiveConfiguration](#) |  
[slmetric.dashboard.setActiveConfiguration](#)

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)

[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getPosition

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Obtain custom widget position within Metrics Dashboard

### Syntax

```
Num = getPosition(CustomWidget)
```

### Description

`Num = getPosition(CustomWidget)` returns the position of an `slmetric.dashboard.CustomWidget` object within an array that holds Metrics Dashboard objects. These objects are in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Group`, or an `slmetric.dashboard.Container` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

### Input Arguments

**CustomWidget — Metrics Dashboard custom widget**

`slmetric.dashboard.CustomWidget` object

Specify the `slmetric.dashboard.CustomWidget` object for which you get its position in the array.

### Output Arguments

**Num — Position of custom widget object**

`double`

Position of `slmetric.dashboard.CustomWidget` object within an array that holds the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Group`, or an `slmetric.dashboard.Container` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setPosition

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Set custom widget position within Metrics Dashboard

## Syntax

```
setPosition(CustomWidget, num)
```

## Description

`setPosition(CustomWidget, num)` sets the position of an `slmetric.dashboard.CustomWidget` object in an array that holds Metrics Dashboard objects. This array contains the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

## Input Arguments

**CustomWidget — Metrics Dashboard custom widget**

`slmetric.dashboard.CustomWidget` object

Specify the `slmetric.dashboard.CustomWidget` object for which you set its position in the array.

## Output Arguments

**Num — Position of widget object**

double

Position of `slmetric.dashboard.CustomWidget` object within an array that holds the Metrics Dashboard objects in either an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setHeight

**Class:** `slmetric.dashboard.CustomWidget`

**Package:** `slmetric.dashboard`

Specify height of Metrics Dashboard custom widget

## Syntax

```
setHeight(CustomWidget,num)
```

## Description

`setHeight(CustomWidget,num)` specifies the height of a custom widget in pixels.

## Input Arguments

### **CustomWidget — Metrics Dashboard customwidget**

`slmetric.dashboard.CustomWidget`

`slmetric.dashboard.CustomWidget` for which you want to specify its height.

### **num — Height in pixels**

integer

Height of `slmetric.dashboard.CustomWidget` object in pixels. These are the minimum heights that you can set.

- For the `SingleValue` custom widget, the minimum height is 25 pixels.
- For the `BarChart` custom widget, the minimum height is 150 pixels.
- For the `RadialGauge` custom widget, the minimum height is 120 pixels.
- For the `DistributionHeatMap` custom widget, the minimum height is 90 pixels.

Example: `setHeight(widget, 50)`

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file.

```
classdef nonvirtualblockcount < slmetric.metric.Metric
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.
    properties
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...
            'GotoTagVisiblity','Mux','SignalSpecification', ...
            'Terminator','Inport'};
    end

    methods
        function this = nonvirtualblockcount()
            this.ID = 'nonvirtualblockcount';
            this.Name = 'Nonvirtual Block Count';
            this.Version = 1;
            this.CompileContext = 'None';
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)';
            this.ValueName = 'Nonvirtual Blocks';
            this.ComponentScope = [Advisor.component.Types.Model, ...
                Advisor.component.Types.SubSystem];
            this.AggregationMode = slmetric.AggregationMode.Sum;
            this.AggregateComponentDetails = true;
            this.ResultChecksumCoverage = true;
            this.SupportsResultDetails = true;
        end

        function res = algorithm(this, component)
            % create a result object for this component
```

```
res = slmetric.metric.Result();

% set the component and metric ID
res.ComponentID = component.ID;
res.MetricID = this.ID;

% Practice
D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
D1.Value=0;
D1.setGroup('Group1','Group1Name');
D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
D2.Value=1;
D2.setGroup('Group1','Group1Name');

% use find_system to get all blocks inside this component
blocks = find_system(getPath(component), ...
    'SearchDepth', 1, ...
    'Type', 'Block');

isNonVirtual = true(size(blocks));

for n=1:length(blocks)
    blockType = get_param(blocks{n}, 'BlockType');

    if any(strcmp(this.VirtualBlockTypes, blockType))
        isNonVirtual(n) = false;
    else
        switch blockType
            case 'SubSystem'
                % Virtual unless the block is conditionally executed
                % or the Treat as atomic unit check box is selected.
                if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                    'on')
                    isNonVirtual(n) = false;
                end
            case 'Outport'
                % Outport: Virtual when the block resides within
                % SubSystem block (conditional or not), and
                % does not reside in the root (top-level) Simulink window.
                if component.Type ~= Advisor.component.Types.Model
                    isNonVirtual(n) = false;
                end
        end
    end
end
```



```

case 'Selector'
    % Virtual only when Number of input dimensions
    % specifies 1 and Index Option specifies Select
    % all, Index vector (dialog), or Starting index (dialog).
    nod = get_param(blocks{n}, 'NumberOfDimensions');
    ios = get_param(blocks{n}, 'IndexOptionArray');

    ios_settings = {'Assign all', 'Index vector (dialog)', ...
        'Starting index (dialog)'};

    if nod == 1 && any(strcmp(ios_settings, ios))
        isNonVirtual(n) = false;
    end
case 'Trigger'
    % Virtual when the output port is not present.
    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
        isNonVirtual(n) = false;
    end
case 'Enable'
    % Virtual unless connected directly to an Output block.
    isNonVirtual(n) = false;

    if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
        pc = get_param(blocks{n}, 'PortConnectivity');

        if ~isempty(pc.DstBlock) && ...
            strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                'Output')
            isNonVirtual(n) = true;
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end
end

```

Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf, 'Filename', 'DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For your model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the **All Metrics** button and run all metrics.

## See Also

```
slmetric.dashboard.getActiveConfiguration |  
slmetric.dashboard.setActiveConfiguration
```

## Topics

["Collect Model Metric Data by Using the Metrics Dashboard"](#)

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

**Introduced in R2018b**

## getMargin

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Obtain distance from group edge to contents

## Syntax

```
pixels = getMargin(Group)
```

## Description

`pixels = getMargin(Group)` returns how far in pixels the edges of an `slmetric.dashboard.Group` object is from the widgets that it contains.

## Input Arguments

**Group — Metrics Dashboard group**

`slmetric.dashboard.Group` object

The `slmetric.dashboard.Group` object for which you are obtaining the margin distance.

## Output Arguments

**pixels — Group margins**

character vector | string scalar

Margin distance from group contents in pixels.

Example: `'40 px'`

Data Types: `char`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getPosition

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Obtain group position within Metrics Dashboard

### Syntax

```
Num = getPosition(Group)
```

### Description

`Num = getPosition(Group)` returns the position of an `slmetric.dashboard.Group` object within an array that holds Metrics Dashboard objects. These objects are in an `slmetric.dashboard.Layout` or an `slmetric.dashboard.Container` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

### Input Arguments

**Group — Metrics Dashboard group**

`slmetric.dashboard.Group` object

Specify the `slmetric.dashboard.Group` object for which you get its position in the array.

### Output Arguments

**Num — Position of group object**

double

Position of `slmetric.dashboard.Group` object within an array that holds the Metrics Dashboard objects in an `slmetric.dashboard.Layout` or an `slmetric.dashboard.Container` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setMargin

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Specify distance from group edge to its contents

## Syntax

```
pixels = setMargin(Group,px)
```

## Description

`pixels = setMargin(Group,px)` specifies how far in pixels the edges of an `slmetric.dashboard.Group` object is from the widgets that it contains.

## Input Arguments

### **Group — Metrics Dashboard group**

`slmetric.dashboard.Group` object

The `slmetric.dashboard.Group` object for which you are specifying margin size in pixels.

Data Types: `char`

### **px — Group margins**

`character vector|string scalar`

Margin distance from group contents in pixels.

Example: `'40 px'`

Data Types: `char`



## See Also

`slmetric.dashboard.getActiveConfiguration |`  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## setPosition

**Class:** `slmetric.dashboard.Group`

**Package:** `slmetric.dashboard`

Set group position within Metrics Dashboard

## Syntax

```
setPosition(Group, num)
```

## Description

`setPosition(Group, num)` sets the position of an `slmetric.dashboard.Group` object in an array that holds Metrics Dashboard objects. This array contains the Metrics Dashboard objects in an `slmetric.dashboard.Layout` or an `slmetric.dashboard.Container` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

## Input Arguments

**Group — Metrics Dashboard group**

`slmetric.dashboard.Group` object

Specify the `slmetric.dashboard.Group` object for which you set its position in the array.

## Output Arguments

**Num — Position of group object**

double

Position of `slmetric.dashboard.Group` object within an array that holds the Metrics Dashboard objects in either an `slmetric.dashboard.Layout` or an `slmetric.dashboard.Container` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## getHeight

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Obtain height of Metrics Dashboard widget

### Syntax

```
Height = getHeight(widget)
```

### Description

`Height = getHeight(widget)` returns the height of a widget in pixels.

### Input Arguments

**widget — Metrics Dashboard widget**

`slmetric.dashboard.Widget` object

`slmetric.dashboard.Widget` for which you want to specify its height.

### Output Arguments

**Height — Height in pixels**

integer

Height of `slmetric.dashboard.Widget` object in pixels.

Example: `Height = getHeight(widget)`

Data Types: `uint32`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

[“Collect Model Metric Data by Using the Metrics Dashboard”](#)  
[“Customize Metrics Dashboard Layout and Functionality”](#)

**Introduced in R2018b**

## getPosition

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Obtain widget position within Metrics Dashboard

### Syntax

```
Num = getPosition(Widget)
```

### Description

`Num = getPosition(Widget)` returns the position of an `slmetric.dashboard.Widget` object within an array that holds Metrics Dashboard objects. These objects are in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Group`, or an `slmetric.dashboard.Container` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

### Input Arguments

**Widget — Metrics Dashboard widget**

`slmetric.dashboard.Widget` object

Specify the `slmetric.dashboard.Widget` object for which you get its position in the array.

### Output Arguments

**Num — Position of widget object**

`double`

Position of `slmetric.dashboard.Widget` object within an array that holds the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Group`, or an `slmetric.dashboard.Container` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## setHeight

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Specify height of Metrics Dashboard widget

### Syntax

```
setHeight(widget,num)
```

### Description

`setHeight(widget,num)` specifies the height of a widget in pixels.

### Input Arguments

**widget — Metrics Dashboard widget**

`slmetric.dashboard.Widget`

`slmetric.dashboard.Widget` for which you want to specify its height.

**num — Height in pixels**

integer

Height of `slmetric.dashboard.Widget` object in pixels. These are the minimum heights that you can set.

- For the `SystemInfo` widget, the minimum height is 90 pixels.
- For the `LibraryReuse` widget, the minimum height is 110 pixels.
- For the `GlocalInterface` widget, the minimum height is 60 pixels.

Example: `setHeight(widget, 70)`



## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

*“Collect Model Metric Data by Using the Metrics Dashboard”*

*“Customize Metrics Dashboard Layout and Functionality”*

**Introduced in R2018b**

## setPosition

**Class:** `slmetric.dashboard.Widget`

**Package:** `slmetric.dashboard`

Set widget position within Metrics Dashboard

## Syntax

```
setPosition(Widget, num)
```

## Description

`setPosition(Widget, num)` sets the position of an `slmetric.dashboard.Widget` object in an array that holds Metrics Dashboard objects. This array contains the Metrics Dashboard objects in an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object. The order of objects in the array corresponds to proceeding from left to right, and then down in the Metrics Dashboard.

## Input Arguments

**Widget — Metrics Dashboard widget**

`slmetric.dashboard.Widget` object

Specify the `slmetric.dashboard.Widget` object for which you set its position in the array.

## Output Arguments

**Num — Position of widget object**

`double`

Position of `slmetric.dashboard.Widget` object within an array that holds the Metrics Dashboard objects in either an `slmetric.dashboard.Layout`, an `slmetric.dashboard.Container`, or an `slmetric.dashboard.Group` object.

Data Types: `double`

## See Also

`slmetric.dashboard.getActiveConfiguration` |  
`slmetric.dashboard.setActiveConfiguration`

## Topics

“Collect Model Metric Data by Using the Metrics Dashboard”  
“Customize Metrics Dashboard Layout and Functionality”

**Introduced in R2018b**

## **slmetric.metric.MetaInformation class**

**Package:** `slmetric.metric`

Set metadata for custom metrics

### **Description**

The `slmetric.metric.MetaInformation` class properties contain metric metadata. On the Metrics Dashboard, when you click the widget for an individual metric, this metadata is in the table. For custom metrics, when you create a custom metric class, you specify the `slmetric.metric.MetaInformation` the applicable properties.

### **Construction**

Create an `slmetric.Engine` object. Use the `getMetricMetaInformation` property to return an `slmetric.metric.MetaInformation` object.

### **Properties**

#### **Name — Metric name**

character value | string scalar

For custom metrics, when you define the custom metric class, specify this property. For shipped metrics, this property is already set.

Example: 'Model Advisor standards check compliance for High Integrity'

Data Types: `char`

#### **Description — Metric description**

character value | string scalar

For custom metrics, when you define the custom metric class, specify this property. For shipped metrics, this property is already set.

Example: 'Metric that counts the percentage of checks that passed for the High Integrity Model Advisor standards check grouping.'

Data Types: char

### **MeasuresNames — Names of metric measures**

cell array of character vectors | cell array of string scalars

For custom metrics, when you define the custom metric class, if applicable, specify this property. For shipped metrics, this property is already set.

Example: {'Passed Checks'} {'Total Checks'}

Data Types: char

### **AggregatedMeasuresNames — Names of aggregated metric measures**

cell array of character vectors | cell array of string scalars

For custom metrics, when you define the custom metric class, if applicable, specify this property. For shipped metrics, this property is already set.

Example: {'Passed Checks (incl. Descendants)'} {'Total Checks'}

Data Types: char

### **ValueName — Value name**

character vector | string scalar

For custom metrics, when you define the custom metric class, specify this property. For shipped metrics, this property is already set.

Example: 'Passed Checks'

Data Types: char

### **AggregatedValueName — Name of aggregated metric value**

cell array of character vectors | cell array of string scalars

For custom metrics, when you define the custom metric class, specify this property. For shipped metrics, this property is already set.

Example: {'Passed Checks (incl. Descendants)'} {'Total Checks'}

Data Types: char

## Examples

### Add a Custom Widget to a Group

Create a custom metric that counts nonvirtual blocks. Specify a widget to display this metric on the Metrics Dashboard. Add it to the Size Group.

Create a custom metric class.

```
className = 'nonvirtualblockcount';  
slmetric.metric.createNewMetricClass(className);
```

Create the nonvirtual block count metric by adding this code to the `nonvirtualblockcount.m` file. The `this = nonvirtualblockcount` function sets the `slmetric.metric.MetaInformation` properties.

```
classdef nonvirtualblockcount < slmetric.metric.Metric  
    %nonvirtualblockcount calculates number of nonvirtual blocks per level.  
    % BusCreator, BusSelector and BusAssign are treated as nonvirtual.  
    properties  
        VirtualBlockTypes = {'Demux','From','Goto','Ground', ...  
            'GotoTagVisiblity','Mux','SignalSpecification', ...  
            'Terminator','Inport'};  
    end  
  
    methods  
        function this = nonvirtualblockcount()  
            this.ID = 'nonvirtualblockcount';  
            this.Name = 'Nonvirtual Block Count';  
            this.Version = 1;  
            this.CompileContext = 'None';  
            this.Description = 'Algorithm that counts nonvirtual blocks per level.';  
            this.AggregatedValueName = 'Nonvirtual Blocks (incl. Descendants)'  
            this.ValueName = 'Nonvirtual Blocks'  
            this.ComponentScope = [Advisor.component.Types.Model, ...  
                Advisor.component.Types.SubSystem];  
            this.AggregationMode = slmetric.AggregationMode.Sum;  
            this.AggregateComponentDetails = true;  
            this.ResultChecksumCoverage = true;  
            this.SupportsResultDetails = true;  
        end  
    end  
end
```

```

function res = algorithm(this, component)
    % create a result object for this component
    res = slmetric.metric.Result();

    % set the component and metric ID
    res.ComponentID = component.ID;
    res.MetricID = this.ID;

    % Practice
    D1=slmetric.metric.ResultDetail('identifier 1','Name 1');
    D1.Value=0;
    D1.setGroup('Group1','Group1Name');
    D2=slmetric.metric.ResultDetail('identifier 2','Name 2');
    D2.Value=1;
    D2.setGroup('Group1','Group1Name');

    % use find_system to get blocks inside this component
    blocks = find_system(getPath(component), ...
        'SearchDepth', 1, ...
        'Type', 'Block');

    isNonVirtual = true(size(blocks));

    for n=1:length(blocks)
        blockType = get_param(blocks{n}, 'BlockType');

        if any(strcmp(this.VirtualBlockTypes, blockType))
            isNonVirtual(n) = false;
        else
            switch blockType
                case 'SubSystem'
                    % Virtual unless the block is conditionally executed
                    % or the Treat as atomic unit check box is selected.
                    if strcmp(get_param(blocks{n}, 'IsSubSystemVirtual'), ...
                        'on')
                        isNonVirtual(n) = false;
                    end
                case 'Outport'
                    % Outport: Virtual when the block resides within
                    % SubSystem block (conditional or not), and
                    % does not reside in the root (top-level) Simulink window.
                    if component.Type ~= Advisor.component.Types.Model

```

```

        isNonVirtual(n) = false;
    end
    case 'Selector'
        % Virtual only when Number of input dimensions
        % specifies 1 and Index Option specifies Select
        % all, Index vector (dialog), or Starting index (dialog).
        nod = get_param(blocks{n}, 'NumberOfDimensions');
        ios = get_param(blocks{n}, 'IndexOptionArray');

        ios_settings = {'Assign all', 'Index vector (dialog)', ...
            'Starting index (dialog)'};

        if nod == 1 && any(strcmp(ios_settings, ios))
            isNonVirtual(n) = false;
        end
    case 'Trigger'
        % Virtual when the output port is not present.
        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'off')
            isNonVirtual(n) = false;
        end
    case 'Enable'
        % Virtual unless connected directly to an Outport block.
        isNonVirtual(n) = false;

        if strcmp(get_param(blocks{n}, 'ShowOutputPort'), 'on')
            pc = get_param(blocks{n}, 'PortConnectivity');

            if ~isempty(pc.DstBlock) && ...
                strcmp(get_param(pc.DstBlock, 'BlockType'), ...
                    'Outport')
                isNonVirtual(n) = true;
            end
        end
    end
end
end
end
end

blocks = blocks(isNonVirtual);

res.Value = length(blocks);
end
end
end
end

```



Register the new metric in the metric repository.

```
[id_metric,err_msg] = slmetric.metric.registerMetric(className);
```

To begin, open the default configuration for the Metrics Dashboard layout.

```
conf = slmetric.dashboard.Configuration.open();
```

Obtain the `slmetric.dashboard.Layout` object from the `slmetric.dashboard.Configuration` object.

```
layout = getDashboardLayout(conf);
```

Obtain widget objects that are in the layout object.

```
layoutWidget=getWidgets(layout);
```

Remove the widget that represents the Simulink block count metric.

```
sizeGroup = layoutWidget(2);  
sizeGroupWidgets = sizeGroup.getWidgets();  
sizeGroup.removeWidget(sizeGroupWidgets(1));
```

Add a widget that displays the nonvirtual block count metric. For custom widgets, the default visualization type is single value. If you want to use a different visualization technique, specify a different value for the `VisualizationType` property.

```
newWidget = sizeGroup.addWidget('Custom', 1);  
newWidget.Title=('Nonvirtual Block Count');  
newWidget.setMetricIDs('nonvirtualblockcount');  
newWidget.setWidths(slmetric.dashboard.Width.Medium);  
newWidget.setHeight(70);
```

Specify whether there are lines separating the custom widget from other widgets in the group. These commands specify that there is a line to the right of the widget.

```
s.top = false;  
s.bottom = false;  
s.left= false;  
s.right= true;  
newWidget.setSeparators([s, s, s, s]);
```

Save the configuration object. This command serializes the API information to an XML file.

```
save(conf,'Filename','DashboardConfig.xml');
```

Set the active configuration.

```
slmetric.dashboard.setActiveConfiguration(fullfile(pwd, 'DashboardConfig.xml'));
```

For a model, open the Metrics Dashboard.

```
metricsdashboard sf_car
```

Click the play button and run all metrics.

## See Also

**Introduced in R2018b**

# getMetricMetaInformation

**Class:** slmetric.Engine

**Package:** slmetric

Obtain metric metadata

## Syntax

```
metaInfo = getMetricMetaInformation(metric_engine,metricID)
```

## Description

`metaInfo = getMetricMetaInformation(metric_engine,metricID)` returns the `slmetric.metric.MetaInformation` object corresponding to the `metricID`.

## Input Arguments

**metric\_engine — Metric engine object**

`slmetric.Engine` object

Create an `slmetric.Engine` object.

```
metric_engine = slmetric.Engine();
```

Data Types: char

**MetricID — Metric identifier**

character vector | string scalar

Metric identifier for shipped or custom metrics. You can get metric identifiers by calling the `slmetric.metric.getAvailableMetrics`.

Data Types: char

## Output Arguments

### **metaInfo** — Meta information object

`slmetric.metric.MetaInformation` object

For a `metricID`, the `slmetric.metric.MetaInformation` object contains its metadata. On the Metrics Dashboard, when you click a widget, this metadata appears on the table.

## Examples

### Obtain Metric Metadata

Obtain metadata for the high-integrity check compliance metric. This metric has a **metric ID** of `mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178`.

Create an `slmetric.Engine` object.

```
metric_engine = slmetric.Engine();
```

To obtain metadata, use the `getMetricMetaInformation` method.

```
getMetricMetaInformation('metricEngine',...  
'mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178')
```

The high-integrity check compliance metric contains this metadata:

```
metaInfo =
```

```
MetaInformation with properties:
```

```
    Name: 'Model Advisor standards check compliance for High Integrity'  
  Description: 'Metric that counts the percentage of checks that passed for the High Integrity'  
 MeasuresNames: {2x1 cell}  
AggregatedMeasuresNames: {2x1 cell}
```

ValueName: 'Checks Passed'  
AggregatedValueName: 'Checks Passed (incl. Descendants)'

## See Also

**Introduced in R2018b**

## ModelAdvisor.ResultDetail class

**Package:** ModelAdvisor

Defines result detail objects

### Description

In the check callback function, `ModelAdvisor.ResultDetail` objects are created for each model element returned by the `find_system()` API, such as a collection of blocks that violate a check.

`ResultDetailObjs` objects are saved as a `ResultDetails` property of the `ModelAdvisor.Check` class.

### Properties

#### **IsInformer — Non-violated results**

false (default)

Provides informational content.

Data Types: logical

#### **IsViolation — Violated results**

true (default)

Provides informational content and a recommended action message for fixing the issues.

Data Types: logical

#### **Description — Description**

character vector

Specify a message that describes the result. This text is presented in the Model Advisor result pane.

Data Types: char

**Title — Title**

comma-separated pairs of Name, Value arguments

Specify a title for the result. This text is presented in the Model Advisor result pane.

Data Types: char

**Information — Additional information**

character vector

Specify a message that provides additional information about the result. This text is presented in the Model Advisor result pane.

Data Types: char

**Status — Execution result**

character vector

Specify a status message for the result. The text is presented in the Model Advisor result pane.

Data Types: char

**RecAction — Recommended action**

character vector

Specify a recommended action message for the result. The text is presented in the Model Advisor result pane.

Data Types: char

## Examples

This example shows result details that correspond to the execution of check "Check whether block names appear below blocks" in the `slvndemo_mdldv` example model. To review the definition of the check, open the `sl_customization.m` file from the example model and see the sample code for `ModelAdvisor.Check('com.mathworks.sample.Check0')`.

## Define a Collection of Result Detail Objects

From the `slvndemo_mdldv` example model, open the `sl_customization.m` file. In the check callback function, the `find_system()` API returns model elements in the

system that meet a specified criteria. In this example, the function returns blocks whose name does not appear below the block (violationBlks).

```
% find all blocks whose name does not appear below blocks
violationBlks = find_system('System','Type','block','NamePlacement','alternate','ShowName','on');
```

`ModelAdvisor.ResultDetail` creates `ResultDetailObjs` for each model element returned by the `find_system` API. When `violationBlks` is empty, the `ElementResults` collection consists of a single object. The `Name, Value` pairs define the collection for a nonviolated check. For this type of collection, the `Simulink.ModelAdvisor.setCheckResultStatus(true)` method specifies that the check is not violated and displays `Passed` on the Model Advisor.

In this code sample, the `find_system` API does not identify blocks whose name appears below the block, therefore `ElementResults` provides information content only.

```
% Results when no blocks
% violate the check
if isempty(violationBlks)
    ElementResults = ModelAdvisor.ResultDetail;
    ElementResults.IsInformer = true;
    ElementResults.Title = 'Identify blocks where the name is not displayed below the block.';
    ElementResults.Information = 'Verifies that the name appears below the block.';
    ElementResults.Description = 'Identify blocks where the name is not displayed below the block.';
    ElementResults.Status = 'All blocks have names displayed below the block.';
    mdladvObj.setCheckResultStatus(true);
end
```

This example shows the `ModelAdvisor.ResultDetail` properties for `ElementResults` when check "Check whether block names appear below blocks" is not violated.

```
ElementResults =
ResultDetail with properties:
    IsInformer = 1
    Description = 'Identify blocks where the name is not displayed below the block.'
    Title = 'Check whether block names appear below blocks'
    Information = 'Verifies that the name appears below the block.'
    Status = 'All blocks have names displayed below the block.'
```

When the `find_system` API returns a list of model elements that meet specified criteria, the `ModelAdvisor.ResultDetail` class creates a `ResultDetailObjs` object for each element in `violationBlks`. The `Name, Value` pairs define `ElementResults` as a collection of objects that violate the check. For this collection, the `Simulink.ModelAdvisor.setCheckResultStatus(false)` method specifies that the check is violated and displays `Warning` or `Failed` on the Model Advisor. The `Simulink.ModelAdvisor.setActionEnable(true)` method enables the ability to fix the check violation issue from the Model Advisor.



In this code sample, the `find_system` API returns a list of blocks whose name appears below the block. `ElementResults` includes each `ResultDetailObj` object that violates the check and provides a recommended action message for fixing the check violation.

```
% Create results when blocks violate the check
else
    ElementResults(1,numel(violationBlks))=ModelAdvisor.ResultDetail;
    for i=1:numel(ElementResults)
        ElementResults(i).setData(violationBlks{i});
        ElementResults.Title = 'Identify blocks where the name is not displayed below the block.';
        ElementResults.Information = 'Verifies that the name appears below the block.';
        ElementResults.Description = 'Identify blocks where the name is not displayed below the block.';
        ElementResults.Status = 'The following blocks have names that do not display below the blocks: ';
        ElementResults.RecAction = 'Change the location such that the block name is below the block.';
    end
    mdladvObj.setCheckResultStatus(false);
    mdladvObj.setActionEnable(true);
```

This example shows the `ModelAdvisor.ResultDetail` properties for `ElementResults` when check "Check whether block names appear below blocks" is violated.

```
ElementResults =
ResultDetail with properties:
    IsInformer = 0;
    Description = 'Identify blocks where the name is not displayed below the block.';
    Title = 'Check whether block names appear below blocks';
    Information = 'Verifies that the name appears below the block.';
    Status = 'The following blocks have names that do not display below the blocks: ';
    RecAction = 'Change the location such that the block name is below the block.'
```

The `ModelAdvisor.Check.setResultDetails` method associates the results with the check (`CheckObj`).

```
% Associate the results with the check
CheckObj.setResultDetails([CheckObj.ResultDetails, ElementResults]);
```

After executing the check, you can view the results in the Model Advisor as a collection, such as by recommended action, block, or subsystem. To define this report style, specify 'DetailStyle' as the callback style in the `ModelAdvisor.Check.setCallbackFcn` method.

```
% Define Model Advisor check "Check whether block names appear
% below blocks".
rec = ModelAdvisor.Check('com.mathworks.sample.Check0');
rec.Title = 'Check whether block names appear below blocks (recommended check style)';
rec.TitleTips = 'Example new style callback (recommended check style)';
```

```
rec.setCallbackFcn(@SampleNewCheckStyleCallback, 'None', 'DetailStyle');
```

## See Also

[ModelAdvisor.Check.ResultDetails](#) | [setResultDetails](#)

## Topics

[“Create Model Advisor Checks”](#)

[“Create Customized Pass/Fail Check with Detailed Result Collections”](#)

[“Check Callback Function for Detailed Result Collections”](#)

**Introduced in R2018b**

# setResultDetails

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Associates result details with a check object

## Syntax

```
setResultDetails(ElementResults)
```

## Description

In the check callback function, use `setResultDetails(ElementResults)` to associate `ElementResults` with the check (`CheckObj`).

`ElementResults` is a collection of instances of the `ModelAdvisor.ResultDetail` class.

## Input Arguments

`ElementResults`    Collection of `ResultDetailObj`s objects

## Examples

This example shows the result details that correspond to the execution of check "Check whether block names appear below blocks" in the `slvndemo_mdadv` example model. At the end of the code, `CheckObj.setResultDetails(ElementResults)`; associates the results with the check object.

```
% -----
% Sample new check style callback function, used for check
% Check whether block names appear below blocks
% -----
function SampleNewCheckStyleCallback(system, CheckObj)
```

```
mdladvObj = Simulink.ModelAdvisor.getModelAdvisor(system); % get object

% find all blocks whose name does not appear below blocks
violationBlks = find_system(system, 'Type','block',...
    'NamePlacement','alternate',...
    'ShowName', 'on');

if isempty(violationBlks)
    ElementResults = ModelAdvisor.ResultDetail;
    ElementResults.IsInformer = true;
    ElementResults.Description = 'Identify blocks where the name
        is not displayed below the block.';
    ElementResults.Status = 'All blocks have names displayed below
        the block.';
    mdladvObj.setCheckResultStatus(true);
else
    ElementResults(1,numel(violationBlks))=ModelAdvisor.ResultDetail;
    for i=1:numel(ElementResults)
        ElementResults(i).setData(violationBlks{i});
        ElementResults(i).Description = 'Identify blocks where the name
            is not displayed below the block.';
        ElementResults(i).Status = 'The following blocks have names that
            do not display below the blocks:';
        ElementResults(i).RecAction = 'Change the location such that the
            block name is below the block.';
    end
    mdladvObj.setCheckResultStatus(false);
    mdladvObj.setActionEnable(true);
end

CheckObj.setResultDetails(ElementResults);
```

## See Also

[ModelAdvisor.Check.ResultDetails](#) | [ModelAdvisor.ResultDetail](#)

## Topics

“Create Model Advisor Checks”

“Create Customized Pass/Fail Check with Detailed Result Collections”

“Check Callback Function for Detailed Result Collections”

## Introduced in R2018b

## ResultDetails property

**Class:** ModelAdvisor.Check

**Package:** ModelAdvisor

Result details in a cell array

### Values

Cell array

**Default:** {} (empty cell array)

### Description

The `ResultDetails` property stores the `ResultDetailObj`s objects associated with the check. This property can contain multiple objects.

**Introduced in R2018b**

## slslicer

Create API object for invoking Model Slicer

### Syntax

```
slslicer(model)
slslicer(model,opts)
obj = slslicer(model)
```

### Description

`slslicer(model)` creates a Model Slicer configuration object for the model `model` by exposing the methods for invoking Model Slicer.

`slslicer(model,opts)` creates a Model Slicer configuration object for the model `model` by using the options object `opts`, as defined by `slsliceroptions`.

`obj = slslicer(model)` creates a Model Slicer configuration object. You can apply the methods on the Model Slicer object `obj`.

### Examples

#### Add Starting Point and Highlight the Model Slice

Add a new starting point to the active Model Slicer configuration, and then highlight the model.

Open the `sldvSliceClimateControlExample` example model.

```
addpath(fullfile(docroot,'toolbox','simulink','examples'));
open_system('sldvSliceClimateControlExample');
```

Create a Model Slicer configuration object for the model by using `slslicer`.

```
obj = slslicer('sldvSliceClimateControlExample');
```

Activate the slice highlighting mode of Model Slicer to compile the model and prepare the model slice for dependency analysis.

```
activate(obj);
```

Add the Out1 output block as the starting point and highlight the model slice.

```
addStartingPoint(obj, 'sldvSliceClimateControlExample/Out1');  
highlight(obj);
```

The area of the model upstream of the starting point and which is active during simulation is highlighted.

Terminate the model highlighting mode and discard the analysis data.

```
terminate(obj);
```

## Input Arguments

### **model** — Name or handle of model

character vector | string scalar

Name of the model whose Model Slicer options object you configure. `slicer` uses the Model Slicer configurations associated with `model`, as defined by `sliceroptions`.

### **opts** — Options you attach to a model or save to a file

`sliceroptions` object

Structure containing the options for the Model Slicer configuration. `sliceroptions` defines the options object `opts`.

## Output Arguments

### **obj** — Model Slicer object

`slicer` object

Model Slicer object handle.

## **See Also**

SLSlicerAPI.SLSlicer | slsliceroptions | slslicertrace

## **Topics**

“Programmatically Resolve Unexpected Behavior in a Model with Model Slicer”

“Workflow for Dependency Analysis”

“Configure Model Highlight and Sliced Models”

“Model Slicer Considerations and Limitations”

**Introduced in R2015b**



# sliceroptions

Create options object for configuring Model Slicer

## Syntax

```
sliceroptions  
sliceroptions(model)  
sliceroptions(file)  
sliceroptions(model,opts)  
sliceroptions(file,opts)
```

## Description

sliceroptions creates an options object for configuring the Model Slicer.

sliceroptions(model) creates a copy of the Model Slicer options object associated with model.

sliceroptions(file) creates a copy of the Model Slicer options object contained in the SLMS file file.

sliceroptions(model,opts) attaches the slicer options opts to the model model by overwriting the existing options.

sliceroptions(file,opts) attaches the slicer options opts to the SLMS file file by overwriting the existing options.

## Examples

### Add Starting Points and Exclusion Points to Active Configuration

Add a new starting point and a new exclusion point to the active Model Slicer configuration.

Open the f14 example model.

```
open_system('f14')
```

Define the options file `opts` for the model.

```
opts = slsliceroptions('f14')
```

Add a new starting point on the Gain block.

```
addStartingPoint(opts, 'f14/Gain')
```

Add a new exclusion point on the `alpha (rad)` block.

```
addExclusionPoint(opts, 'f14/alpha (rad)')
```

## **Add Starting Points and Exclusion Points to New Configuration**

Add a starting point and an exclusion point to the active Model Slicer configuration without overwriting the original configuration.

Open the f14 example model.

```
open_system('f14')
```

Define the options file `opts` for the model.

```
opts = slsliceroptions('f14')
```

Create a second Model Slicer options configuration for the model.

```
addConfiguration(opts)
```

Add a new starting point on the Gain block for the second Model Slicer options configuration.

```
addStartingPoint(opts.Configuration(2), 'f14/Gain')
```

Add a new exclusion point on the `alpha (rad)` block for the second Model Slicer options configuration.

```
addExclusionPoint(opts.Configuration(2), 'f14/alpha (rad)')
```

## Input Arguments

### **model** — Name or handle of model

character vector | string scalar

Name of the model whose Model Slicer options object you configure.

### **file** — Name of file

character vector | string scalar

Name of the SLMS file containing the Model Slicer options object that you configure.

Example: `sliceroptions('f14.slms')`

### **opts** — Options you attach to a model or save to a file

structure

Structure containing the options for the Model Slicer configuration.

## See Also

`sliceroptions` | `slicertrace`

## Topics

“Workflow for Dependency Analysis”

“Configure Model Highlight and Sliced Models”

“Model Slicer Considerations and Limitations”

**Introduced in R2015b**

## slslicertrace

Return block handles in sliced model or source model after using Model Slicer

### Syntax

```
slslicertrace('slice',object)  
slslicertrace('source',object)
```

### Description

`slslicertrace('slice',object)` returns the block handles in the sliced model that correspond to blocks specified by `object` in the source model.

`slslicertrace('source',object)` returns the block handles in the source model that correspond to blocks specified by `object` in the sliced model.

### Examples

#### Highlight a Block in the Source Model

Highlight the Switch block in the `sldvSliceClimateControlExample` source model.

Open the `sldvSliceClimateControlExample` example model.

```
addpath(fullfile(docroot,'toolbox','simulink','examples'));  
open_system('sldvSliceClimateControlExample');
```

Create a slicer object `obj` and add `Out1` as the starting point.

```
obj = slslicer('sldvSliceClimateControlExample');  
activate(obj);  
addStartingPoint(obj,'sldvSliceClimateControlExample/Out1');  
highlight(obj)
```

Create a sliced model by using `slice`.

```
slice(obj, 'sldvSliceClimateControlExample_sliced')
```

Highlight the On Switch block in the source model by using `slicertrace`

```
h=slicertrace('SOURCE', 'sldvSliceClimateControlExample_sliced/Refrigeration/On');  
hilite_system(h);  
terminate(obj);
```

## Input Arguments

### **object** — Object in source model or sliced model

character vector | cell array of character vectors | string array

An `object` can be specified as an array of block handles, cell arrays of block paths, or cell arrays of Simulink Identifiers (SID).

## See Also

`slicer` | `sliceroptions`

## Topics

“Workflow for Dependency Analysis”

“Configure Model Highlight and Sliced Models”

“Model Slicer Considerations and Limitations”

**Introduced in R2015b**

## **edittime.getAdvisorChecking**

Determine whether edit-time checking is on

### **Syntax**

```
val = edittime.getAdvisorChecking(model)
```

### **Description**

Use the `val = edittime.getAdvisorChecking(model)` returns whether edit-time checking is on or off.

### **Examples**

#### **Determine if edit-time checking is on**

Determine your model interactively.

Open the vdp example model.

```
vdp
```

Check whether edit-time checking is on.

```
edittime.getAdvisorChecking('vdp')
```

### **Input Arguments**

**model** — **model name**

character vector

Model for which you want to determine whether edit-time checking is on.

Data Types: char

## Output Arguments

**val** — Returns on or off

char

If edit-time checking is on, this function returns on. If edit-time checking is off, this function returns off.

## See Also

`edittime.setAdvisorChecking`

## Topics

“Check Model Compliance by Using the Model Advisor”

**Introduced in R2019a**

## **edittime.setAdvisorChecking**

Check your model interactively against modeling standards

### **Syntax**

```
edittime.setAdvisorChecking(model,value)
```

### **Description**

`edittime.setAdvisorChecking(model,value)` specifies whether to use the Model Advisor to check your model interactively against modeling standards. This function is the equivalent to selecting **Modeling > Model Advisor > Edit-Time Checks**.

### **Examples**

#### **Enable edit-time checking through the command line**

Check your model interactively.

Open the vdp example model.

```
vdp
```

Turn on edit-time checking.

```
edittime.setAdvisorChecking('vdp','on')
```

Confirm that edit-time checking is on.



```
edittime.getAdvisorChecking('vdp')
```

## Input Arguments

### **model** — model name

character vector

Model that you want to apply edit-time checking to

Data Types: char

### **value** — Turn edit-time checking on or off

character vector

To turn edit-time checking on, set `value` to `on`. To turn edit-time checking off, set the `value` to `off`.

Data Types: char

## See Also

`edittime.getAdvisorChecking`

## Topics

“Check Model Compliance by Using the Model Advisor”

**Introduced in R2019a**

## SLSlicerAPI.SLSlicer.addExclusionPoint

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Adds block handles, block paths, or Simulink Identifiers (SID) as slice exclusion point

### Syntax

```
addExclusionPoint(obj,ExclusionPoint)
```

### Description

`addExclusionPoint(obj,ExclusionPoint)` adds the `ExclusionPoint` as the exclusion point in the model slice for dependency analysis.

### Input Arguments

**obj** — Model Slicer configuration

slicer object

Class containing the options of Model Slicer configuration.

**ExclusionPoint** — Slice exclusion point

character vector | cell array of character vectors | string array

### Example

#### Exclusions at Block handles

Specifies the block handle. To get the block handle, use the `getSimulinkBlockHandle` command.

```
blkH = get_param('sldvSliceClimateControlExample/Out1','Handle');
addExclusionPoint(obj,blkH);
```

To add multiple block handles as exclusion point, use cell array, for example:

```
blkH = {get_param('sldvSliceClimateControlExample/Out1','Handle'),...
        get_param('sldvSliceClimateControlExample/Out2','Handle')};
addExclusionPoint(obj,blkH);
```

## Exclusions at Block Paths

Block path name, specified as a character vector or a cell array of character vectors.

```
bPath = {'sldvSliceClimateControlExample/Out1'};
addExclusionPoint(obj, bPath);
```

To add multiple block handles as exclusion point, use cell array, for example:

```
bPath = {'sldvSliceClimateControlExample/Out1',...
         'sldvSliceClimateControlExample/Heater/HeaterAct'};
addExclusionPoint(obj, bPath);
```

## Exclusions at SID

Simulink Identifier, a unique designation assigned to a Simulink block or model annotation. To get the SID, use the `Simulink.ID.getSID` command.

```
addExclusionPoint(obj, 'sldvSliceClimateControlExample:39')
```

## Exclusions at LineHs

Handles of line that connects from the Inport block to the Outport block. To get the Line Handle, use `get_param` command.

```
lh1 = get_param('sldvSliceClimateControlExample/Heater/Heat','LineHandles');
lh2 = get_param('sldvSliceClimateControlExample/Heater','LineHandles');
LineHs = [lh1.Inport(1), lh2.Outport(2)];
addExclusionPoint(obj,LineHs);
```

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select **Analysis > Model Slicer**. To add the block as the exclusion point, in the model, right-click the block and select **Model Slicer > Add as Exclusion Point**.

## **See Also**

“Highlight Functional Dependencies” |  
`SLSlicerAPI.SLSlicer.removeExclusionPoint`

**Introduced in R2015b**

# SLSlicerAPI.SLSlicer.removeExclusionPoint

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Removes the exclusion point from the model slice

## Syntax

```
removeExclusionPoint(obj,ExclusionPoint)
```

## Description

`removeExclusionPoint(obj,ExclusionPoint)` removes the `ExclusionPoint` from the model slice for dependency analysis.

## Input Arguments

**obj — Model Slicer configuration**

slicer object

Class containing the options of Model Slicer configuration.

**ExclusionPoint — Slice exclusion point**

character vector | cell array of character vectors | string array

Specify the slice exclusion point to remove from the model slice. Table summarizes the options for slice exclusion point.

Starting Point	Description
Block handles	<p>Specifies the block handle. To get the block handle, use the <code>getSimulinkBlockHandle</code> command.</p> <p>Example:</p> <pre>blkH = get_param('sldvSliceClimateControlExample/Out1', 'Handle'); removeExclusionPoint(obj, blkH);</pre> <p>To remove multiple block handles exclusion point, use cell array, for example:</p> <pre>blkH = {get_param('sldvSliceClimateControlExample/Out1', 'Handle'),         get_param('sldvSliceClimateControlExample/Out2', 'Handle')}; removeExclusionPoint(obj, blkH);</pre>
Block paths	<p>Block path name, specified as a character vector or a cell array of character vectors.</p> <p>Example:</p> <pre>bPath = {'sldvSliceClimateControlExample/Out1'}; removeExclusionPoint(obj, bPath);</pre> <p>To remove multiple block handles exclusion point, use cell array, for example:</p> <pre>bPath = {'sldvSliceClimateControlExample/Out1', ...          'sldvSliceClimateControlExample/Heater/HeaterAct'}; removeExclusionPoint(obj, bPath);</pre>
SID	<p>Simulink Identifier, a unique designation assigned to a Simulink block or model annotation. To get the SID, use the <code>Simulink.ID.getSID</code> command.</p> <p>Example:</p> <pre>removeExclusionPoint(obj, 'sldvSliceClimateControlExample/Out1');</pre>

Starting Point	Description
LineHs	<p>Handles of line that connects from the Inport block to the Outport block. To get the Line Handle, use <code>get_param</code> command.</p> <p>Example:</p> <pre>lh1 = get_param('sldvSliceClimateControlExample/Heater/Heat lh2 = get_param('sldvSliceClimateControlExample/Heater','Li LineHs = [lh1.Inport(1), lh2.Outport(2)]; removeExclusionPoint(obj,LineHs);</pre>

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select **Analysis > Model Slicer**. To add the block as the starting point, in the model, right-click the block and select **Model Slicer > Remove Exclusion Point**.

## See Also

“Highlight Functional Dependencies” | `SLSlicerAPI.SLSlicer.addExclusionPoint`

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.removeStartingPoint

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Removes the starting point from the model slice

### Syntax

```
removeStartingPoint(obj,StartingPoint)
```

### Description

`removeStartingPoint(obj,StartingPoint)` removes the `StartingPoint` from the model slice for dependency analysis.

### Input Arguments

**obj** — Model Slicer configuration

slslicer object

Class containing of Model Slicer configuration options.

**StartingPoint** — Slice starting point

character vector | cell array of character vectors | string array

### Examples

#### Remove Starting using Blocks Handles

Specifies the block handle. To get the block handle, use the `getSimulinkBlockHandle` command.



```
blkH = get_param('sldvSliceClimateControlExample/Out1','Handle');
removeStartingPoint(obj,blkH);
```

To remove multiple block handles starting point, use cell array, for example:

```
blkH = {get_param('sldvSliceClimateControlExample/Out1','Handle'),...
        get_param('sldvSliceClimateControlExample/Out2','Handle')};
removeStartingPoint(obj,blkH);
```

## Remove Starting using Blocks Paths

Block path name, specified as a character vector or a cell array of character vectors.

```
bPath = {'sldvSliceClimateControlExample/Out1'};
removeStartingPoint(obj, bPath);
```

To remove multiple block paths starting point, use cell array, for example:

```
bPath = {'sldvSliceClimateControlExample/Out1',...
        'sldvSliceClimateControlExample/Heater/HeaterAct'};
removeStartingPoint(obj, bPath);
```

## Remove Starting using SID

Simulink Identifier, a unique designation assigned to a Simulink block or model annotation. To get the SID, use the `Simulink.ID.getSID` command.

```
removeStartingPoint(obj, 'sldvSliceClimateControlExample:39')
```

## Remove Starting using LineHs

Handles of line that connects from the Inport block to the Outport block. To get the Line Handle, use `get_param` command.

```
lh1 = get_param('sldvSliceClimateControlExample/Heater/Heat','LineHandles');
lh2 = get_param('sldvSliceClimateControlExample/Heater','LineHandles');
LineHs = [lh1.Inport(1), lh2.Outport(2)];
removeStartingPoint(obj,LineHs);
```

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select **Analysis > Model Slicer**. To remove a block or signal as the starting point, in the model, right-click the block or signal and select **Model Slicer > Remove Starting Point**.

## **See Also**

“Highlight Functional Dependencies” | `SLSlicerAPI.SLSlicer.addStartingPoint`

**Introduced in R2015b**

# SLSlicerAPI.SLSlicer.addStartingPoint

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Add block handles, block paths, or Simulink Identifiers (SID) as the slice starting point

## Syntax

```
addStartingPoint(obj,StartingPoint)
```

## Description

`addStartingPoint(obj,StartingPoint)` adds `StartingPoint` as slice starting point for dependency analysis.

## Input Arguments

**obj** — Model Slicer configuration

*slslicer* object

Class containing Model Slicer configuration options.

**StartingPoint** — Slice starting point

character vector | cell array of character vectors | string array

## Example

### Starting Point at Block Handles

Specifies the block handle. To get the block handle, use the `getSimulinkBlockHandle` command.

```
blkH = get_param('sldvSliceClimateControlExample/Out1','Handle');  
addStartingPoint(obj,blkH);
```

To add multiple block handles as starting point, use cell array, for example:

```
blkH = {get_param('sldvSliceClimateControlExample/Out1','Handle'),...  
        get_param('sldvSliceClimateControlExample/Out2','Handle')};  
addStartingPoint(obj,blkH);
```

## Starting Point at Block Path

Block path name, specified as a character vector or a cell array of character vectors.

```
bPath = {'sldvSliceClimateControlExample/Out1'};  
addStartingPoint(obj, bPath);
```

To add multiple block paths as starting point, use cell array, for example:

```
bPath = {'sldvSliceClimateControlExample/Out1',...  
        'sldvSliceClimateControlExample/Heater/HeaterAct'};  
addStartingPoint(obj, bPath);
```

## Starting Point at SID

Simulink Identifier, a unique designation assigned to a Simulink block or model annotation. To get the SID, use the `Simulink.ID.getSID` command.

```
addStartingPoint(obj, 'sldvSliceClimateControlExample:39')
```

## Starting Point at LineHs

Handles of line that connects from the Inport block to the Outport block. To get the Line Handle, use `get_param` command.

```
lh1 = get_param('sldvSliceClimateControlExample/Heater/Heat','LineHandles');  
lh2 = get_param('sldvSliceClimateControlExample/Heater','LineHandles');  
LineHs = [lh1.Inport(1), lh2.Outport(2)];  
addStartingPoint(obj,LineHs);
```

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select **Analysis > Model Slicer**. To add a block or signal as the starting point, in the model, right-click the block and select **Model Slicer > Add as Starting Point**.

## See Also

“Highlight Functional Dependencies” |  
SLSlicerAPI.SLSlicer.removeStartingPoint

**Introduced in R2015b**

## **SLSlicerAPI.SLSlicer.activate**

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Activates the model for analysis

### **Syntax**

```
activate(obj)
```

### **Description**

activate(obj) activates the model for dependency analysis.

### **Input Arguments**

**obj** — Model Slicer configuration

slicer object

Object containing Model Slicer configuration options.

### **Example**

#### **Activate Slicer Object**

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
activate(obj);
```

#### **Terminate Slicer Object**

```
terminate(obj)
```

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. Invoking Model Slicer on the model automatically activates the model for analysis.

## See Also

`SLSlicerAPI.SLSlicer.simulate(t1,t2)` | `SLSlicerAPI.SLSlicer.terminate`

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.ActiveBlocks

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Returns the active non-virtual block handles

### Syntax

ActiveBlocks(obj)

### Description

ActiveBlocks(obj) returns the active non-virtual block handles.

### Input Arguments

**obj** — Model Slicer configuration

slicer object

Object containing Model Slicer configuration options.

### Example

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
activeblocks(obj);
```

### Alternatives

To open the Model Slicer manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. To view the active blocks, simulate the model while the model is in slicer mode.



## **See Also**

SLSlicerAPI.SLSlicer.activate | SLSlicerAPI.SLSlicer.terminate

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.highlight

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Updates the model highlighting

### Syntax

```
highlight(obj)
```

### Description

`highlight(obj)` updates the model highlighting.

### Input Arguments

**obj** — Model Slicer configuration

slicer object

Object containing Model Slicer configuration options.

### Example

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
highlight(obj);
```

### Alternatives

To open the Model Slicer manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. To highlight the active blocks, simulate the model while the model is in slicer mode.

## **See Also**

SLSlicerAPI.SLSlicer.unhighlight | SLSlicerAPI.SLSlicer.terminate

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.simulate(t1,t2)

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Simulates a test case for dynamic slicing from time "t1" to time "t2"

### Syntax

```
simulate(obj,t1,t2)
```

### Description

`simulate(obj,t1,t2)` simulates a test case for dynamic slicing from time "t1" to time "t2". If t1 is a non-zero value, the simulation first happens from zero to t2 and then the results reported are trimmed automatically for the specified duration t1 to t2.

### Input Arguments

**obj** — Model Slicer configuration

slicer object

Object containing Model Slicer configuration options.

### Example

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
simulate(obj);
```

### Alternatives

To open the Model Slice Manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. The initial simulation of the model in slicer mode gives you the option to

select the upper boundary of the simulation time (t2), with 0 being the lower boundary (t1). Post simulation and analysis you can view the simulation results for the selected time period (t1 to t2). To select the desired time period to view simulation results, from the **Model Slice Manager**, enter the t1 and t2 values in the **Time window** field and click **Highlight**.

## See Also

SLSlicerAPI.SLSlicer.terminate

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.terminate

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Discards the analysis data and reverts the model highlighting (invoked when the object goes out of scope)

### Syntax

```
terminate(obj)
```

### Description

`terminate(obj)` discards the analysis data and reverts the model highlighting (invoked when the object goes out of scope).

### Input Arguments

**obj** — Model Slicer configuration

slicer object

Object containing Model Slicer configuration options.

### Example

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
terminate(obj);
```

## Alternatives

To open the Model Slicer manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. To terminate slicer mode on the model, simply close the Model Slice Manager.

## See Also

SLSlicerAPI.SLSlicer.activate | SLSlicerAPI.SLSlicer.simulate(t1,t2)

**Introduced in R2015b**

## **SLSlicerAPI.SLSlicer.unhighlight**

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Removes the model highlighting without changing the activation status

### **Syntax**

```
unhighlight(obj)
```

### **Description**

`unhighlight(obj)` removes the model highlighting without changing the activation status.

### **Input Arguments**

**obj** — **Model Slicer configuration**

slicer object

Class containing Model Slicer configuration options.

### **Example**

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
unhighlight(obj);
```

### **See Also**

SLSlicerAPI.SLSlicer.highlight



**Introduced in R2015b**

## SLSlicerAPI.SLSlicer.unlock

**Class:** SLSlicerAPI.SLSlicer

**Package:** SLSlicerAPI

Discards the analysis data while retaining model highlights

### Syntax

```
unlock(obj)
```

### Description

`unlock(obj)` discards the analysis data while retaining model highlights.

### Input Arguments

**obj** — Model Slicer configuration


slicer object

Class containing Model Slicer configuration options.

### Example

```
open_system('sldvSliceClimateControlExample');  
obj=slicer('sldvSliceClimateControlExample');  
unlock(obj);
```

### Alternatives

To open the Model Slicer manager, in the Simulink Editor, select the **APPS** tab, and click Model Slicer. To unlock the model, click  on the **Model Slice Manager** window.

## **See Also**

SLSlicerAPI.SLSlicer.simulate(t1,t2) | SLSlicerAPI.SLSlicer.terminate

**Introduced in R2015b**

## SLSlicerAPI.SLSlicer class

**Package:** SLSlicerAPI

Class representing a set of Model Slicer functionality

### Description

The SLSlicerAPI.SLSlicer class provides set of methods to access Model Slicer functionality from the MATLAB command line. Use an instance of SLSlicerAPI.SLSlicer class to:

- Activate or terminate slice simulation.
- Add or remove starting point, exclusion point, constraints, slice component and configuration.
- Highlight the functional dependencies.
- Set time window and simulate slice.
- Refine slice for dead logic analysis.

### Construction

`slslicer(model)` creates an API object for the model `model` by exposing the methods for invoking Model Slicer. Uses the Model Slicer configurations associated with `model`, as defined by `slsliceroptions`.

`slslicer(model, opts)` creates an API object `model` by using the options object `opts`, as defined by `slsliceroptions`.

### Input Arguments

**model** — Name or handle of model

character vector | string scalar

Name of the model whose Model Slicer options object you configure. `slslicer` uses the Model Slicer configurations associated with the `model`, as defined by `slsliceroptions`.

Example: `sldvSliceClimateControlExample`

Data Types: `char | string`

**opts — Options you attach to a model or save to a file**  
structure

Structure containing the options for the Model Slicer configuration. `slsliceroptions` defines the options object `opts`.

Example: `slsliceroptions`

Data Types: `struct`

## Methods

<code>addExclusionPoint</code>	Adds block handles, block paths, or Simulink Identifiers (SID) as slice exclusion point
<code>removeExclusionPoint</code>	Removes the exclusion point from the model slice
<code>removeStartingPoint</code>	Removes the starting point from the model slice
<code>addStartingPoint</code>	Add block handles, block paths, or Simulink Identifiers (SID) as the slice starting point
<code>activate</code>	Activates the model for analysis
<code>ActiveBlocks</code>	Returns the active non-virtual block handles
<code>highlight</code>	Updates the model highlighting
<code>simulate(t1,t2)</code>	Simulates a test case for dynamic slicing from time "t1" to time "t2"
<code>terminate</code>	Discards the analysis data and reverts the model highlighting (invoked when the object goes out of scope)
<code>unhighlight</code>	Removes the model highlighting without changing the activation status
<code>unlock</code>	Discards the analysis data while retaining model highlights

**Additional Methods**

Method	Description
slice	<p>Creates a sliced model from the model highlight.</p> <p>Example:  <code>slice(obj, 'sldvSliceClimateControlExample_sliced')</code></p>
addConstraint	<p>Adds the constraint on these blocks:</p> <ul style="list-style-type: none"> <li>• Switch or Multiport switch</li> <li>• Stateflow state or transition</li> </ul> <p>Example:  <code>bpath={'sldvSliceClimateControlExample/Refrigeration/On'};</code>  <code>addConstraint(obj,bpath,{1, 1})</code></p>
removeConstraint	<p>Removes the constraint from the model slice.</p> <p>Example:  <code>removeConstraint(obj,bpath)</code></p>
addSliceComponent	<p>Adds a model or a subsystem as a slice component.</p> <p>Example:  <code>addSliceComponent(obj, 'sldvdemo_cruise_control/Controller/PIController')</code></p>
removeSliceComponent	<p>Removes the slice component from the model slice.</p> <p>Example: <code>removeSliceComponent(obj);</code></p>

Method	Description
refineDeadLogic	Updates the model highlighting with dead logic refinement.  Example: analysis_time=100;  refineDeadLogic(obj, 'sldvSlicerdemo_dead_logic', analysis_time)
removeDeadLogic	Removes the dead logic refinement.  Example: removeDeadLogic(obj, 'sldvSlicerdemo_dead_logic')

## Object Functions

slicer Create API object for invoking Model Slicer

## Examples

### Add Starting Point and Highlight the Model Slice

Add a new starting point to the active Model Slicer configuration, and then highlight the model.

Open the sldvSliceClimateControlExample example model.

```
addpath(fullfile(docroot, 'toolbox', 'simulink', 'examples'));
open_system('sldvSliceClimateControlExample');
```

Create a Model Slicer configuration object for the model by using slicer.

```
obj = slicer('sldvSliceClimateControlExample');
```

To compile the model and prepare the model slice for dependency analysis, activate the slice highlighting mode of Model Slicer.

```
activate(obj);
```

Add the `Out1` output block as the starting point and highlight the model slice.

```
addStartingPoint(obj, 'sldvSliceClimateControlExample/Out1');  
highlight(obj);
```

The area of the model upstream of the starting point and which is active during simulation is highlighted.

Terminate the model highlighting mode and discard the analysis data.

```
terminate(obj);
```

## See Also

“Programmatically Resolve Unexpected Behavior in a Model with Model Slicer”

**Introduced in R2015b**



# Model Advisor Checks

---

- “Simulink Check Checks” on page 2-2
- “DO-178C/DO-331 Checks” on page 2-8
- “High Integrity System Modeling Checks” on page 2-24
- “IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks” on page 2-149
- “MathWorks Automotive Advisory Board Checks” on page 2-165
- “Japan MATLAB Automotive Advisory Board Checks” on page 2-261
- “MISRA C:2012 Checks” on page 2-377
- “Secure Coding Checks for CERT C, CWE, and ISO/IEC TS 17961 Standards” on page 2-396
- “Model Metrics” on page 2-419

## Simulink Check Checks

In this section...
“Simulink Check Checks” on page 2-2
“Simulink Requirements Checks” on page 2-3
“Modeling Standards Checks” on page 2-3
“Modeling Standards for MAAB” on page 2-3
“Naming Conventions” on page 2-4
“Model Architecture” on page 2-4
“Model Configuration Options” on page 2-5
“Simulink” on page 2-5
“ Stateflow ” on page 2-5
“MATLAB Functions” on page 2-6
“Modeling Standards for JMAAB” on page 2-6



### Simulink Check Checks

Simulink Check checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements and modeling guidelines.

For descriptions of the modeling standards checks, see

- “DO-178C/DO-331 Checks” on page 2-8
- “IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks” on page 2-149
- “MathWorks Automotive Advisory Board Checks” on page 2-165
- “Japan MATLAB Automotive Advisory Board Checks” on page 2-261
- “MISRA C:2012 Checks” on page 2-377
- “Secure Coding Checks for CERT C, CWE, and ISO/IEC TS 17961 Standards” on page 2-396

#### See Also

- “Run Model Advisor Checks and Review Results”

- “Simulink Checks” (Simulink)

## Simulink Requirements Checks

Simulink Requirements™ checks facilitate linking between requirements documentation and your model .

For descriptions of the requirements consistency checks, see “Requirements Consistency Checks” (Simulink Requirements).

### See Also

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)

## Modeling Standards Checks

Modeling standards checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements or MathWorks® Automotive Advisory Board (MAAB) modeling guidelines.

The Model Advisor performs a checkout of the Simulink Check license when you run the modeling standards checks.

For descriptions of the modeling standards checks, see

- “DO-178C/DO-331 Checks” on page 2-8
- “IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks” on page 2-149
- “MathWorks Automotive Advisory Board Checks” on page 2-165
- “Japan MATLAB Automotive Advisory Board Checks” on page 2-264

### See Also

- “Run Model Advisor Checks and Review Results”

## Modeling Standards for MAAB

Group of MathWorks Automotive Advisory Board (MAAB) checks. MAAB checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run the modeling standards for MAAB checks.

### See Also

- “Run Model Advisor Checks and Review Results”
- “Model Advisor Checks for MAAB Guidelines” (Simulink)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines
- The *Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow* MAAB guidelines on the MathWorks website

## Naming Conventions

Group of MathWorks Automotive Advisory Board (MAAB) checks related to naming conventions.

The Model Advisor performs a checkout of the Simulink Check license when you run the naming conventions checks.

### See Also

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## Model Architecture

Group of MathWorks Automotive Advisory Board (MAAB) checks related to model architecture.

The Model Advisor performs a checkout of the Simulink Check license when you run the model architecture checks.

### See Also

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)

- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## **Model Configuration Options**

Group of MathWorks Automotive Advisory Board (MAAB) checks related to model configuration options.

The Model Advisor performs a checkout of the Simulink Check license when you run the model configuration options checks.

### **See Also**

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## **Simulink**

Group of MathWorks Automotive Advisory Board (MAAB) checks related to the Simulink product.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to the Simulink product.

### **See Also**

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## **Stateflow**

Group of MathWorks Automotive Advisory Board (MAAB) checks related to the Stateflow product.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to the Stateflow product.

### **See Also**

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## **MATLAB Functions**

MathWorks Automotive Advisory Board (MAAB) checks related to MATLAB functions.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks related to MATLAB functions.

### **See Also**

- “Run Model Advisor Checks and Review Results”
- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- “MAAB Control Algorithm Modeling” (Simulink) guidelines

## **Modeling Standards for JMAAB**

Group of MathWorks Japan MATLAB Automotive Advisory Board (JMAAB) checks. JMAAB checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run the modeling standards for JMAAB checks.

### **See Also**

- “Run Model Advisor Checks and Review Results”
- “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”

- The *Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow* JMAAB guidelines on the MathWorks website

# DO-178C/DO-331 Checks

In this section...
“DO-178C/DO-331 Checks” on page 2-8
“Check safety-related code generation settings” on page 2-8
“Check usage of Math Operations blocks” on page 2-14
“Check usage of Logic and Bit Operations blocks” on page 2-17
“Check usage of Ports and Subsystems blocks” on page 2-19
“Display model version information” on page 2-22

## DO-178C/DO-331 Checks

DO-178C/DO-331 checks facilitate designing and troubleshooting models from which code is generated for applications that must meet safety or mission-critical requirements.

The Model Advisor performs a checkout of the Simulink Check license when you run the DO-178C/DO-331 checks.

These checks are qualified by the DO Qualification Kit for use in projects involving the DO-178 standard and related standards.

### See Also

- “Simulink Checks” (Simulink)
- “Simulink Coder Checks” (Simulink Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

## Check safety-related code generation settings

**Check ID:** `mathworks.do178.CodeSet`

Check model configuration for code generation settings that can impact safety.



## Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The option to include comments in the generated code is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Select <b>Include comments</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>GenerateComments</code> to on.
The option to include comments that describe the code for blocks is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Select <b>Simulink block comments</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>SimulinkBlockComments</code> to on.
The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Select <b>Show eliminated blocks</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ShowEliminatedStatement</code> to on.
The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in <code>model_prm.h</code> is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Select <b>Verbose comments for 'Model default' storage class</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ForceParamTrailComments</code> to on.

Condition	Recommended Action
<p>The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments provide good traceability between the code and the model. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)</p>	<p>Select <b>Requirements in block comments</b> (Simulink Coder) on the <b>Code Generation &gt; Custom comments</b> pane in the Configuration Parameters dialog box or set the parameter ReqsInCode to on.</p>
<p>The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Support: non-finite numbers</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter SupportNonFinite to off.</p>
<p>The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Support: absolute time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter SupportAbsoluteTime to off.</p>
<p>The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Support: continuous time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter SupportContinuousTime to off.</p>

Condition	Recommended Action
<p>The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Support: non-inlined S-functions</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter <code>SupportNonInlinedSFcns</code> to off.</p>
<p>The option to generate model function calls compatible with the main program module of the pre-R2012a GRT target is selected. This option is inappropriate for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Classic call call interface</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>GRTInterface</code> to off.</p>
<p>The option to generate the <code>model_update</code> function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Select <b>Single output/update function</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>CombineOutputUpdateFcns</code> to on.</p>
<p>The option to generate the <code>model_terminate</code> function is selected. This function deallocates dynamic memory, which is unsuitable for real-time safety-related systems. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>Terminate function</b> (Simulink Coder) on the <b>Code Generation</b> pane in the Configuration Parameters dialog box or set the parameter <code>IncludeMdlTerminateFcn</code> to off.</p>

Condition	Recommended Action
<p>The option to log or monitor error status is cleared. If you do not select this option, the Simulink Coder™ product generates extra code that might not be reachable for testing. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Select <b>Remove error status field in real-time model data structure</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter SuppressErrorStatus to on.</p>
<p>MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer and DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer.)</p>	<p>Clear <b>MAT-file logging</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter MatFileLogging to off.</p>
<p>The option that specifies the style for parenthesis usage is set to <b>Minimum (Rely on C/C++ operators precedence)</b> or to <b>Nominal (Optimize for readability)</b>. For safety-related applications, explicitly specify precedence with parentheses. (See DO-331, Section MB.6.3.1.c - High-level requirements are compatible with target computer, DO-331, Section MB.6.3.2.c - Low-level requirements are compatible with target computer, and MISRA C:2012, Rule 12.1.)</p>	<p>Set parameter ParenthesesLevel to <b>Maximum (Specify precedence with parentheses)</b>.</p>
<p>The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)</p>	<p>Set parameter PreserveExpressionOrder to on.</p>

Condition	Recommended Action
The option that specifies whether to preserve empty primary condition expressions in <code>if</code> statements is cleared. This option increases the traceability of the generated code. ( See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Set parameter <code>PreserveIfCondition</code> to on.
The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews. (See DO-331, Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	Set <b>Minimum mangle length</b> (Simulink Coder) on the <b>Code Generation &gt; Symbols</b> pane in the Configuration Parameters dialog box or the parameter <code>MangleLength</code> to a value of 4 or greater.

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- “hisl\_0038: Configuration Parameters > Code Generation > Comments” (Simulink)
- “hisl\_0039: Configuration Parameters > Code Generation > Interface” (Simulink)
- “hisl\_0047: Configuration Parameters > Code Generation > Code Style” (Simulink)
- “hisl\_0049: Configuration Parameters > Code Generation > Identifiers” (Simulink)

- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Identifiers” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Style” (Embedded Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

### **Check usage of Math Operations blocks**

**Check ID:** `mathworks.do178.MathOperationsBlocksUsage`

Identify usage of Math Operation blocks that might impact safety.

#### **Description**

This check inspects the usage of the following blocks:

- Abs
- Gain
- Math Function
  - Natural logarithm
  - Common (base 10) logarithm
  - Remainder after division
  - Reciprocal
- Assignment

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains an Absolute Value block that is operating on one of the following:</p> <ul style="list-style-type: none"> <li>• A boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code</li> <li>• A signed integer value with the <b>Saturate on integer overflow</b> check box not selected. For signed data types, the absolute value of the most negative value is problematic because it is not representable by the data type. This condition results in an overflow in the generated code.</li> </ul>	<p>If the identified Absolute Value block is operating on a boolean or unsigned data type, do one of the following:</p> <ul style="list-style-type: none"> <li>• Change the input of the Absolute Value block to a signed input type.</li> <li>• Remove the Absolute Value block from the model.</li> </ul> <p>If the identified Absolute Value block is operating on a signed data type, in the <b>Block Parameters &gt; Signal Attributes</b> dialog box, select <b>Saturate on integer overflow</b>.</p>
<p>The model or subsystem contains Gain blocks with a of value 1 or an identity matrix.</p>	<p>If you are using Gain blocks as buffers, consider replacing them with Signal Conversion blocks.</p>
<p>The model or subsystem contains Math Function - Natural logarithm (<b>log</b>) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.</p>	<p>When using the Math Function block with a <b>log</b> function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a NaN or -Inf and result in a run-time error in the generated code.</p>
<p>The model or subsystem contains Math Function - Common (base 10) (<b>base 10 logarithm</b>) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.</p>	<p>When using the Math Function block with a <b>log10</b> function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a NaN or -Inf and result in a run-time error in the generated code.</p>

Condition	Recommended Action
The model or subsystem contains Math Function - Remainder after division ( <code>rem</code> ) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a <code>rem</code> function, protect the second input to the block from being equal to zero. Otherwise the output can produce a <code>Inf</code> or <code>-Inf</code> and result in a run-time error in the generated code.
The model or subsystem contains Math Function - Reciprocal ( <code>reciprocal</code> ) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a <code>reciprocal</code> function, protect the input to the block from being equal to zero. Otherwise the output can produce a <code>Inf</code> or <code>-Inf</code> and result in a run-time error in the generated code.
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter <b>Action if any output element is not assigned</b> set to <b>Error</b> or <b>Warning</b> .	Set block parameter <b>Action if any output element is not assigned</b> to one of the recommended values: <ul style="list-style-type: none"> <li>• <b>Error</b>, if Assignment block is not in an Iterator subsystem.</li> <li>• <b>Warning</b>, if Assignment block is in an Iterator subsystem.</li> </ul>

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- DO-331 Section MB.6.3.1.d - High-level requirements are verifiable
- DO-331 Section MB.6.3.2.d - Low-level requirements are verifiable
- MISRA C:2012, Dir 4.1
- MISRA C:2012, Rule 9.1
- hisl\_0001: Usage of Abs block



- hisl\_0002: Usage of Math Function blocks (rem and reciprocal)
- hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)
- hisl\_0029: Usage of Assignment blocks
- hisl\_0066: Usage of Gain blocks

## Check usage of Logic and Bit Operations blocks

**Check ID:** mathworks.do178.LogicBlockUsage

Identify usage of Logical Operator and Bit Operations blocks that might impact safety.

### Description

This check inspects the usage of:

- Blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare To Zero, Detect Change, and If blocks
- Logical Operator blocks

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.	For the identified blocks, use common data types as inputs. You can use Data Type Conversion blocks to change input data types.
The model or subsystem contains a block computing a relational operator that does not have Boolean output. The condition can lead to unpredictable results in the generated code.	For the specified blocks, on the Block Parameters > Signal Attributes pane, set the <b>Output data type</b> to boolean.

Condition	Recommended Action
<p>The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues. These operators can lead to unpredictable results in the generated code.</p>	<p>For the identified block, do one of the following:</p> <ul style="list-style-type: none"> <li>• Change the signal data type.</li> <li>• Rework the model to eliminate using == or ~= operators on floating-point signals.</li> </ul>
<p>The model or subsystem contains a Logical Operator block that has inputs or outputs that are not Boolean inputs or outputs. The block might result in floating-point equality or inequality comparisons in the generated code.</p>	<ul style="list-style-type: none"> <li>• Modify the Logical Operator block so that all inputs and outputs are Boolean. On the Block Parameters &gt; Signal Attributes pane, consider selecting <b>Require all inputs to have the same data type</b> and setting <b>Output data type</b> to boolean.</li> <li>• In the Configuration Parameters dialog box, consider selecting the <b>Implement logic signals as boolean data (vs. double)</b>.</li> </ul>

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- MISRA C:2012, Dir 1.1
- MISRA C:2012, Rule 10.1
- “hisl\_0016: Usage of blocks that compute relational operators” (Simulink)
- “hisl\_0017: Usage of blocks that compute relational operators (2)” (Simulink)

- “hisl\_0018: Usage of Logical Operator block” (Simulink)

## **Check usage of Ports and Subsystems blocks**

**Check ID:** `mathworks.do178.PortsSubsystemsUsage`

Identify usage of Ports and Subsystems blocks that might impact safety.

### **Description**

This check inspects the usage of these blocks:

- For Iterator
- While Iterator
- If
- Switch Case

The check does not flag Switch Case blocks that do not use integer data types or enumeration values for inputs. To comply with “hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks” (Simulink) - C, use an integer data type or an enumeration value for the inputs to Switch Case blocks.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
<p>The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.</p>	<p>For the identified For Iterator blocks, do one of the following:</p> <ul style="list-style-type: none"> <li>• Set the <b>Iteration limit source</b> parameter to <code>internal</code>.</li> <li>• If the <b>Iteration limit source</b> parameter must be <code>external</code>, use a Constant, Probe, or Width block as the source.</li> <li>• Clear the <b>Set next i (iteration variable) externally</b> check box.</li> <li>• Consider selecting the <b>Show iteration variable</b> check box and observe the iteration value during simulation.</li> </ul>
<p>The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code.</p>	<p>For the identified While Iterator blocks:</p> <ul style="list-style-type: none"> <li>• Set the <b>Maximum number of iterations (-1 for unlimited)</b> parameter to a positive integer value.</li> <li>• Consider selecting the <b>Show iteration number port</b> check box and observe the iteration value during simulation.</li> </ul>
<p>The model or subsystem contains an If block with an If expression or Elseif expressions that might cause floating-point equality or inequality comparisons in generated code.</p>	<p>Modify the expressions in the If block to avoid floating-point equality or inequality comparisons in generated code.</p>
<p>The model or subsystem contains an If block using Elseif expressions without an Else condition.</p>	<p>In the If block Block Parameters dialog box, select <b>Show else condition</b>. Connect the resulting Else output port to an If Action Subsystem block.</p>
<p>The model or subsystem contains an If block with output ports that do not connect to If Action Subsystem blocks.</p>	<p>Verify that output ports of the If block connect to If Action Subsystem blocks.</p>

Condition	Recommended Action
The model or subsystem contains an Switch Case block without a default case.	In the Switch Case block Block Parameters dialog box, select <b>Show default case</b> . Connect the resulting default output port to a Switch Case Action Subsystem block.
The model or subsystem contains a Switch Case block with an output port that does not connect to a Switch Case Action Subsystem block.	Verify that output ports of the Switch Case blocks connect to Switch Case Action Subsystem blocks.
<p>The model or subsystem contains one of the following time-dependent blocks in a For Iterator or While Iterator subsystem:</p> <ul style="list-style-type: none"> <li>• Discrete Filter</li> <li>• Discrete FIR Filter</li> <li>• Discrete State-Space</li> <li>• Discrete Transfer Fcn</li> <li>• Discrete Zero-Pole</li> <li>• Transfer Fcn First Order</li> <li>• Transfer Fcn Lead or Lag</li> <li>• Transfer Fcn Real Zero</li> <li>• Discrete Derivative</li> <li>• Discrete Transfer Fcn (with initial outputs)</li> <li>• Discrete Transfer Fcn (with initial states)</li> <li>• Discrete Zero-Pole (with initial outputs)</li> <li>• Discrete Zero-Pole (with initial states)</li> </ul>	In the model or subsystem, consider removing the time-dependent blocks.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.

### See Also

- DO-331, Section MB.6.3.3.b—Software architecture is consistent
- DO-331, Sections MB.6.3.1.g and MB.6.3.2.g - Algorithms are accurate
- DO-331, Section MB.6.3.1.e - High-level requirements conform to standards
- DO-331, Section MB.6.3.2.e - Low-level requirements conform to standards
- DO-331, Section MB.6.3.1.b - High-level requirements are accurate and consistent
- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- MISRA C:2012, Rule 14.2
- MISRA C:2012, Rule 16.4
- MISRA C:2012, Dir 4.1
- “hisl\_0006: Usage of While Iterator blocks” (Simulink)
- “hisl\_0007: Usage of For Iterator or While Iterator subsystems” (Simulink)
- “hisl\_0008: Usage of For Iterator Blocks” (Simulink)
- “hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks” (Simulink)

## Display model version information

**Check ID:** `mathworks.do178.MdlChecksum`

Display model version information in your report.

### Description

This check displays the following information for the current model:

- Version number
- Author
- Date
- Model checksum

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Could not retrieve model version and checksum information.	This summary is provided for your information. No action is required.

## Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

## See Also

- “Reports for Code Generation” (Simulink Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards

## High Integrity System Modeling Checks

**In this section...**

- “High-Integrity Systems Modeling Checks” on page 2-27
- “Split Checks for High Integrity Systems Modeling” on page 2-28
- “Check for inconsistent vector indexing methods” on page 2-30
- “Check for root Inports with missing properties” on page 2-32
- “Check for root Inports with missing range definitions” on page 2-33
- “Check for root Outports with missing range definitions” on page 2-35
- “Check safety-related diagnostic settings for data store memory” on page 2-36
- “Check safety-related diagnostic settings for data used for debugging” on page 2-38
- “Check safety-related diagnostic settings for parameters” on page 2-39
- “Check safety-related solver settings for solver options” on page 2-41
- “Check safety-related solver settings for tasking and sample-time” on page 2-42
- “Check usage of shift operations for Stateflow data” on page 2-42
- “Check usage of Signal Routing blocks” on page 2-43
- “Check state machine type of Stateflow charts” on page 2-44
- “Check Stateflow charts for ordering of states and transitions” on page 2-46
- “Check Stateflow charts for strong data typing” on page 2-47
- “Check Stateflow charts for unary operators” on page 2-48
- “Check Stateflow charts for uniquely defined data objects” on page 2-49
- “Check Stateflow debugging options” on page 2-50
- “Check usage of lookup table blocks” on page 2-51
- “Check for variant blocks with 'Generate preprocessor conditionals' active” on page 2-53
- “Check safety-related diagnostic settings for signal connectivity” on page 2-54
- “Check safety-related diagnostic settings for bus connectivity” on page 2-55
- “Check safety-related diagnostic settings that apply to function-call connectivity” on page 2-57
- “Check safety-related diagnostic settings for compatibility” on page 2-58



**In this section...**

- "Check safety-related diagnostic settings for model initialization" on page 2-59
- "Check safety-related diagnostic settings for saving" on page 2-61
- "Check MATLAB Code Analyzer messages" on page 2-62
- "Check safety-related diagnostic settings for Merge blocks" on page 2-64
- "Check safety-related diagnostic settings for Stateflow" on page 2-65
- "Check for Strong Data Typing with Simulink I/O" on page 2-67
- "Check for MATLAB Function interfaces with inherited properties" on page 2-68
- "Check usage of Abs blocks" on page 2-69
- "Check usage of Math Function blocks (rem and reciprocal functions)" on page 2-71
- "Check usage of Math Function blocks (log and log10 functions)" on page 2-72
- "Check usage of Assignment blocks" on page 2-73
- "Check safety-related optimization settings for data type conversions" on page 2-74
- "Check safety-related optimization settings for data initialization" on page 2-75
- "Check safety-related optimization settings for application lifespan" on page 2-77
- "Check safety-related block reduction optimization settings" on page 2-78
- "Check safety-related optimization settings for logic signals" on page 2-79
- "Check safety-related optimization settings for division arithmetic exceptions" on page 2-80
- "Check usage of Logical Operator blocks" on page 2-81
- "Check for Relational Operator blocks that equate floating-point types" on page 2-82
- "Check usage of Relational Operator blocks" on page 2-83
- "Check usage of Switch Case blocks and Switch Case Action Subsystem blocks" on page 2-84
- "Check usage of If blocks and If Action Subsystem blocks" on page 2-85
- "Check usage of For Iterator blocks" on page 2-86
- "Check usage of For and While Iterator subsystems" on page 2-87
- "Check usage of While Iterator blocks" on page 2-89
- "Check safety-related code generation settings for comments" on page 2-90
- "Check safety-related code generation interface settings" on page 2-91

### **In this section...**

- "Check safety-related code generation settings for code style" on page 2-93
- "Check safety-related code generation identifier settings" on page 2-95
- "Check safety-related diagnostic settings for model referencing" on page 2-96
- "Check safety-related diagnostic settings for sample time" on page 2-98
- "Check MATLAB Function metrics" on page 2-100
- "Check safety-related diagnostic settings for type conversions" on page 2-102
- "Check safety-related solver settings for simulation time" on page 2-103
- "Check safety-related diagnostic settings for solvers" on page 2-104
- "Check safety-related model referencing settings" on page 2-106
- "Check Stateflow charts for transition paths that cross parallel state boundaries" on page 2-108
- "Check assignment operations in Stateflow Charts" on page 2-109
- "Check usage of bit operation blocks" on page 2-110
- "Check data types for blocks with index signals" on page 2-111
- "Check model file name" on page 2-112
- "Check if/elseif/else patterns in MATLAB Function blocks" on page 2-113
- "Check switch statements in MATLAB Function blocks" on page 2-114
- "Check global variables in graphical functions" on page 2-115
- "Check for length of user-defined object names" on page 2-116
- "Check usage of Merge blocks" on page 2-116
- "Check usage of conditionally executed subsystems" on page 2-117
- "Check usage of standardized MATLAB function headers" on page 2-120
- "Check usage of relational operators in MATLAB Function blocks" on page 2-121
- "Check usage of equality operators in MATLAB Function blocks" on page 2-122
- "Check usage of logical operators and functions in MATLAB Function blocks" on page 2-123
- "Check type and size of condition expressions" on page 2-124
- "Check naming of ports in Stateflow charts" on page 2-126
- "Check scoping of Stateflow data objects" on page 2-127

**In this section...**

“Check usage of Gain blocks” on page 2-127

“Check data type of loop control variables” on page 2-128

“Check for inappropriate use of transition paths” on page 2-129

“Check usage of bitwise operations in Stateflow charts” on page 2-130

“Check safety-related diagnostic settings for signal data” on page 2-131

“Check for model elements that do not link to requirements” on page 2-134

“Check safety-related optimization settings for Loop unrolling threshold” on page 2-136

“Check safety-related optimization settings for specified minimum and maximum values” on page 2-137

“Check model object names” on page 2-138

“Check for blocks not recommended for C/C++ production code deployment” on page 2-141

“Check configuration parameters for MISRA C:2012” on page 2-142

“Check for blocks not recommended for MISRA C:2012” on page 2-146

## High-Integrity Systems Modeling Checks

High-Integrity Systems Modeling checks facilitate designing and troubleshooting models, subsystems, and the corresponding generated code for applications to comply with High-Integrity Systems Modeling Guidelines.

The Model Advisor performs a checkout of the Simulink Check license when you run the High-Integrity Systems Modeling checks.

High-Integrity Systems Modeling checks are classified in to the following groups:

### Simulink

High-Integrity Systems Modelling checks for Simulink blocks and components.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### Stateflow

High-Integrity Systems Modelling checks for Stateflow charts and components.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### **MATLAB**

High-Integrity Systems Modelling checks for MATLAB code.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### **Configuration**

High-Integrity Systems Modelling checks for Configuration settings.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### **Naming**

High-Integrity Systems Modelling checks for Naming conventions.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### **Requirements**

High-Integrity Systems Modelling checks for Simulink Requirements traceability.

For more information, see “Model Checks for High Integrity Systems Modeling”.

### **Code**

High-Integrity Systems Modelling checks for generated code .

For more information, see “Model Checks for High Integrity Systems Modeling”.

## **Split Checks for High Integrity Systems Modeling**

From R2018b and later, the following checks are not recommended for use. These checks are split into multiple checks that focus on a single action or operation. For more information, see the Split and New Checks table below.

<b>Old Check Title</b>	<b>Split Check Titles</b>
Check usage of Math Operations blocks	Check usage of Abs blocks

Old Check Title	Split Check Titles
	Check usage of Math Function blocks (rem and reciprocal functions)
	Check usage of Math Function blocks (log and log10 functions)
	Check usage of Assignment blocks
Check usage of Logic and Bit Operations blocks	Check for Relational Operator blocks that equate floating-point types
	Check usage of Relational Operator blocks
	Check usage of Logical Operator blocks
Check usage of Ports and Subsystems blocks	Check usage of While Iterator blocks
	Check sample time-dependent blocks
	Check usage of For Iterator blocks
	Check usage of If blocks and If Action Subsystem blocks
	Check usage Switch Case blocks and Switch Case Action Subsystem blocks
Check safety-related code generation settings	Check safety-related code generation settings for comments
	Check safety-related code generation interface settings
	Check safety-related code generation settings for code style
	Check safety-related code generation symbols settings
Check usage of Stateflow constructs	Check usage of Stateflow constructs
	Check Stateflow charts for ordering of states and transitions
	Check Stateflow debugging options
	Check Stateflow charts for uniquely defined data objects

Old Check Title	Split Check Titles
Check safety-related optimization settings	Check safety-related optimization settings for logic signals
	Check safety-related block reduction optimization settings
	Check safety-related optimization settings for application lifespan
	Check safety-related optimization settings for data initialization
	Check safety-related optimization settings for data type conversions
	Check safety-related optimization settings for division arithmetic exceptions

## Check for inconsistent vector indexing methods

**Check ID:** `mathworks.hism.hisl_0021`

Identify blocks with inconsistent indexing method.

### Description

Using inconsistent block indexing methods can result in modeling errors. You should use a consistent vector indexing method for all blocks. The indexing methods are zero-based, one-based or user-specified.

Blocks that support configurable indexing:

- Assignment
- For Iterator
- Index Vector
- Multiport Switch
- Selector

Blocks that support only one-based indexing:

- Fcn (deprecated)

- MATLAB Function
- MATLAB System
- State Transition Table
- Test Sequence
- Truth Table
- Stateflow chart with MATLAB action language
- Truth Table function with MATLAB action language

Blocks that supports only zero-based indexing:

- Stateflow chart with C action language
- Truth Table function with C action language

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks with inconsistent indexing methods. The indexing methods are zero-based, one-based or user-specified.	Modify the model to use a single consistent indexing method.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0021: Consistent vector indexing method

## Check for root Inports with missing properties

**Check ID:** `mathworks.hism.hisl_0024`

Identify root model Inport blocks with missing or inherited sample times, data types or port dimensions.

### Description

Using root model Inport blocks that do not have defined sample time, data types or port dimensions can lead to undesired simulation results. Simulink back-propagates dimensions, sample times, and data types from downstream blocks unless you explicitly assign these values. You can specify Inport block properties with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. When you run the check, a results table provides links to Inport blocks and signal objects that do not pass, along with conditions triggering the warning.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<b>Missing port dimension</b> — Model contains Inport blocks with inherited port dimensions.	For the listed Inport blocks and Simulink signal objects, specify port dimensions.
<b>Missing signal data type</b> — Model contains Inport blocks with inherited data types.	For the listed Inport blocks and Simulink signal objects, specify data types.
<b>Missing port sample time</b> — Model contains Inport blocks with inherited sample times.	For the listed Inport blocks and Simulink signal objects, specify sample times. The sample times for root Inports with bus type must match the sample times specified at the leaf elements of the bus object.
<b>Implicit resolution to a Simulink signal object</b> — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property <b>Signal name must resolve to Simulink signal object</b> . To set this option programmatically, use the port parameter <b>MustResolveToSignalObject</b> .



### Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

### Tips

The following configurations pass this check:

- **Configuration Parameters > Solver > Periodic sample time constraint** is set to Ensure sample time independent
- For export-function models, *inherited sample time* is not flagged.

### See Also

- hisl\_0024: Inport interface definition

## Check for root Inports with missing range definitions

**Check ID:** mathworks.hism.hisl\_0025

Identify root level Inport blocks with missing or erroneous minimum or maximum range values.

### Description

The check identifies root level Inport blocks with missing or erroneous minimum or maximum range values. You can specify Inport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Inport blocks and signal objects that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<b>Missing range</b> — Model contains Inport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Inport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.
<b>Missing range(s) for bus object</b> — Bus objects defining the Inport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters .
<b>Range specified will be ignored</b> — Minimum or maximum values at Inports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.
<b>No data type specified</b> — Model contains Inport blocks or Simulink signal objects with inherited data types.	Specify one of the supported data types: <ul style="list-style-type: none"> <li>• Enum</li> <li>• Simulink.AliasType</li> <li>• Simulink.Bus</li> <li>• Simulink.NumericType</li> <li>• build-in</li> </ul>
<b>Implicit resolution to a Simulink signal object</b> — Model contains Inport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.	For the listed Simulink signal objects, in the property dialog, select signal property <b>Signal name must resolve to Simulink signal object</b> . To set this option programmatically, use the port parameter <b>MustResolveToSignalObject</b> .

### Capabilities and Limitations

- Does not run on library models.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0025: Design min/max specification of input interfaces

## Check for root Outports with missing range definitions

**Check ID:** `mathworks.hism.hisl_0026`

Identify root level Outport blocks with missing or erroneous minimum or maximum range values.

### Description

The check identifies root level Outport blocks with missing or erroneous minimum or maximum range values. You can specify Outport block minimum and maximum values with block parameters or Simulink signal objects that explicitly resolve to the connected signal lines. A results table provides links to Outport blocks that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<b>Missing range</b> — Model contains Outport blocks with numeric data types that have missing range parameters (minimum and/or maximum).	For the listed Outport blocks and Simulink signal objects, specify scalar minimum and maximum parameters.
<b>Missing range(s) for bus object</b> — Bus objects defining the Outport blocks have leaf elements with missing ranges.	For the listed leaf elements, to specify the model interface range, provide scalar minimum and maximum parameters.
<b>Range specified at Outport will be ignored</b> — Minimum or maximum values at Outports or Simulink signal objects are not supported for bus data types. The values are ignored during range checking.	To enable range checking, specify minimum and maximum signal values on the leaf elements of the bus objects defining the data type.

Condition	Recommended Action
<p><b>No bus data type specified</b> — Model contains Outport block or Simulink signal objects with inherited bus data types.</p>	<p>For the Outport blocks and Simulink signal objects, specify one of the supported data types:</p> <ul style="list-style-type: none"> <li>• Enum</li> <li>• Simulink.AliasType</li> <li>• Simulink.Bus</li> <li>• Simulink.NumericType</li> <li>• built-in</li> </ul>
<p><b>Implicit resolution to a Simulink signal object</b> — Model contains Outport block signal names that implicitly resolve to a Simulink signal object in the base workspace, model workspace, or Simulink data dictionary.</p>	<p>For the listed Simulink signal objects, in the property dialog, select signal property <b>Signal name must resolve to Simulink signal object</b>. To set this option programmatically, use the port parameter <b>MustResolveToSignalObject</b>.</p>

**Capabilities and Limitations**

- Does not run on library models.
- Allows exclusions of blocks and charts.

**See Also**

- hisl\_0026: Design min/max specification of output interfaces

**Check safety-related diagnostic settings for data store memory**

**Check ID:** `mathworks.hism.hisl_0013`

Check model configuration for diagnostic settings that apply to data store memory and that can impact safety.

**Description**

This check verifies that model diagnostic configuration parameters pertaining to data store memory are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether the model attempts to read data from a data store in which it has not stored data in the current time step is set to a value other than <b>Enable all as errors</b> . Reading data before it is written can result in use of stale data or data that is not initialized.	Set <b>Detect read before write</b> (Simulink) in the Configuration Parameters dialog box or set the parameter <code>ReadBeforeWriteMsg</code> to <b>Enable all as errors</b> .
The diagnostic that detects whether the model attempts to store data in a data store, after previously reading data from it in the current time step, is set to a value other than <b>Enable all as errors</b> . Writing data after it is read can result in use of stale or incorrect data.	Set <b>Detect write after read</b> (Simulink) in the Configuration Parameters dialog box or set the parameter <code>WriteAfterReadMsg</code> to <b>Enable all as errors</b> .
The diagnostic that detects whether the model attempts to store data in a data store twice in succession in the current time step is set to a value other than <b>Enable all as errors</b> . Writing data twice in one time step can result in unpredictable data.	Set <b>Detect write after write</b> (Simulink) in the Configuration Parameters dialog box or set the parameter <code>WriteAfterWriteMsg</code> to <b>Enable all as errors</b> .
The diagnostic that detects when one task reads data from a Data Store Memory block to which another task writes data is set to <b>none</b> or <b>warning</b> . Reading or writing data in different tasks in multitask mode can result in corrupted or unpredictable data.	Set <b>Multitask data store</b> (Simulink) in the Configuration Parameters dialog box or set the parameter <code>MultiTaskDSMsg</code> to <b>error</b> .
The diagnostic detects that the parameter <b>Duplicate data store names</b> is not set to <b>error</b> .	Set <b>Duplicate data store names</b> in the Configuration Parameters dialog box or set the parameter <code>UniqueDataStoreMsg</code> to <b>error</b> .

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data store memory and that can impact safety.

### Capabilities and Limitations

- This check does not check the data store reads and writes that occur across model and atomic subsystem boundaries.
- This check does not check the usage of data stores to write and read data at different rates.
- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0013: Usage of data store blocks

## Check safety-related diagnostic settings for data used for debugging

**Check ID:** `mathworks.hism.hisl_0305`

Check model configuration for diagnostic settings that apply to data used for debugging and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to debugging are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The diagnostic that enables model verification blocks is set to <code>Use local settings</code> or <code>Enable all</code> . Such blocks should be disabled because they are assertion blocks, which are for verification only. Model developers should not use assertions in embedded code.	In the Configuration Parameters dialog box, set <b>Model Verification block enabling</b> (Simulink) or set parameter <code>AssertControl</code> to <code>Disable All</code> .

## Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to data used for debugging and that can impact safety.

## Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

## See Also

- “hisl\_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging” (Simulink)

## Check safety-related diagnostic settings for parameters

**Check ID:** `mathworks.hism.hisl_0302`

Check model configuration for diagnostic settings that apply to parameters and that can impact safety.

## Description

This check verifies that model diagnostic configuration parameters pertaining to parameters are set optimally for generating code for a safety-related application.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a parameter downcast occurs is set to <code>none</code> or <code>warning</code> . A downcast to a lower signal range can result in numeric overflows of parameters, resulting in unexpected behavior.	Set <b>Detect downcast</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>ParameterDowncastMsg</code> to <code>error</code> .

Condition	Recommended Action
<p>The diagnostic that detects when a parameter underflow occurs is set to none or warning. When the data type of a parameter does not have enough resolution, the parameter value is zero instead of the specified value. This can lead to incorrect operation of generated code.</p>	<p>Set <b>Detect underflow</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>ParameterUnderflowMsg</code> to error.</p>
<p>The diagnostic that detects when a parameter overflow occurs is set to none or warning. Numeric overflows can result in unexpected application behavior and should be detected and fixed in safety-related applications.</p>	<p>Set <b>Detect overflow</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>ParameterOverflowMsg</code> to error.</p>
<p>The diagnostic that detects when a parameter loses precision is set to none or warning. Not detecting such errors can result in a parameter being set to an incorrect value in the generated code.</p>	<p>Set <b>Detect precision loss</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>ParameterPrecisionLossMsg</code> to error.</p>
<p>The diagnostic that detects when an expression with tunable variables is reduced to its numerical equivalent is set to none or warning. This can result in a tunable parameter unexpectedly not being tunable in generated code.</p>	<p>Set <b>Detect loss of tunability</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>ParameterTunabilityLossMsg</code> to error.</p>

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to parameters and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters



## Check safety-related solver settings for solver options

**Check ID:** `mathworks.hism.hisl_0041`

Check solver settings in the model configuration that apply to solvers and might impact safety.

### Description

This check verifies that the model solver configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The solver setting to specify the type of solver to simulate model is set to <code>Variable-step</code> .	In the Configuration Parameters dialog box, set "Type" (Simulink) or set the parameter <code>SolverType</code> to <code>Fixed-step</code> .
The solver setting to specify the solver to compute the states of the model during simulation or code generation is set to a value other than <code>Discrete(no continuous states)</code> .	In the Configuration Parameters dialog box, set "Solver" (Simulink) to <code>discrete(no continuous states)</code> or set the parameter <code>Solver</code> to <code>FixedStepDiscrete</code> .

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

### See Also

- `hisl_0041`: Configuration Parameters > Solver > Solver options

## Check safety-related solver settings for tasking and sample-time

**Check ID:** `mathworks.hism.hisl_0042`

Check solver settings in the model configuration that apply to periodic sample time constraints and might impact safety.

### Description

This check verifies that model configuration parameters are set optimally to ensure that the model operates at a specific set of prioritized periodic sample times for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Configuration parameter <b>Automatically handle rate transition for data transfer</b> is selected.	Clear <b>Automatically handle rate transition for data transfer</b> in the Configuration Parameters dialog box or set parameter <code>AutoInsertRateTranBlk</code> to off.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- `hisl_0042`: Configuration Parameters > Solver > Tasking and sample time options

## Check usage of shift operations for Stateflow data

**Check ID:** `mathworks.hism.hisf_0064`

Identify usage of shift operations for Stateflow data that might impact safety.

## Description

This check inspects the shift operations that have shift operand values greater than the bit-width of the input or output type or a shift operand that has a negative value.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Right-shift operations are greater than the bit-width of the input type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.
Left-shift operations are greater than the bit-width of the output type.	Explicitly modify the value of the bit-shift operations to be less than the shift operand.

## Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Does not support the shift operation that has the shift size defined as a Simulink signal or a variable.
- Does not support the shift operations that consist of shift size decided at run time.
- External functions written in C and embedded in Stateflow will not work with this check.

## See Also

- hisf\_0064: Shift operations for Stateflow data to improve code compliance

## Check usage of Signal Routing blocks

**Check ID:** mathworks.hism.hisl\_0034

Identify usage of Signal Routing blocks that might impact safety.

**Description**

This check identifies model or subsystem Switch blocks that might generate code with inequality operations (~=) in expressions that contain a floating-point variable or constant.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The model or subsystem contains a Switch block that might generate code with inequality operations (~=) in expressions where at least one side of the expression contains a floating-point variable or constant. The Switch block might cause floating-point inequality comparisons in the generated code.	For the identified block, do one of the following: <ul style="list-style-type: none"> <li>• For the control input block, change the <b>Data type</b> parameter setting.</li> <li>• Change the Switch block <b>Criteria for passing first input</b> parameter setting. This might change the algorithm.</li> </ul>

**Capabilities and Limitations**

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- hisl\_0034: Usage of Signal Routing blocks

**Check state machine type of Stateflow charts**

**Check ID:** mathworks.hism.hisf\_0001

Identify Stateflow Charts whose State Machine Type differs from the type set in the Model Advisor Configuration Editor.

## Description

Compares the state machine type of all Stateflow charts to the type that you specify in the input parameters.

Available with Simulink Check.

This check requires a Stateflow license.

## Input Parameters

### Classic

Check whether all charts are Classic charts.

### Mealy

Check whether all charts are Mealy charts.

### Moore

Check whether all charts are Moore charts.

## Results and Recommended Actions

Condition	Recommended Action
The input parameter is set to <b>Classic</b> and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Classic</b> .
The input parameter is set to <b>Moore</b> and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Moore</b> .
The input parameter is set to <b>Mealy</b> and charts in the model use other state machine types.	For each chart, in the Chart Properties dialog box, specify <b>State Machine Type</b> to <b>Mealy</b> .

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to **on**.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to **all**.

- Allows exclusions of blocks and charts.

### See Also

- hisf\_0001: State Machine Type

## Check Stateflow charts for ordering of states and transitions

**Check ID:** `mathworks.hism.hisf_0002`

Identify Stateflow charts that have **User specified state/transition execution order** cleared.

### Description

Identify Stateflow charts that have **User specified state/transition execution order** cleared, and therefore do not use explicit ordering of parallel states and transitions.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Stateflow charts have <b>User specified state/transition execution order</b> cleared.	For the specified charts, in the Chart Properties dialog box, select <b>User specified state/transition execution order</b> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Allows exclusions of blocks and charts.

**Action Results**

Clicking **Modify** selects **User specified state/transition execution order** for the specified charts.

**See Also**

- hisf\_0002: User-specified state/transition execution order

**Check Stateflow charts for strong data typing**

**Check ID:** mathworks.hism.hisf\_0015

Identify variables and parameters in expressions with different data types in Stateflow objects.

**Description**

To facilitate strong data typing, this check identifies the variables and parameters in expressions with different data types in Stateflow states and transitions.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
The Stateflow objects have variables and parameters in expressions with different data types.	Explicitly cast variables and parameters in expressions to the same data types. For more information see, <code>cast</code> .

**Capabilities and Limitations**

- Does not run on library models.
- Allows exclusions of blocks and charts.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to `on`.

- Does not analyze the type of literals in expressions in Stateflow objects. Explicitly casts types of literals to the intended data type.
- Does not flag expressions with true and false keywords. For more information, see “Reserved Keywords for Code Generation” (Embedded Coder).
- External functions written in C and embedded in Stateflow will not work with this check.

### See Also

- hisf\_0015: Strong data typing (casting variables and parameters in expressions)

## Check Stateflow charts for unary operators

**Check ID:** `mathworks.hism.hisf_0211`

Identify unary operators in Stateflow charts.

### Description

This check identifies the unary minus operators on unsigned data types in Stateflow charts.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart consists of a unary minus operator on unsigned data types.	Explicitly modify the unary operator on unsigned data types. For more information, see “Unary Operations and Actions” (Stateflow).

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.



- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Does not flag expressions with bitwise and arithmetic operators. For example, - (u1/u2) is not flagged.

### See Also

- hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

## Check Stateflow charts for uniquely defined data objects

**Check ID:** mathworks.hism.hisl\_0061

Identify Stateflow charts that include data objects that are not uniquely defined.

### Description

This check searches your model for local data in Stateflow charts that is not uniquely defined.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The Stateflow chart contains a data object identifier defined in two or more scopes.	For the identified chart, do one of the following: <ul style="list-style-type: none"> <li>• Create a unique data object identifier within each of the scopes.</li> <li>• Create a unique data object identifier within the chart, at the parent level.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.

- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### **See Also**

- `hisl_0061`: Unique identifiers for clarity

## **Check Stateflow debugging options**

**Check ID:** `mathworks.hism.hisf_0011`

Check the Stateflow debugging settings.

### **Description**

Verify the following debugging settings.

- **Wrap on overflow**
- **Simulation range checking**
- **Detect Cycles**
- **Underspecified**
- **Overspecified**

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
<p>Any of the following:</p> <ul style="list-style-type: none"> <li>• <b>Wrap on overflow</b> is not set to error.</li> <li>• <b>Simulation range checking</b> is not set to error.</li> <li>• <b>Detect Cycles</b> is cleared.</li> </ul>	<p>In the Configuration Parameters dialog box, set:</p> <ul style="list-style-type: none"> <li>• <b>Wrap on overflow</b> to error, or set the parameter <b>IntegerOverflowMsg</b> to error.</li> <li>• <b>Simulation range checking</b> to error, or set the parameter <b>SignalRangeChecking</b> to error.</li> </ul> <p>In the model window, select:</p> <ul style="list-style-type: none"> <li>• <b>Debug &gt; Diagnostics &gt; Detect Cyclical Behavior</b></li> </ul>

## Capabilities and Limitations

- Truth tables are not analyzed in this check.
- Does not run on library models.
- Does not analyze content of library linked blocks.
- Allows exclusions of blocks and charts.

## Action Results

Clicking **Modify** selects the specified debugging options.

## See Also

- hisf\_0011: Stateflow debugging settings

## Check usage of lookup table blocks

**Check ID:** mathworks.hism.hisl\_0033

Check for lookup table blocks that do not generate out-of-range checking code.

### Description

This check verifies that the following blocks generate code to protect against inputs that fall outside the range of valid breakpoint values:

- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table
- Prelookup

This check also verifies that Interpolation Using Prelookup blocks generate code to protect against inputs that fall outside the range of valid index values.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The lookup table block does not generate out-of-range checking code.	Change the setting on the block dialog box so that out-of-range checking code is generated. <ul style="list-style-type: none"> <li>• For the 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, and Prelookup blocks, clear the check box for <b>Remove protection against out-of-range input in generated code.</b></li> <li>• For the Interpolation Using Prelookup block, clear the check box for <b>Remove protection against out-of-range index in generated code.</b></li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**Action Results**

Clicking **Modify** verifies that lookup table blocks are set to generate out-of-range checking code.

**See Also**

- hisl\_0033: Usage of Lookup Table blocks

**Check for variant blocks with 'Generate preprocessor conditionals' active**

**Check ID:** mathworks.hism.hisl\_0023

Check variant block parameters for settings that might result in code that does not trace to requirements.

**Description**

This check verifies that variant block parameters for code generation are set to trace to requirements.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The option to generate preprocessor conditionals is selected in one or more variant blocks in the model.	In order to simplify the tracing of code to requirements, consider clearing the option to generate preprocessor conditionals in variant blocks.

**Capabilities and Limitations**

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems.
- Does not allow exclusions of blocks or charts.

**See Also**

- hisl\_0023: Verification of model and subsystem variants

**Check safety-related diagnostic settings for signal connectivity**

**Check ID:** `mathworks.hism.hisl_0306`

Check model configuration for diagnostic settings that apply to signal connectivity and that can impact safety.

**Description**

This check verifies that model diagnostic configuration parameters pertaining to signal connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The diagnostic that detects virtual signals that have a common source signal but different labels is set to <code>none</code> or <code>warning</code> . This diagnostic pertains to virtual signals only and has no effect on generated code. However, signal label mismatches can lead to confusion during model reviews.	Set <b>Signal label mismatch</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>SignalLabelMismatchMsg</code> to <code>error</code> .
The diagnostic that detects when the model contains a block with an unconnected input signal is set to <code>none</code> or <code>warning</code> . This must be detected because code is not generated for unconnected block inputs.	Set <b>Unconnected block input ports</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>UnconnectedInputMsg</code> to <code>error</code> .
The diagnostic that detects when the model contains a block with an unconnected output signal is set to <code>none</code> or <code>warning</code> . This must be detected because dead code can result from unconnected block output signals.	Set <b>Unconnected block output ports</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>UnconnectedOutputMsg</code> to <code>error</code> .

Condition	Recommended Action
The diagnostic that detects unconnected signal lines and unmatched Goto or From blocks is set to none or warning. This error must be detected because code is not generated for unconnected lines.	Set <b>Unconnected line</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter UnconnectedLineMsg to error.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal connectivity and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals

## Check safety-related diagnostic settings for bus connectivity

**Check ID:** mathworks.hism.hisl\_0307

Check model configuration for diagnostic settings that apply to bus connectivity and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to bus connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic that detects whether a Model block's root Outport block is connected to a bus but does not specify a bus object is set to <b>none</b> or <b>warning</b>. For a bus signal to cross a model boundary, the signal must be defined as a bus object for compatibility with higher level models that use a model as a reference model.</p>	<p>Set <b>Unspecified bus object at root Outport block</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>RootOutportRequireBusObject</code> to error.</p>
<p>The diagnostic that detects whether the name of a bus element matches the name specified by the corresponding bus object is set to <b>none</b> or <b>warning</b>. This diagnostic prevents the use of incompatible buses in a bus-capable block such that the output names are inconsistent.</p>	<p>Set <b>Element name mismatch</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>BusObjectLabelMismatch</code> to error.</p>
<p>The diagnostic that detects when some blocks treat a signal as a mux/vector, while other blocks treat the signal as a bus, is set to <b>none</b> or <b>warning</b>. When the Simulink software automatically converts a muxed signal to a bus, it is possible for an unintended operation or unpredictable behavior to occur.</p>	<p>Set <b>Bus signal treated as vector</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box to error, or the parameter <code>StrictBusMsg to ErrorOnBusTreatedAsVector</code>.</p>
<p>The diagnostic detects that the parameter <b>Non-bus signals treated as bus signals</b> is not set to error.</p>	<p>Set <b>Non-bus signals treated as bus signals</b> on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box, or the parameter <code>NonBusSignalsTreatedAsBus</code> to error.</p>
<p>The diagnostic detects that the parameter <b>Repair bus selections</b> is not set to warn and repair.</p>	<p>Set <b>Repair bus selections</b> on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box, or the parameter <code>BusNameAdapt</code> to warn and repair.</p>

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to bus connectivity and that can impact safety.



### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses

## Check safety-related diagnostic settings that apply to function-call connectivity

**Check ID:** `mathworks.hism.hisl_0308`

Check model configuration for diagnostic settings that apply to function-call connectivity and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to function-call connectivity are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects incorrect use of a function-call subsystem is set to none or warning. If this condition is undetected, incorrect code might be generated.	Set <b>Invalid function-call connection</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>InvalidFcnCallConnMsg</code> to error.
The diagnostic that specifies whether the Simulink software has to compute inputs of a function-call subsystem directly or indirectly while executing the subsystem is set to Use local settings or Disable all. This diagnostic detects unpredictable data coupling between a function-call subsystem and the inputs of the subsystem in the generated code.	Set <b>Context-dependent inputs</b> (Simulink) on the <b>Diagnostics &gt; Connectivity</b> pane in the Configuration Parameters dialog box or set the parameter <code>FcnCallInpInsideContextMsg</code> to error.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to function-call connectivity and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls

## Check safety-related diagnostic settings for compatibility

**Check ID:** `mathworks.hism.hisl_0301`

Check model configuration for diagnostic settings that affect compatibility and that might impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to compatibility are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects when a block has not been upgraded to use features of the current release is set to <code>none</code> or <code>warning</code> . An S-function written for an earlier version might not be compatible with the current version and generated code could operate incorrectly.	Set <b>S-function upgrades needed</b> (Simulink) on the <b>Diagnostics &gt; Compatibility</b> pane in the Configuration Parameters dialog box or set the parameter <code>SFcnCompatibilityMsg</code> to <code>error</code> .

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that affect compatibility and that might impact safety.

**Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

**See Also**

- hisl\_0301: Configuration Parameters > Diagnostics > Compatibility

**Check safety-related diagnostic settings for model initialization**

**Check ID:** `mathworks.hism.hisl_0304`

In the model configuration, check diagnostic settings that affect model initialization and might impact safety.

**Description**

This check verifies that model diagnostic configuration parameters for initialization are optimally set to generate code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to <b>Classic</b>, ensuring compatibility with previous releases of Simulink. The “Check undefined subsystem initial output” (Simulink) diagnostic is cleared. This diagnostic specifies whether Simulink displays a warning if the model contains a conditionally executed subsystem, in which a block with a specified initial condition drives an Output block with an undefined initial condition. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.</p>	<p>Do one of the following:</p> <ul style="list-style-type: none"> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Simplified</b>.</li> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Classic</b> and select <b>Check undefined subsystem initial output</b> (Simulink).</li> <li>• Set the parameter <code>CheckSSInitialOutputMsg</code> to on.</li> </ul>
<p>In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to <b>Classic</b>, ensuring compatibility with previous releases of Simulink. This diagnostic detects potential initial output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized. If undetected, this condition can produce behavior that is nondeterministic.</p>	<p>Do one of the following:</p> <ul style="list-style-type: none"> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Simplified</b>.</li> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Classic</b>.</li> <li>• Set the parameter <code>CheckExecutionContextPreStartOutputMsg</code> to on.</li> </ul>

Condition	Recommended Action
<p>In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to <b>Classic</b>, ensuring compatibility with previous releases of Simulink. The “Check runtime output of execution context” (Simulink) diagnostic is cleared. This diagnostic detects potential output differences from earlier releases. A conditionally executed subsystem could have an output that is not initialized and feeds into a block with a tunable parameter. If undetected, this condition can cause the behavior of the downstream block to be nondeterministic.</p>	<p>Do one of the following:</p> <ul style="list-style-type: none"> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Simplified</b>.</li> <li>• In the Configuration Parameters dialog box, set <b>Underspecified initialization detection</b> (Simulink) to <b>Classic</b> and select <b>Check runtime output of execution context</b> (Simulink).</li> <li>• Set the parameter <code>CheckExecutionContextRuntimeOutputMsg</code> to on.</li> </ul>

### Action Results

To configure the diagnostic settings that affect model initialization and might impact safety, click **Modify Settings**.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- “hisl\_0304: Configuration Parameters > Diagnostics > Data Validity > Model initialization” (Simulink)

## Check safety-related diagnostic settings for saving

**Check ID:** `mathworks.hism.hisl_0036`

Check model configuration for diagnostic settings that apply to saving model files

### Description

This check verifies that model configuration parameters are set optimally for saving a model for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains disabled library links before the model is saved is set to <code>none</code> or <code>warning</code> . If this condition is undetected, incorrect code might be generated.	Set <b>Block diagram contains disabled library links</b> (Simulink) in the Configuration Parameters dialog box or set parameter <code>SaveWithDisabledLinkMsg</code> to <code>error</code> .
The diagnostic that detects whether a model contains library links that are using parameters not in a mask before the model is saved is set to <code>none</code> or <code>warning</code> . If this condition is undetected, incorrect code might be generated.	Set <b>Block diagram contains parameterized library links</b> (Simulink) in the Configuration Parameters dialog box or set parameter <code>SaveWithParameterizedLinkMsg</code> to <code>error</code> .

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to saving a model file.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- `hisl_0036`: Configuration Parameters > Diagnostics > Saving

## Check MATLAB Code Analyzer messages

**Check ID:** `mathworks.hism.himl_0004`

Check MATLAB Functions for `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs.

### Description

Verifies `%#codegen` directive, MATLAB Code Analyzer messages, and justification message IDs for:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
For MATLAB code in MATLAB Function blocks, either of the following: <ul style="list-style-type: none"> <li>• Code lines are not justified with a <code>%#ok</code> comment.</li> <li>• Codes lines justified with <code>%#ok</code> do not specify a message id.</li> </ul>	<ul style="list-style-type: none"> <li>• Implement MATLAB Code Analyzer recommendations.</li> <li>• Justify not following MATLAB Code Analyzer recommendations with a <code>%#ok</code> comment.</li> <li>• Specify justified code lines with a message id. For example, <code>%#ok&lt;NOPRT&gt;</code>.</li> </ul>
For MATLAB functions defined in Stateflow charts, either of the following: <ul style="list-style-type: none"> <li>• Code lines are not justified with a <code>%#ok</code> comment.</li> <li>• Codes lines justified with <code>%#ok</code> do not specify a message id.</li> </ul>	<ul style="list-style-type: none"> <li>• Implement MATLAB Code Analyzer recommendations.</li> <li>• Justify not following MATLAB Code Analyzer recommendations with a <code>%#ok</code> comment.</li> <li>• Specify justified code lines with a message id. For example, <code>%#ok&lt;NOPRT&gt;</code>.</li> </ul>

Condition	Recommended Action
<p>For called MATLAB functions:</p> <ul style="list-style-type: none"> <li>• Code does not have the <code>codegen</code> directive.</li> <li>• Code lines are not justified with a <code>ok</code> comment.</li> <li>• Codes lines justified with <code>ok</code> do not specify a message id.</li> </ul>	<ul style="list-style-type: none"> <li>• Insert <code>codegen</code> directive in the MATLAB code.</li> <li>• Implement MATLAB Code Analyzer recommendations.</li> <li>• Justify not following MATLAB Code Analyzer recommendations with a <code>ok</code> comment.</li> <li>• Specify justified code lines with a message id. For example, <code>ok&lt;NOPRT&gt;</code>.</li> </ul>

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- `himl_0004`: MATLAB Code Analyzer recommendations for code generation

## Check safety-related diagnostic settings for Merge blocks

**Check ID:** `mathworks.hism.hisl_0303`

Check model configuration for diagnostic settings that apply to Merge blocks

### Description

This check verifies that model configuration parameters are set optimally for Merge blocks for a safety-related application.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
The diagnostic that detects whether a model contains Merge blocks with more than one driving block executing at the same time step is set to none or warning. In the Configuration Parameters dialog box, the “Underspecified initialization detection” (Simulink) diagnostic is set to Classic.	In the Configuration Parameters dialog box, set “Detect multiple driving blocks executing at the same time step” (Simulink) or set the parameter MergeDetectMultiDrivingBlocksExec to error.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- “hisl\_0303: Configuration Parameters > Diagnostics > Data Validity > Merge blocks” (Simulink)

## Check safety-related diagnostic settings for Stateflow

**Check ID:** `mathworks.hism.hisl_0311`

Check safety-related diagnostic settings for Stateflow

### Description

This check verifies that model configuration parameters are set optimally for Stateflow for a safety-related application.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The diagnostic that detects whether a chart configuration leads to unwanted backtracking during simulation is set to none or warning.	In the Configuration Parameters dialog box, set “Unexpected backtracking” (Simulink) or set the parameter SFUnexpectedBacktrackingDiag to error.
The diagnostic that detects whether a chart configuration has blocks that connect to chart input ports do not initialize their outputs during initialization is set to none or warning.	In the Configuration Parameters dialog box, set “Invalid input data access in chart initialization” (Simulink) or set the parameter SFInvalidInputDataAccessInChartInitDiag to error.
The diagnostic that detects whether a chart has an unconditional default transition to a state or a junction is set to none or warning.	In the Configuration Parameters dialog box, set “No unconditional default transitions” (Simulink) or set the parameter SFNoUnconditionalDefaultTransitionDiag to error.
The diagnostic that detects whether a chart contains a transition that loops outside of the parent state or junction is set to none or warning.	In the Configuration Parameters dialog box, set “Transition outside natural parent” (Simulink) or set the parameter SFTransitionOutsideNaturalParentDiag to error.
The diagnostic that detects whether a chart is constructed on a valid execution path is set to none or warning.	In the Configuration Parameters dialog box, set “Unreachable execution path” (Simulink) or set the parameter SFUnreachableExecutionPathDiag to error.
The diagnostic detects that the parameter <b>Undirected event broadcasts</b> is not set to none or warning.	Set <b>Undirected event broadcasts</b> in the Configuration Parameters dialog box or set the parameter SFUndirectedBroadcastEventsDiag to error.
The diagnostic detects that the parameter <b>Transition action specified before condition action</b> is not set to none or warning.	Set <b>Transition action specified before condition action</b> in the Configuration Parameters dialog box or set the parameter SFTransitionActionBeforeConditionDiag to error.

**Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

**Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

**See Also**

- hisl\_0311: Configuration Parameters > Diagnostics > Stateflow

**Check for Strong Data Typing with Simulink I/O**

**Check ID:** `mathworks.hism.hisf_0009`

Identify usage of Stateflow constructs that might impact safety.

**Description**

This check identifies instances of Stateflow software being used in a way that can impact an application's safety by using strong data typing.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
A Stateflow chart is not configured for strong data typing on boundaries between a Simulink model and the Stateflow chart.	In the Chart properties dialog box, select <b>Use Strong Data Typing with Simulink I/O</b> for the Stateflow chart. When you select this check box, the Stateflow chart accepts input signals of any data type that Simulink models support, provided that the type of the input signal matches the type of the corresponding Stateflow input data object.

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts. Exclusions will not work for library linked charts.

## See Also

- hisf\_0009: Strong data typing (Simulink and Stateflow boundary)

## Check for MATLAB Function interfaces with inherited properties

**Check ID:** `mathworks.hism.himl_0002`

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

### Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	<p>Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Functions identified in the results.</p> <p>If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink):</p> <ul style="list-style-type: none"> <li>• Change <b>Complexity</b> from Inherited to On or Off.</li> <li>• Change <b>Type</b> from Inherit: Same as Simulink to an explicit type.</li> </ul>

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- himl\_0002: Strong data typing at MATLAB function boundaries

### Check usage of Abs blocks

**Check ID:** mathworks.hism.hisl\_0001

Identify usage of Math Operation blocks that might impact safety.

### Description

This check inspects the usage of the Abs block.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains an Absolute Value block that is operating on one of the following:</p> <ul style="list-style-type: none"> <li>• A boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code</li> <li>• A signed integer value with the <b>Saturate on integer overflow</b> check box not selected. For signed data types, the absolute value of the most negative value is problematic because it is not representable by the data type. This condition results in an overflow in the generated code.</li> </ul>	<p>If the identified Absolute Value block is operating on a boolean or unsigned data type, do one of the following:</p> <ul style="list-style-type: none"> <li>• Change the input of the Absolute Value block to a signed input type.</li> <li>• Remove the Absolute Value block from the model.</li> </ul> <p>If the identified Absolute Value block is operating on a signed data type, in the <b>Block Parameters &gt; Signal Attributes</b> dialog box, select <b>Saturate on integer overflow</b>.</p>

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0001: Usage of Abs block

## Check usage of Math Function blocks (rem and reciprocal functions)

**Check ID:** mathworks.hism.hisl\_0002

Identify usage of Math Operation blocks that might impact safety.

### Description

This check inspects the usage of the Math Function blocks that have remainder after division and reciprocals.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains Math Function - Remainder after division (rem) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a rem function, protect the second input to the block from being equal to zero. Otherwise the output can produce a Inf or -Inf and result in a run-time error in the generated code.
The model or subsystem contains Math Function - Reciprocal (reciprocal) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a reciprocal function, protect the input to the block from being equal to zero. Otherwise the output can produce a Inf or -Inf and result in a run-time error in the generated code.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

**See Also**

- hisl\_0002: Usage of Math Function blocks (rem and reciprocal)

**Check usage of Math Function blocks (log and log10 functions)**

**Check ID:** `mathworks.hism.hisl_0004`

Identify usage of Math Operation blocks that might impact safety.

**Description**

This check inspects the usage of the Math Function blocks that have Natural logarithm and Common (base 10) logarithms.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The model or subsystem contains Math Function - Natural logarithm ( <code>log</code> ) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a <code>log</code> function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a NaN or <code>-Inf</code> and result in a run-time error in the generated code.
The model or subsystem contains Math Function - Common (base 10) ( <code>base 10 logarithm</code> ) blocks that might result in non-finite output signals. Non-finite signals are not supported in real-time embedded systems.	When using the Math Function block with a <code>log10</code> function, protect the input to the block from being less than or equal to zero. Otherwise, the output can produce a NaN or <code>-Inf</code> and result in a run-time error in the generated code.

**Capabilities and Limitations**

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.



- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)

## Check usage of Assignment blocks

**Check ID:** mathworks.hism.hisl\_0029

Identify usage of Math Operation blocks that might impact safety.

### Description

This check inspects the usage of the Assignment blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem might contain Assignment blocks with incomplete array initialization (not in the iterator subsystem) that do not have block parameter <b>Action if any output element is not assigned</b> set to <b>Error</b> or <b>Warning</b> .	Set block parameter <b>Action if any output element is not assigned</b> to one of the recommended values: <ul style="list-style-type: none"> <li>• <b>Error</b></li> <li>• <b>Warning</b></li> </ul>
The model or subsystem might contain Assignment blocks in the iterator subsystem and the parameter <b>Action if any output element is not assigned</b> is not set to <b>Error</b> .	Set block parameter <b>Action if any output element is not assigned</b> to <b>Error</b> .

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- hisl\_0029: Usage of Assignment blocks

**Check safety-related optimization settings for data type conversions**

**Check ID:** `mathworks.hism.hisl_0053`

Check model configuration for optimization settings that can impact safety.

**Description**

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The optimization that suppresses generation of code resulting from floating-point to integer conversions that wrap out-of-range values is cleared. You must avoid overflows for safety-related code. When this optimization is off and your model includes blocks that disable the <b>Saturate on overflow</b> parameter, the code generator wraps out-of-range values for those blocks. This can result in unreachable and, therefore, untestable code.	If you have a Simulink Coder license, select Configuration Parameter <b>Remove code from floating-point to integer conversions that wraps out-of-range values</b> (Simulink Coder) or set the parameter <code>EfficientFloat2IntCast</code> to on.

**Action Results**

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### **Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### **See Also**

- hisl\_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values

## **Check safety-related optimization settings for data initialization**

**Check ID:** `mathworks.hism.hisl_0052`

Check model configuration for optimization settings that can impact safety.

### **Description**

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The optimization that suppresses the generation of initialization code for root-level inports and outports that are set to zero is selected. For safety-related code, you should explicitly initialize all variables.</p>	<p>If you have an Embedded Coder® license and are using an ERT-based system target file, clear Configuration Parameter <b>Remove root level I/O zero initialization</b> (Simulink Coder) or set the parameter <code>ZeroExternalMemoryAtStartup</code> to on. Alternatively, confirm that your compiler is standards-compliant, as most modern C compilers initialize global data to zero. Or, integrate external, handwritten code that initializes all I/O variables to zero explicitly.</p>
<p>The optimization that suppresses the generation of initialization code for internal work structures, such as block states and block outputs that are set to zero, is selected. For safety-related code, you should explicitly initialize every variable.</p>	<p>If you have an Embedded Coder license and are using an ERT-based system target file, clear Configuration Parameter <b>Remove internal data zero initialization</b> (Simulink Coder) or set the parameter <code>ZeroInternalMemoryAtStartup</code> to on. Alternatively, confirm that your compiler is standards-compliant, as most modern C compilers initialize global data to zero. Or, integrate external, handwritten code that initializes every state variable to zero explicitly.</p>

### Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0052: Configuration Parameters > Optimization > Data initialization

- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)

## Check safety-related optimization settings for application lifespan

**Check ID:** `mathworks.hism.hisl_0048`

Check model configuration for optimization settings that can impact safety.

### Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model includes blocks that depend on elapsed or absolute time and is configured to minimize the amount of memory allocated for the timers. Such a configuration limits the number of days the application can execute before a timer overflow occurs. Many aerospace products are powered on continuously and timers should not assume a limited lifespan.	Set Configuration Parameter <b>Application lifespan (days)</b> (Simulink) or set the parameter <code>LifeSpan</code> to <code>inf</code> .

### Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)
- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)

## Check safety-related block reduction optimization settings

**Check ID:** `mathworks.hism.hisl_0046`

Check model configuration for optimization settings that can impact safety.

### Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Block reduction optimization is selected. This optimization can remove blocks from generated code, resulting in requirements without associated code and violations for traceability requirements.	Clear Configuration Parameter <b>Block reduction</b> (Simulink) or set parameter <code>BlockReduction</code> to <code>off</code> .

### Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

**Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

**See Also**

- hisl\_0046: Configuration Parameters > Simulation Target > Block reduction

**Check safety-related optimization settings for logic signals****Check ID:** `mathworks.hism.hisl_0045`

Check model configuration for optimization settings that can impact safety.

**Description**

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
Implementation of logic signals as Boolean data is cleared. Strong data typing is recommended for safety-related code.	Select Configuration Parameter <b>Implement logic signals as boolean data (vs. double)</b> (Simulink) or set the parameter <code>BooleanDataType</code> to on.

**Action Results**Clicking **Modify Settings** configures model optimization settings that can impact safety.Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.**Capabilities and Limitations**

- Does not run on library models.

- Does not allow exclusions of blocks or charts.

**See Also**

- hisl\_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)
- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)

**Check safety-related optimization settings for division arithmetic exceptions**

**Check ID:** `mathworks.hism.hisl_0054`

Check model configuration for optimization settings that can impact safety.

**Description**

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The optimization that suppresses generation of code that guards against division by zero for fixed-point data is selected. You must avoid division-by-zero exceptions in safety-related code.	If you have an Embedded Coder license and are using an ERT-based system target file, clear Configuration Parameter <b>Remove code that protects against division arithmetic exceptions</b> (Simulink Coder) or set the parameter <code>NoFixptDivByZeroProtection</code> to off.

**Action Results**

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.



### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions
- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)

## Check usage of Logical Operator blocks

**Check ID:** mathworks.hism.hisl\_0018

Identify usage of Logical Operator blocks that might impact safety.

### Description

This check inspects the usage of Logical Operator blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains a Logical Operator block that has inputs or outputs that are not Boolean inputs or outputs. The block might result in floating-point equality or inequality comparisons in the generated code.</p>	<ul style="list-style-type: none"> <li>• Modify the Logical Operator block so that all inputs and outputs are Boolean. On the Block Parameters &gt; Signal Attributes pane, consider selecting <b>Require all inputs to have the same data type</b> and setting <b>Output data type</b> to boolean.</li> <li>• In the Configuration Parameters dialog box, consider selecting the <b>Implement logic signals as boolean data (vs. double)</b>.</li> </ul>

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0018: Usage of Logical Operator block

## Check for Relational Operator blocks that equate floating-point types

**Check ID:** `mathworks.hism.hisl_0016`

### Description

This check inspects the usage of:

- Blocks that equate floating point types, including Relational Operator, Compare To Constant, Compare To Zero and, Detect Change blocks.
- Equality operators (`==` and `~=`) in expressions in the if blocks.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues. These operators can lead to unpredictable results in the generated code.	For the identified block, do one of the following: <ul style="list-style-type: none"> <li>• Change the signal data type.</li> <li>• Rework the model to eliminate using == or ~= operators on floating-point signals.</li> </ul>

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## See Also

- hisl\_0016: Usage of blocks that compute relational operators

## Check usage of Relational Operator blocks

**Check ID:** mathworks.hism.hisl\_0017

## Description

This check inspects the usage of blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare To Zeroand, Detect Change blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.	For the identified blocks, use common data types as inputs. You can use Data Type Conversion blocks to change input data types.
The model or subsystem contains a block computing a relational operator that does not have Boolean output. The condition can lead to unpredictable results in the generated code.	For the specified blocks, on the Block Parameters > Signal Attributes pane, set the <b>Output data type</b> to boolean.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0017: Usage of blocks that compute relational operators (2)

## Check usage of Switch Case blocks and Switch Case Action Subsystem blocks

**Check ID:** mathworks.hism.hisl\_0011

### Description

This check inspects the usage of Switch Case blocks

The check flags Switch Case blocks that do not use integer data types or enumeration values for inputs. To comply with “hisl\_0011: Usage of Switch Case blocks and Action

Subsystem blocks” (Simulink) - C, use an integer data type or an enumeration value for the inputs to Switch Case blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains an Switch Case block without a default case.	In the Switch Case block Block Parameters dialog box, select <b>Show default case</b> . Connect the resulting default output port to a Switch Case Action Subsystem block.
The model or subsystem contains a Switch Case block with an output port that does not connect to a Switch Case Action Subsystem block.	Verify that output ports of the Switch Case blocks connect to Switch Case Action Subsystem blocks.
The model or subsystem contains an Switch Case block with non-integer or non-enum input port data types.	Make sure that input data type of the Switch Case blocks is integer or enum.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks

## Check usage of If blocks and If Action Subsystem blocks

**Check ID:** mathworks.hism.hisl\_0010

### Description

This check inspects the usage of If blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains an If block using Elseif expressions without an Else condition.	In the If block Block Parameters dialog box, select <b>Show else condition</b> . Connect the resulting Else output port to an If Action Subsystem block.
The model or subsystem contains an If block with output ports that do not connect to If Action Subsystem blocks.	Verify that output ports of the If block connect to If Action Subsystem blocks.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0010: Usage of If blocks and If Action Subsystem blocks

## Check usage of For Iterator blocks

**Check ID:** mathworks.hism.hisl\_0008

### Description

This check inspects the usage of For Iterator blocks.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.</p>	<p>For the identified For Iterator blocks, do one of the following:</p> <ul style="list-style-type: none"> <li>• Set the <b>Iteration limit source</b> parameter to <code>internal</code>.</li> <li>• If the <b>Iteration limit source</b> parameter must be <code>external</code>, use a Constant, Probe, or Width block as the source.</li> <li>• Clear the <b>Set next i (iteration variable) externally</b> check box.</li> <li>• Consider selecting the <b>Show iteration variable</b> check box and observe the iteration value during simulation.</li> </ul>

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Allows exclusions of blocks and charts.

## See Also

- `hisl_0008`: Usage of For Iterator Blocks

## Check usage of For and While Iterator subsystems

**Check ID:** `mathworks.hism.hisl_0007`

### Description

This check inspects the usage of time-dependent blocks in a For Iterator or While Iterator subsystem.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
<p>The model or subsystem contains one of the following time-dependent blocks in a For Iterator or While Iterator subsystem:</p> <ul style="list-style-type: none"> <li>• Discrete State-Space</li> <li>• Discrete-Time Integrator</li> <li>• Discrete FIR Filter</li> <li>• Discrete Filter</li> <li>• Discrete Transfer Fcn</li> <li>• Discrete Zero-Pole</li> <li>• Transfer Fcn First Order</li> <li>• Transfer Fcn Real Zero</li> <li>• Transfer Fcn Lead or Lag</li> <li>• Discrete Transfer Function with Initial Outputs</li> <li>• Discrete Transfer Function with Initial States</li> <li>• Discrete Zero-Pole with Initial Outputs</li> <li>• Discrete Zero-Pole with Initial States</li> <li>• Discrete Derivative</li> </ul>	<p>In the model or subsystem, consider removing the time-dependent blocks.</p>

**Capabilities and Limitations**

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.



**See Also**

- hisl\_0007: Usage of For Iterator or While Iterator subsystems

**Check usage of While Iterator blocks**

**Check ID:** mathworks.hism.hisl\_0006

**Description**

This check inspects the usage of While Iterator blocks.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
<p>The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code. mo</p>	<p>For the identified While Iterator blocks:</p> <ul style="list-style-type: none"> <li>• Set the <b>Maximum number of iterations (-1 for unlimited)</b> parameter to a positive integer value.</li> <li>• Consider selecting the <b>Show iteration number port</b> check box and observe the iteration value during simulation.</li> </ul>

**Capabilities and Limitations**

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

**See Also**

- hisl\_0006: Usage of While Iterator blocks

## Check safety-related code generation settings for comments

**Check ID:** `mathworks.hism.hisl_0038`

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The option to include comments in the generated code is cleared. Comments provide good traceability between the code and the model.	Select <b>Include comments</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>GenerateComments</code> to on.
The option to include comments that describe the code for blocks is cleared. Comments provide good traceability between the code and the model.	Select "Simulink block comments" (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>SimulinkBlockComments</code> to on.
The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments provide good traceability between the code and the model.	Select <b>Show eliminated blocks</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ShowEliminatedStatement</code> to on.
The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in <code>model_prm.h</code> is cleared. Comments provide good traceability between the code and the model.	Select <b>Verbose comments for 'Model default' storage class</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ForceParamTrailComments</code> to on.

Condition	Recommended Action
The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments provide good traceability between the code and the model.	Select <b>Requirements in block comments</b> (Simulink Coder) on the <b>Code Generation &gt; Custom comments</b> pane in the Configuration Parameters dialog box or set the parameter ReqsInCode to on.

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0038: Configuration Parameters > Code Generation > Comments
- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Identifiers” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Style” (Embedded Coder)

## Check safety-related code generation interface settings

**Check ID:** mathworks.hism.hisl\_0039

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems.	Clear <b>Support: non-finite numbers</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportNonFinite</code> to off.
The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems.	Clear <b>Support: absolute time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportAbsoluteTime</code> to off.
The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems.	Clear <b>Support: continuous time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportContinuousTime</code> to off.
The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems.	Clear <b>Support: non-inlined S-functions</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter <code>SupportNonInlinedSFcns</code> to off.
The option to generate model function calls compatible with the main program module of the pre-R2012a GRT target is selected. This option is inappropriate for real-time safety-related systems.	Clear <b>Classic call call interface</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>GRTInterface</code> to off.
The option to generate the <code>model_update</code> function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code.	Select <b>Single output/update function</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>CombineOutputUpdateFcns</code> to on.
The option to generate the <code>model_terminate</code> function is selected. This function deallocates dynamic memory, which is unsuitable for real-time safety-related systems.	Clear <b>Terminate function</b> (Simulink Coder) on the <b>Code Generation</b> pane in the Configuration Parameters dialog box or set the parameter <code>IncludeMdlTerminateFcn</code> to off.

Condition	Recommended Action
The option to log or monitor error status is cleared. If you do not select this option, the Simulink Coder product generates extra code that might not be reachable for testing.	Select <b>Remove error status field in real-time model data structure</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SuppressErrorStatus</code> to on.
MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses.	Clear <b>MAT-file logging</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter <code>MatFileLogging</code> to off.

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0039: Configuration Parameters > Code Generation > Interface
- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Identifiers” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Style” (Embedded Coder)

## Check safety-related code generation settings for code style

**Check ID:** `mathworks.hism.hisl_0047`

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The option that specifies the style for parenthesis usage is set to <code>Minimum</code> (Rely on C/C++ operators precedence) or to <code>Nominal</code> (Optimize for readability). For safety-related applications, explicitly specify precedence with parentheses.	Set parameter <code>ParenthesesLevel</code> to <code>Maximum</code> (Specify precedence with parentheses).
The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code.	Set parameter <code>PreserveExpressionOrder</code> to <code>on</code> .

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- `hisl_0047`: Configuration Parameters > Code Generation > Code Style
- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Identifiers” (Simulink Coder)

- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Style” (Embedded Coder)

## Check safety-related code generation identifier settings

**Check ID:** `mathworks.hism.hisl_0049`

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews.	Set <b>Minimum mangle length</b> (Simulink Coder) on the <b>Code Generation &gt; Identifier</b> pane in the Configuration Parameters dialog box or the parameter <code>MangleLength</code> to a value of 4 or greater.

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.

- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0049: Configuration Parameters > Code Generation > Identifiers
- “Model Configuration Parameters: Comments” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Identifiers” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Interface” (Simulink Coder)
- “Model Configuration Parameters: Code Style” (Embedded Coder)

## Check safety-related diagnostic settings for model referencing

**Check ID:** `mathworks.hism.hisl_0310`

Check model configuration for diagnostic settings that apply to model referencing and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic that detects a mismatch between the version of the model that creates or refreshes a Model block and the current version of the referenced model is set to error or warning. The detection occurs during load and update operations. When you get the latest version of the referenced model from the software configuration management system, rather than an older version that was used in a previous simulation, if this diagnostic is set to error, the simulation is aborted. If the diagnostic is set to warning, a warning message is issued. To resolve the issue, the user must resave the model being simulated, which may not be the desired action.</p>	<p>Set <b>Model block version mismatch</b> (Simulink) on the <b>Diagnostics &gt; Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>ModelReferenceVersionMismatchMessage</code> to none.</p>
<p>The diagnostic that detects port and parameter mismatches during model loading and updating is set to none or warning. If undetected, such mismatches can lead to incorrect simulation results because the parent and referenced models have different interfaces.</p>	<p>Set <b>Port and parameter mismatch</b> (Simulink) on the <b>Diagnostics &gt; Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>ModelReferenceIOMismatchMessage</code> to error.</p>
<p>The diagnostic that detects invalid internal connections to the current model's root-level Inport and Outport blocks is set to none or warning. When this condition is detected, the Simulink software might automatically insert hidden blocks into the model to fix the condition. The hidden blocks can result in generated code without traceable requirements. Setting the diagnostic to error forces model developers to fix the referenced models manually.</p>	<p>Set <b>Invalid root Inport/Outport block connection</b> (Simulink) on the <b>Diagnostics &gt; Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>ModelReferenceIOMessage</code> to error.</p>

Condition	Recommended Action
<p>The diagnostic that detects whether To Workspace or Scope blocks are logging data in a referenced model is set to none or warning. Data logging is not supported for To Workspace and Scope blocks in referenced models.</p>	<p>Set <b>Unsupported data logging</b> (Simulink) on the <b>Diagnostics &gt; Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>ModelReferenceDataLoggingMessage</code> to error. To log data, remove the blocks and log the referenced model signals. For more information, see "Logging Referenced Model Signals" (Simulink).</p>

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to model referencing and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing

## Check safety-related diagnostic settings for sample time

**Check ID:** `mathworks.hism.hisl_0044`

Check model configuration for diagnostic settings that apply to sample time and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to sample times are set optimally for generating code for a safety-related application.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic for detecting when a source block, such as a Sine Wave block, inherits a sample time (specified as -1) is set to <b>none</b> or <b>warning</b>. The use of inherited sample times for a source block can result in unpredictable execution rates for the source block and blocks connected to it. For safety-related applications, source blocks should have explicit sample times to prevent incorrect execution sequencing.</p>	<p>Set <b>Source block specifies -1 sample time</b> (Simulink) on the <b>Diagnostics &gt; Sample Time</b> pane in the Configuration Parameters dialog box or set the parameter <code>InheritedTsInSrcMsg</code> to error.</p>
<p>The diagnostic for detecting invalid rate transitions between two blocks operating in multitasking mode is set to <b>none</b> or <b>warning</b>. Such rate transitions should not be used for embedded real-time code.</p>	<p>Set <b>Multitask rate transition</b> (Simulink) on the <b>Diagnostics &gt; Sample Time</b> pane in the Configuration Parameters dialog box or set the parameter <code>MultiTaskRateTransMsg</code> to error.</p>
<p>The diagnostic for detecting subsystems that can cause data corruption or nondeterministic behavior is set to <b>none</b> or <b>warning</b>. This diagnostic detects whether conditionally executed multirate subsystems (enabled, triggered, or function-call subsystems) operate in multitasking mode. Such subsystems can corrupt data and behave unpredictably in real-time environments that allow preemption.</p>	<p>Set <b>Multitask conditionally executed subsystem</b> (Simulink) on the <b>Diagnostics &gt; Sample Time</b> pane in the Configuration Parameters dialog box or set the parameter <code>MultiTaskCondExecSysMsg</code> to error.</p>
<p>The diagnostic for checking sample time consistency between a Signal Specification block and the connected destination block is set to <b>none</b> or <b>warning</b>. An over-specified sample time can result in an unpredictable execution rate.</p>	<p>Set <b>Enforce sample times specified by Signal Specification blocks</b> (Simulink) on the <b>Diagnostics &gt; Sample Time</b> pane in the Configuration Parameters dialog box or set the parameter <code>SigSpecEnsureSampleTimeMsg</code> to error.</p>
<p>The diagnostic detects that the parameter <b>Single task rate transition</b> is not set to error.</p>	<p>Set <b>Single task rate transition</b> in the Configuration Parameters dialog box or set the parameter <code>SingleTaskRateTransMsg</code> to error.</p>

Condition	Recommended Action
The diagnostic detects that the parameter <b>Tasks with equal priority</b> is not set to error.	Set <b>Tasks with equal priority</b> in the Configuration Parameters dialog box or set the parameter <code>TasksWithSamePriorityMsg</code> to error.
The diagnostic for detecting whether a model contains an S-function that has not been specified explicitly to inherit sample time is set to none or warning. These settings can result in unpredictable behavior. A model developer needs to know when such an S-function exists in a model so it can be modified to produce predictable behavior.	Set <b>Unspecified inheritability of sample time</b> (Simulink) in the Configuration Parameters dialog box or set parameter <code>UnknownTsInhSupMsg</code> to error.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to sample time and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- `hisl_0044`: Configuration Parameters > Diagnostics > Sample Time

## Check MATLAB Function metrics

**Check ID:** `mathworks.hism.himl_0003`

Display complexity and code metrics for MATLAB Functions. Report metric violations.

### Description

This check provides complexity and code metrics for MATLAB Functions. The check reports two categories of metrics violations:

- Block-based metrics — Metrics for the overall code of the MATLAB Function block

- Function-based metrics — Metrics for each function of the block, presented individually

Available with Simulink Check.

This check requires a Stateflow license.

### Input Parameters

You can define the tolerance for these complexity parameters by using the Model Advisor Configuration Editor:

- **Maximum effective lines of code per function** — Effective lines do not include empty lines, comment lines, or lines with a function end keyword. The default value is 60.
- **Minimum density of comments** — Density is ratio of comment lines to total lines of code. The default value is 0.2.
- **Maximum cyclomatic complexity per function** — Cyclomatic complexity is the number of linearly independent paths through the source code. The default value is 15.

### Results and Recommended Actions

Condition	Recommended Action
MATLAB Function violates the complexity input parameters.	For the MATLAB Function: <ul style="list-style-type: none"> <li>• If effective lines of code is too high, further divide the MATLAB Function.</li> <li>• If comment density is too low, add comment lines.</li> <li>• If cyclomatic complexity per function is too high, further divide the MATLAB Function.</li> </ul>

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Runs on library models.

- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- `himl_0003`: Limitation of MATLAB function complexity

## Check safety-related diagnostic settings for type conversions

**Check ID:** `mathworks.hism.hisl_0309`

Check model configuration for diagnostic settings that apply to type conversions and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to type conversions are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic that detects Data Type Conversion blocks when the type conversion is set to none. The Simulink software might remove unnecessary Data Type Conversion blocks from generated code, which might result in requirements without corresponding code. The removal of these blocks needs to be identified so model developers can explicitly remove the unnecessary blocks .</p>	<p>Set the <b>Unnecessary type conversions</b> (Simulink) Configuration Parameter or <code>UnnecessaryDatatypeConvMsg</code> parameter to warning.</p>

Condition	Recommended Action
The diagnostic that detects vector-to-matrix or matrix-to-vector conversions at block inputs is set to none or warning. When the Simulink software automatically converts between vector and matrix dimensions, unintended operations or unpredictable behavior can occur.	Set the <b>Vector/matrix block input conversion</b> (Simulink) Configuration Parameter or VectorMatrixConversionMsg parameter to error
The diagnostic that detects when a 32-bit integer value is converted to a floating-point value is set to none. This type of conversion can result in a loss of precision due to truncation of the least significant bits for large integer values.	Set the <b>32-bit integer to single precision float conversion</b> (Simulink) Configuration Parameter or Int32ToFloatConvMsg parameter to warning.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to type conversions and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion

## Check safety-related solver settings for simulation time

**Check ID:** mathworks.hism.hisl\_0040

Check solver settings in the model configuration that apply to simulation time and might impact safety.

### Description

This check verifies that the model solver configuration parameters pertaining to simulation time are set optimally for generating code for a safety-related application.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The solver setting to specify the start time for the simulation or generated code is set to a value other than 0.0.	In the Configuration Parameters dialog box, set “Start time” (Simulink) or set the parameter <code>StartTime</code> to 0.0.
The solver setting to specify the stop time for the simulation or generated code is set to a negative value or a positive value greater than the value of “Application lifespan (days)” (Simulink). By default, “Application lifespan (days)” (Simulink) is <code>auto</code> . If you do not change this setting, any positive value for “Stop time” (Simulink) is valid.	In the Configuration Parameters dialog box, set “Stop time” (Simulink) or set the parameter <code>StopTime</code> to a positive value that is less than the value of “Application lifespan (days)” (Simulink).

**Action Results**

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

**Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

**See Also**

- `hisl_0040`: Configuration Parameters > Solver > Simulation time

**Check safety-related diagnostic settings for solvers**

**Check ID:** `mathworks.hism.hisl_0043`

Check model configuration for diagnostic settings that apply to solvers and that can impact safety.



## Description

This check verifies that model diagnostic configuration parameters pertaining to solvers are set optimally for generating code for a safety-related application.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic for detecting automatic breakage of algebraic loops is set to <b>none</b> or <b>warning</b>. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur.</p>	<p>Set <b>Algebraic loop</b> (Simulink) on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box or set the parameter <b>AlgebraicLoopMsg</b> to <b>error</b>. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.</p>
<p>The diagnostic for detecting automatic breakage of algebraic loops for Model blocks, atomic subsystems, and enabled subsystems is set to <b>none</b> or <b>warning</b>. The breaking of algebraic loops can affect the predictability of the order of block execution. For safety-related applications, a model developer needs to know when such breaks occur.</p>	<p>Set <b>Minimize algebraic loop</b> (Simulink) on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box or set the parameter <b>ArtificialAlgebraicLoopMsg</b> to <b>error</b>. Consider breaking such loops explicitly with Unit Delay blocks so that the execution order is predictable. At a minimum, verify that the results of loops breaking automatically are acceptable.</p>
<p>The diagnostic for detecting potential conflict in block execution order is set to <b>none</b> or <b>warning</b>. For safety-related applications, block execution order must be predictable. A model developer needs to know when conflicting block priorities exist.</p>	<p>Set <b>Block priority violation</b> (Simulink) on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box or set the parameter <b>BlockPriorityViolationMsg</b> to <b>error</b>.</p>

Condition	Recommended Action
<p>The diagnostic for detecting whether the Simulink software automatically modifies the solver, step size, or simulation stop time is set to none or warning. Such changes can affect the operation of generated code. For safety-related applications, it is better to detect such changes so a model developer can explicitly set the parameters to known values.</p>	<p>Set <b>Automatic solver parameter selection</b> (Simulink) on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box or set the parameter SolverPrmCheckMsg to error.</p>
<p>The diagnostic for detecting when a name is used for more than one state in the model is set to none. State names within a model should be unique. For safety-related applications, it is better to detect name clashes so a model developer can fix them.</p>	<p>Set <b>State name clash</b> (Simulink) on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box or set the parameter StateNameClashWarn to warning.</p>

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0043: Configuration Parameters > Diagnostics > Solver

## Check safety-related model referencing settings

**Check ID:** `mathworks.hism.hisl_0037`

Check model configuration for model referencing settings that can impact safety.

### Description

This check verifies that model configuration parameters for model referencing are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The referenced model is configured such that its target is rebuilt whenever you update, simulate, or generate code for the model, or if the Simulink software detects changes in known dependencies. These configuration settings can result in unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references.</p>	<p>Set <b>Rebuild</b> (Simulink) on the <b>Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>UpdateModelReferenceTargets</code> to <code>Never</code> or <code>If any changes detected</code>.</p>
<p>The diagnostic that detects whether a target needs to be rebuilt is set to <code>None</code> or <code>Warn</code> if targets require rebuild. For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent. This diagnostic parameter is available only if <b>Rebuild</b> is set to <code>Never</code>.</p>	<p>Set the configuration parameter <b>Never rebuild diagnostics</b> (Simulink) on the <b>Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>CheckModelReferenceTargetMessage</code> to <code>error</code>.</p>
<p>The ability to pass scalar root input by value is off. This capability should be off because scalar values can change during a time step and result in unpredictable data. This parameter is only available when the config parameter <b>Total number of instances allowed per top model</b> is set to <code>One</code> or <code>Multiple</code> (<code>ModelReferenceNumInstancesAllowed</code> is <code>single</code> or <code>multi</code>).</p>	<p>Set <b>Pass fixed-size scalar root inputs by value for code generation</b> (Simulink) on the <b>Model Referencing</b> pane in the Configuration Parameters dialog box or set the parameter <code>ModelReferencePassRootInputsByReference</code> to <code>off</code>.</p>
<p>The model is configured to minimize algebraic loop occurrences. This configuration is incompatible with the recommended setting of <b>Single output/update function</b> for embedded systems code.</p>	<p>In the Configuration Parameters dialog box, set <b>Minimize algebraic loop occurrences</b> (Simulink) or set parameter <code>ModelReferenceMinAlgLoopOccurrences</code> to <code>off</code>.</p>

### Action Results

Clicking **Modify Settings** configures model referencing settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0037: Configuration Parameters > Model Referencing

## Check Stateflow charts for transition paths that cross parallel state boundaries

**Check ID:** `mathworks.hism.hisf_0013`

Identify transition paths that cross parallel state boundaries in Stateflow charts.

### Description

Identify transition paths that cross parallel state boundaries in Stateflow charts. This check identifies transition paths that cross parallel state boundaries in Stateflow charts.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The Stateflow charts have transition paths that cross parallel state boundaries.	Modify the Stateflow charts so that transitions do not cross parallel state boundaries. For more information see, "Transition Between Operating Modes" (Stateflow).

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Allows exclusions of blocks and charts.
- Analyzes content in all masked subsystems.

## See Also

- `hisf_0013`: Usage of transition paths (crossing parallel state boundaries)

## Check assignment operations in Stateflow Charts

**Check ID:** `mathworks.hism.hisf_0065`

Identify assignment operations in Stateflow objects.

### Description

This check identifies the assignment operations in Stateflow objects that implicitly cast integer and fixed-point arithmetic calculations to wider data types than the input data types.

This check identifies only the assignments with arithmetic operations.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The Stateflow object consists of assignment operations that cast integer and fixed-point calculations to wider data types than the input data types.	Explicitly replace assignment operator ( <code>=</code> ) to <code>:=</code> operator in Stateflow objects.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- External functions written in C and embedded in Stateflow will not work with this check.

### See Also

- hisf\_0065: Type cast operations in Stateflow to improve code compliance

## Check usage of bit operation blocks

**Check ID:** `mathworks.hism.hisl_0019`

Identify bit operation blocks with signed data types as inputs

### Description

This check identifies the use of the Bitwise Operations for the input and output data types. The check also looks at other bit operations blocks as following:

- Bit Clear
- Bit Set
- Bitwise operator
- Extract Bits
- Shift Arithmetic

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Bitwise Operations are not used with signed data types.	Use signed data type for Bitwise operations.

### Capabilities and Limitations

- Allows exclusions of blocks and charts.

- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.

### See Also

- `hisl_0019`: Usage of bitwise operations

## Check data types for blocks with index signals

**Check ID:** `mathworks.hism.hisl_0022`

### Description

This check Identifies the blocks with index signals that have data types other than integers or enum and are within the range of indexed values.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more Simulink blocks in the model have index signals that have data types other than integer or enum.	Change the data type of block index signals to an integer or enum data type that covers the range of indexed values.
One or more MATLAB Function blocks have index variables with inappropriate data types.	Change the data type of index variables to an integer or enum data type that covers the range of indexed values.
One or more Stateflow charts in the model have index variables that have data types other than integer or enum.	Change the data type of index signals of the blocks to an integer or enum data type that covers the range of indexed values.

### Capabilities and Limitations

- This check does not support dialog set indices.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

**See Also**

- `hisl_0022`: Data type selection for index signals

**Check model file name**

**Check ID:** `mathworks.hism.hisl_0031`

**Description**

This check inspects the model file name to ensure that the name complies with the recommended guidelines.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
The file name contains illegal characters.	Rename the file. Allowed characters are a-z, A-Z, 0-9, and underscore (_).
The file name starts with a number.	Make sure that the file name does not start with a number.
The file name starts with an underscore ("_").	Make sure that the file name does not start with a underscore ("_").
The file name ends with an underscore ("_").	Make sure that the file name does not end with a underscore ("_").
The file extension contains one or more underscores.	Change the file extension.
The file name has consecutive underscores.	Rename the file to eliminate trailing underscore ` `.
The file name contains more than one dot (".").	Make sure that the file name does not have more than one dot (".").



Condition	Recommended Action
the file name is a C/C++ or MATLAB keyword or built in function	Rename the file.

### Capabilities and Limitations

- Runs on library models.

### See Also

- hisl\_0031: Model file names

## Check if/elseif/else patterns in MATLAB Function blocks

**Check ID:** mathworks.hism.himl\_0006

### Description

This check identifies the if/elseif/else patterns without appropriate else conditions in embedded MATLAB code.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Inappropriate if/elseif/else patterns are present in the embedded MATLAB code.	For every if/elseif/else pattern, add an else statement that includes at least one meaningful comment.

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.

**See Also**

- himl\_0006: MATLAB code if / elseif / else patterns

**Check switch statements in MATLAB Function blocks****Check ID:** mathworks.hism.himl\_0007**Description**

This check identifies the switch/case/otherwise statements without appropriate conditions in embedded MATLAB code.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
Switch statement does not have any otherwise condition.	Make sure that the switch statement has an otherwise condition.
Otherwise statement is left blank with no comments.	Make sure that the otherwise statement has at least one meaningful comment.
Switch statement has only one case statement.	Make sure that the switch statement has at least two case statements.

**Capabilities and Limitations**

- This check excludes a single **case** statement with a cell array of two or more elements.
- This check only analyzes the functions that are directly referenced by the Simulink model.
- Runs on library models.
- You can configure the check to run on referenced MATLAB files using the input parameter **Check .m files referenced in the model** in the Configuration Editor. By default this parameter is selected.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- himl\_0007: MATLAB code switch / case / otherwise patterns

## Check global variables in graphical functions

**Check ID:** mathworks.hism.hisl\_0062

### Description

This check Identifies the expressions that read and write to the same global data in a Stateflow.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more expressions operate on graphical functions and global variables used within graphical functions.	Remodel the expressions so that the functions and the global variables are not used in the same expression.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0062: Global variables in graphical functions

## Check for length of user-defined object names

**Check ID:** `mathworks.hism.hisl_0063`

### Description

This check inspects the length of the names of these user-defined objects against the Maximum Identifier length parameter in configuration settings:

- Subsystems with function name options set to User-specified.
- Data objects described in the guideline.
- Signal and parameter objects.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Length of the function name in a subsystem greater than the set threshold.	Change the function name in the Subsystem blocks to have a length less than the set threshold.
Data object names have a length greater than threshold.	Change the function name in the Subsystem blocks to have a length less than the set threshold.

### Capabilities and Limitations

- This check do not flag the signals that do not resolve to objects.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.

### See Also

- `hisl_0063`: Length of user-defined object names to improve MISRA C:2012 compliance

## Check usage of Merge blocks

**Check ID:** `mathworks.hism.hisl_0015`

## Description

This check identifies the Merge blocks that are not directly connected to a conditionally executed subsystem and have the **Allow unequal port widths** parameter set to on.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Merge block is not connected directly to a conditionally executed subsystem.	Remodel so that the Merge blocks are connected directly to conditionally executed subsystems.
The Merge block parameter <b>Allow unequal port widths</b> is set to <b>on</b> .	Set the Merge block parameter <b>Allow unequal port widths</b> to <b>off</b>
Multiple subsystem executes during a time step in conditionally executed subsystems.	Specify execution of the conditionally executed subsystems such that only one subsystem executes during a time step.
The Outport block parameter <b>Output when disabled</b> is not set to <b>held</b> for any conditionally executed subsystems in the model.	Set the Outport block parameter <b>Output when disabled</b> to <b>held</b> for each conditionally executed subsystem being merged.

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to **all**.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to **on**.

## See Also

- hisl\_0015: Usage of Merge blocks

## Check usage of conditionally executed subsystems

**Check ID:** mathworks.hism.hisl\_0012

**Description**

This check identifies the blocks with incorrect sample times in conditionally executed subsystems and asynchronously executed sample time dependent blocks.

Available with Simulink Check.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Sample time for the blocks is not set to -1 in a conditionally executed subsystem: <ul style="list-style-type: none"><li>• If Action</li><li>• Switch Case Action</li><li>• Function-Call</li><li>• Triggered</li><li>• Enabled</li></ul>	Change the sample time for the blocks to be -1.

Condition	Recommended Action
<p>The model contains asynchronously executed sample time dependent blocks:</p> <ul style="list-style-type: none"> <li>• Discrete State-Space</li> <li>• Discrete-Time Integrator</li> <li>• Discrete FIR Filter</li> <li>• Discrete Filter</li> <li>• Discrete Transfer Fcn</li> <li>• Discrete Zero-Pole</li> <li>• Transfer Fcn First Order</li> <li>• Transfer Fcn Real Zero</li> <li>• Transfer Fcn Lead or Lag</li> <li>• Discrete Transfer Function with Initial Outputs</li> <li>• Discrete Transfer Function with Initial States</li> <li>• Discrete Zero-Pole with Initial Outputs</li> <li>• Discrete Zero-Pole with Initial States</li> <li>• Discrete Derivative</li> </ul>	<p>Remodel to remove the sample time dependent blocks.</p>

### Capabilities and Limitations

- The asynchronously executed sample-time dependent blocks are flagged only if Triggered and Function-call blocks are present.
- Does not run on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.

### See Also

- hisl\_0012: Usage of conditionally executed subsystems

## Check usage of standardized MATLAB function headers

**Check ID:** mathworks.hism.himl\_0001

### Description

This check inspects all MATLAB functions in the model, local functions, and referenced MATLAB files for standardized function headers and checks for these details:

- Function name
- Function description
- Description of input variables
- Description of output variables

Following is an example of how to define function headers:

```
%<Function Name> - Description of the function
```

```
%<Input variable 1> - Description of input variable 1
```

```
%<Input variable 2> - Description of input variable 2
```

```
%<Output variable 1> - Description of output variable 1
```

Available with Simulink Check.

This check requires a Stateflow license.



## Results and Recommended Actions

Condition	Recommended Action
MATLAB functions use nonstandard function headers.	Consider adding a function header to the functions according to these guidelines: <ul style="list-style-type: none"> <li>• Must be a valid MATLAB comment.</li> <li>• Must immediately follow the function signature.</li> <li>• Must have a "Function Description" section.</li> <li>• Must have an "Inputs Description" section.</li> <li>• Must have an "Outputs Description" section.</li> </ul>

## Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- You can configure the check to run on referenced MATLAB files using the input parameter **Check .m files referenced in the model** in the Configuration Editor. By default this parameter is selected.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

## See Also

- `himl_0001`: Usage of standardized MATLAB function headers

## Check usage of relational operators in MATLAB Function blocks

**Check ID:** `mathworks.hism.himl_0008`

### Description

This check inspects all MATLAB functions in the model, local functions, and referenced MATLAB files for the relational operator statements which operate on operands of different data types.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Relational operators in the function blocks operating on operands of different data types.	Type-cast the operands to be of the same data type.

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Does not run on library models.
- You can configure the check to run on referenced MATLAB files using the input parameter **Check .m files referenced in the model** in the Configuration Editor. By default this parameter is selected.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- `himl_0008`: MATLAB code relational operator data types

## Check usage of equality operators in MATLAB Function blocks

**Check ID:** `mathworks.hism.himl_0009`

## Description

This check inspects the use of equality operators with floating-point operands in MATLAB Function blocks.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
One or more MATLAB functions have equality operators with floating-point operands.	Consider comparing ranges instead of direct comparison.

## Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Does not run on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- You can configure the check to run on referenced MATLAB files using the input parameter **Check .m files referenced in the model** in the Configuration Editor. By default this parameter is selected.
- Allows exclusions of blocks and charts.

## See Also

- `himl_0009`: MATLAB code with equal / not equal relational operators

## Check usage of logical operators and functions in MATLAB Function blocks

**Check ID:** `mathworks.hism.himl_0010`

### Description

This check identifies the logical operators and functions operating on operands with numeric data types in MATLAB Function blocks.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Logical operators or functions used on operands with numeric data types.	Type-cast the operands to be of a logical data type.

### Capabilities and Limitations

- This check only analyzes the functions that are directly referenced by the Simulink model.
- Does not run on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- You can configure the check to run on referenced MATLAB files using the input parameter **Check .m files referenced in the model** in the Configuration Editor. By default this parameter is selected.
- Allows exclusions of blocks and charts.

### See Also

- `himl_0010`: MATLAB code with logical operators and functions

## Check type and size of condition expressions

**Check ID:** `mathworks.hism.himl_0011`

## Description

This check evaluates the model to check that logical scalars are used for these condition expressions:

- `if` expressions
- `elseif` expressions
- `while` expressions
- Condition expressions of Stateflow transitions

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
<code>if</code> expression is not a logical scalar.	Change expression to a logical scalar.
<code>elseif</code> expression is not a logical scalar.	Change expression to a logical scalar.
<code>while</code> expression is not a logical scalar.	Change expression to a logical scalar.
Condition expression of Stateflow transition expression is not a logical scalar.	Change expression to a logical scalar.

## Action Results

The results table identifies each expression that contains an expression that is not a logical scalar. To review the issue, select the corresponding function link from the result table. The Model Advisor opens and highlights the corresponding function definition or, for Stateflow, opens the chart and highlights the state in which the issue occurs.

Manually change the condition expressions to logical scalars. Save the model and rerun the check.

## Capabilities and Limitations

- The conditions of Stateflow transitions and state actions are not checked when using C as the Stateflow action language.
- This check does not run on library models.

- Unreachable invariant conditions and code fragments might not be analyzed.

**See Also**

- “himl\_0011: Data type and size of condition expressions” (Simulink)
- “State Action Types” (Stateflow)
- “Transition Action Types” (Stateflow)
- “Differences Between MATLAB and C as Action Language Syntax” (Stateflow)

**Check naming of ports in Stateflow charts****Check ID:** `mathworks.hism.hisf_0016`**Description**

This check identifies the mismatches between names of Stateflow ports and associated signals. The reusable Stateflow blocks can have different port names.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Names of the input and output ports of Stateflow charts are not the same as the names of the signals connected.	Make sure that the names of the input and output ports of Stateflow charts are same as the names of the signals connected.

**Capabilities and Limitations**

- This check does not analyze port names of Stateflow Truth Tables or Stateflow State Transition Tables.
- This check considers reusable Stateflow charts as library linked charts and are not flagged.
- This check does not flag signals without names.
- Does not analyze content of library-linked blocks.
- Does not analyze content in masked subsystems.

- Allows exclusions of blocks and charts.

### See Also

- hisf\_0016: Stateflow port names

## Check scoping of Stateflow data objects

**Check ID:** mathworks.hism.hisf\_0017

### Description

This check identifies the Stateflow data objects with local scope that are not scoped at the chart level or below.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more Stateflow data objects with local scope are not defined at the chart level or below.	Make sure to define all the Stateflow data objects with local scope at the chart level or below.

### Capabilities and Limitations

- Does not analyze content of library linked blocks.
- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- hisf\_0017: Stateflow data object scoping

## Check usage of Gain blocks

**Check ID:** mathworks.hism.hisl\_0066

### Description

This check identifies the Gain blocks with value that resolves to 1, an identity matrix, or a matrix of ones.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more Gain blocks in the model resolve to 1 or an identity matrix.	Remodel the Gain blocks so that the gain value does not resolve to 1, an identity matrix, or a matrix of ones.

### Capabilities and Limitations

- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- hisl\_0066: Usage of Gain blocks

## Check data type of loop control variables

**Check ID:** mathworks.hism.hisl\_0102

### Description

This check identifies loop control variables using non-integer data types on the following:

- For iterator blocks.
- For loops in MATLAB function blocks.

Available with Simulink Check.

This check requires a Stateflow license.



## Results and Recommended Actions

Condition	Recommended Action
One or more For Iterator blocks are using non-integer data type for loop control counter variable.	Set the data type of loop control counter variable to an integer data type.
One or more For loops are using non-integer data type for loop control counter variable in MATLAB Function blocks.	Set the data type of loop control counter variable to an integer data type.

## Capabilities and Limitations

- This check does not look at loop control variables inside Stateflow charts.
- Does not run on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Allows exclusions of blocks and charts.

## See Also

- hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance

## Check for inappropriate use of transition paths

**Check ID:** mathworks.hism.hisf\_0014

### Description

This check inspects the use of junctions inside states and identifies the junctions that lie on a path that goes in and out of a state.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more transition paths in the model traverses through a state without ending on a substate.	Remodel the junctions to avoid transition paths that go into and out of a state without ending on a substate.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `all`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of charts. You cannot exclude a Stateflow element directly.

### See Also

- `hisf_0014`: Usage of transition paths (passing through states)

## Check usage of bitwise operations in Stateflow charts

**Check ID:** `mathworks.hism.hisf_0003`

### Description

Identifies the usage of signed integer operands to bitwise operators in Stateflow charts with C action language.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more Stateflow objects in the model use signed integer operands with bitwise operators.	Make sure to not use signed integer operands with bitwise operators.

### Capabilities and Limitations

- The following MATLAB bit wise operations are not flagged with this check
  - swapbytes
  - bitset
  - bitget
  - bitcmp
- Does not run on library models.
- Analyzes content of library linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to all.
- Analyzes content in all masked subsystems.

### See Also

- hisf\_0003: Usage of bitwise operations

## Check safety-related diagnostic settings for signal data

**Check ID:** mathworks.hism.hisl\_0314

Check model configuration for diagnostic settings that apply to signal data and that can impact safety.

### Description

This check verifies that model diagnostic configuration parameters pertaining to signal data are set optimally for generating code for a safety-related application.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The diagnostic that specifies how the Simulink software resolves signals associated with <code>Simulink.Signal</code> objects is set to <code>Explicit and implicit</code> or <code>Explicit and warn implicit</code>. For safety-related applications, model developers should be required to define signal resolution explicitly. (See DO-331, Section MB.6.3.3.b - Software architecture is consistent.)</p>	<p>Set <b>Signal resolution</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>SignalResolutionControl</code> to <code>Explicit only</code>. This provides predictable operation by requiring users to define each signal and block setting that must resolve to <code>Simulink.Signal</code> objects in the workspace.</p> <p>Alternatively, to disable the use of <code>Simulink.Signal</code> objects, set the configuration parameter to <code>None</code>.</p>
<p>The Product block diagnostic that detects a singular matrix while inverting one of its inputs in matrix multiplication mode is set to <code>none</code> or <code>warning</code>. Division by a singular matrix can result in numeric exceptions when executing generated code. This is not acceptable in safety-related systems. (See DO-331, Section MB.6.3.1.g - Algorithms are accurate, DO-331, Section MB.6.3.2.g - Algorithms are accurate, and MISRA C:2012, Dir 4.1.)</p>	<p>Set <b>Division by singular matrix</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>CheckMatrixSingularityMsg</code> to <code>error</code>.</p>
<p>The diagnostic that detects when the Simulink software cannot infer the data type of a signal during data type propagation is set to <code>none</code> or <code>warning</code>. For safety-related applications, model developers must verify the data types of signals. (See DO-331, Section MB.6.3.1.e - High-level requirements conform to standards, and DO-331, Section MB.6.3.2.e - Low-level requirements conform to standards.)</p>	<p>Set <b>Underspecified data types</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>UnderSpecifiedDataTypeMsg</code> to <code>error</code>.</p>

Condition	Recommended Action
<p>The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type is set to <b>none</b> or <b>warning</b>. Undetected numeric overflows can result in unexpected application behavior. (See DO-331, Section MB.6.3.1.g - Algorithms are accurate, DO-331, Section MB.6.3.2.g - Algorithms are accurate, and MISRA C:2012, Dir 4.1.)</p>	<p>Set <b>Wrap on overflow</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>IntegerOverflowMsg</code> to error.</p>
<p>The diagnostic that detects whether the value of a signal is too large to be represented by the signal data type, resulting in a saturation, is set to <b>none</b> or <b>warning</b>. Undetected numeric overflows can result in unexpected application behavior. (See DO-331, Section MB.6.3.1.g - Algorithms are accurate, DO-331, Section MB.6.3.2.g - Algorithms are accurate, and MISRA C:2012, Dir 4.1.)</p>	<p>Set <b>Saturate on overflow</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>IntegerSaturationMsg</code> to error.</p>
<p>The diagnostic that detects when the value of a block output signal is <b>Inf</b> or <b>NaN</b> at the current time step is set to <b>none</b> or <b>warning</b>. When this type of block output signal condition occurs, numeric exceptions can result, and numeric exceptions are not acceptable in safety-related applications. (See DO-331, Section MB.6.3.1.g - Algorithms are accurate, DO-331, Section MB.6.3.2.g - Algorithms are accurate, and MISRA C:2012, Dir 4.1.)</p>	<p>Set <b>Inf or NaN block output</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>SignalInfNanChecking</code> to error.</p>
<p>The diagnostic that detects Simulink object names that begin with <b>rt</b> is set to <b>none</b> or <b>warning</b>. This diagnostic prevents name clashes with generated signal names that have an <b>rt</b> prefix. (See DO-331, Section MB.6.3.1.e - High-level requirements conform to standards, and DO-331, Section MB.6.3.2.e - Low-level requirements conform to standards.)</p>	<p>Set <b>"rt" prefix for identifiers</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter <code>RTPrefix</code> to error.</p>

Condition	Recommended Action
The diagnostic that detects simulation range checking is set to none or warning. This diagnostic detects when signals exceed their specified ranges during simulation. Simulink compares the signal values that a block outputs with the specified range and the block data type. (See DO-331, Section MB.6.3.1.g - Algorithms are accurate, DO-331, Section MB.6.3.2.g - Algorithms are accurate, and MISRA C:2012, Dir 4.1.)	Set <b>Simulation range checking</b> (Simulink) on the <b>Diagnostics &gt; Data Validity</b> pane in the Configuration Parameters dialog box or set the parameter SignalRangeChecking to error.

### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to signal data and that can impact safety.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals
- “Model Configuration Parameters: Data Validity Diagnostics” (Simulink)
- “View Diagnostics” (Simulink)

## Check for model elements that do not link to requirements

**Check ID:** mathworks.hism.hisl\_0070

Check whether Simulink model elements link to a requirements document.

### Description

This check verifies whether model objects link to a document containing engineering requirements for traceability.

Available with Simulink Check.

This check requires a Simulink Requirements license.

### Results and Recommended Actions

Condition	Recommended Action
Blocks do not link to a requirements document.	Link to requirements document.

### Capabilities and Limitations

- Runs on library models.
- Analyzes the content of library linked blocks. By default, the input parameter **Follow links** is set to `off`.
- Analyzes content in masked subsystems that have no workspaces and no dialogs. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.
- Does not allow the exclusion of Stateflow elements.

### Tip

Run this check from the top model or subsystem that you want to check.

### See Also

- hisl\_0070: Placement of requirement links in a model
- hisl\_0070: Placement of requirement links in a model
- “Requirements Traceability in Simulink” (Simulink)
- “Requirements Traceability” (Simulink Requirements)
- “Find Model Elements in Simulink Models” (Simulink)
- DO-331, Section MB.6.3.1.f - High-level requirements trace to system requirements
- DO-331, Section MB.6.3.2.f - Low-level requirements trace to high-level requirements
- IEC 61508-3, Table A.2 (12) - Computer-aided specification and design tools, Table A.2 (9) - Forward traceability between the software safety requirements specification and software architecture, Table A.2 (10) - Backward traceability between the software safety requirements specification and software architecture, Table A.4 (8) - Forward traceability between the software safety requirements specification and software design, Table A.8 (1) - Impact analysis

- IEC 62304, 5.2 - Software requirements analysis, 7.4.2 - Analyze impact of software changes on existing risk control measures
- ISO 26262-6, Table 8 (1a) - Documentation of the software unit design in natural language, ISO 26262-6: 7.4.2.a - The verifiability of the software architectural design, ISO 26262-8: 8.4.3 Change request analysis
- EN 50128, Table A.3 (23) - Modeling supported by computer aided design and specification tools, Table D.58 - Traceability, Table A.10 (1) - Impact Analysis

### Check safety-related optimization settings for Loop unrolling threshold

**Check ID:** `mathworks.hism.hisl_0051`

Check optimization settings in the model configuration that apply to Loop unrolling threshold and might impact safety.

#### Description

This check verifies that the model optimization configuration parameters pertaining to the minimum signal or parameter width for which a `for` loop is generated is set optimally for generating code for a safety-related application.

Available with Simulink Check.

#### Results and Recommended Actions

Condition	Recommended Action
The optimization setting to specify the minimum signal or parameter width for which a <code>for</code> loop is generated is set to a value less than 2.	In the Configuration Parameters dialog box, set <b>Loop unrolling threshold</b> or set the parameter <code>RollThreshold</code> to a value equal to or greater than 2.

#### Action Results

Clicking **Modify Settings** configures model diagnostic settings that apply to solvers and that can impact safety.



### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Analyzes content in masked subsystems that have no workspace and no dialog boxes.

### See Also

- “hisl\_0051: Configuration Parameters > Code Generation > Optimization > Loop unrolling threshold” (Simulink)

## Check safety-related optimization settings for specified minimum and maximum values

**Check ID:** mathworks.hism.hisl\_0056

Check model configuration for optimization settings that can impact safety.

### Description

This check verifies that model optimization configuration parameters are set optimally for generating code for a safety-related application. Although highly optimized code is desirable for most real-time systems, some optimizations can have undesirable side effects that impact safety.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The optimization that uses the specified minimum and maximum values for signals and parameters to optimize the generated code is selected. This might result in requirements without traceable code. (See DO-331 Section MB.6.3.4.e - Source code is traceable to low-level requirements.)	If you have an Embedded Coder license and are using an ERT-based system target file, clear Configuration Parameter <b>Optimize using the specified minimum and maximum values</b> (Simulink Coder), or parameter UseSpecifiedMinMax to off.

### Action Results

Clicking **Modify Settings** configures model optimization settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- “Optimize Generated Code Using Minimum and Maximum Values” (Embedded Coder)
- Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
- hisl\_0056: Configuration Parameters > Optimization > Optimize using the specified minimum and maximum values

## Check model object names

**Check ID:** `mathworks.hism.hisl_0032`

Check model object names.

### Description

This check verifies that the following model object names comply with your own modeling guidelines or the high-integrity modeling guidelines. The check also verifies that the model object does not use a reserved name.

- Blocks
- Signals
- Parameters
- Buses
- Stateflow objects

Reserved names:

- MATLAB keywords
- Reserved keywords for C, C++, and code generation. For a complete list, see “Reserved Keywords” (Simulink Coder)

- int8, uint8
- int16, uint16
- int32, uint32
- inf, Inf
- NaN, nan
- eps
- intmin, intmax
- realmin, realmax
- pi
- infinity
- Nil

---

**Note** In some cases, the Model Advisor reports an issue in multiple subchecks of this check.

---

Available with Simulink Check.

### Input Parameters

To specify the naming standard and model object names that the check flags, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check model object names**. In the **Input Parameters** pane, for each of the model objects, select one of the following:
  - MAAB to use the MAAB naming standard. When you select MAAB, the check uses the regular expression  $(^{\{32, \}}$) | ([^a-zA-Z_0-9]) | (^d) | (^ ) | (__) | (^_) | (_$)$  to verify that names:
    - Use these characters: a-z, A-Z, 0-9, and the underscore (\_).
    - Do not start with a number.
    - Do not use underscores at the beginning or end of a string.
    - Do not use more than one consecutive underscore.
    - Use strings that are less than 32 characters.

- **Custom** to use your own naming standard. When you select **Custom**, you can enter your own **Regular expression for prohibited <model object> names**. For example, if you want to allow more than one consecutive underscore, enter `(^\{32,\}$)|([\^a-zA-Z_0-9])|(\^d)|(\^ )|(\^_)|(\_$)`
  - **None** if you do not want the check to verify the model object name
- 2 Click **Apply**.
  - 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
The model object names do not comply with the naming standard specified in the input parameters.	Update the model object names to comply with your own guidelines or the high-integrity guidelines.

### Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Does not analyze content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- Check model object names
- MAAB guideline, Version 3.0: jc\_0201: Usable characters for Subsystem names
- MAAB guideline, Version 3.0: jc\_0211: Usable characters for Inport blocks and Outport blocks
- MAAB guideline, Version 3.0: jc\_0221: Usable characters for signal line names
- MAAB guideline, Version 3.0: jc\_0231: Usable characters for block names
- MAAB guideline, Version 3.0: na\_0019: Restricted Variable Names
- MAAB guideline, Version 3.0: na\_0030: Usable characters for Simulink Bus names

## Check for blocks not recommended for C/C++ production code deployment

**Check ID:** `mathworks.hism.hisl_0020`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

### Description

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder.

In some instances, this check flags blocks that are supported for code generation. For these blocks, you should review the footnote information that is provided in the support notes and adhere to the recommended action provided by the Model Advisor.

Available with Simulink Check and Embedded Coder.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.
The Model Advisor flags blocks that are supported for code generation	Review the support notes and adhere to the recommended action provided by the Model Advisor.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
- “hisl\_0020: Blocks not recommended for MISRA C:2012 compliance” (Simulink)
- “Blocks and Products Supported for Code Generation” (Simulink Coder)
- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets
- EN 50128, Table A.4 (11) - Language Subset
- “Blocks and Products Supported for Code Generation” (Simulink Coder)

**Check configuration parameters for MISRA C:2012**

**Check ID:** `mathworks.misra.CodeGenSettings`

Identify configuration parameters that can impact MISRA C:2012 compliant code generation.

**Description**

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
<b>Math and Data Types</b>	
Configuration parameter <b>Use division for fixed-point net slope computation</b> is not set to 0n or Use division for reciprocals of integers only.	Set <b>Use division for fixed-point net slope computation</b> to 0n or Use division for reciprocals of integers only.

Condition	Recommended Action
<p>Configuration parameter <b>Inf or NaN block output</b> is set to None or error and <b>Support non-finite numbers</b> is set to on.</p> <p>Configuration parameter <b>Inf or NaN block output</b> is set to None and <b>Support non-finite numbers</b> is set to off.</p>	<p>When <b>Support non-finite numbers</b> is:</p> <ul style="list-style-type: none"> <li>• on, set <b>Inf or NaN block output</b> to warning</li> <li>• off, set <b>Inf or NaN block output</b> to warning or error</li> </ul>
<p>Configuration parameter <b>Model Verification block enabling</b> is set to Use local settings or Enable All.</p>	<p>Set <b>Model Verification block enabling</b> to Disable All.</p>
<p>Configuration parameter <b>Undirected event broadcasts</b> is set to none or warning.</p>	<p>Set <b>Undirected event broadcasts</b> to error.</p>
<p>Configuration parameter <b>Wrap on overflow</b> is set to None</p>	<p>Set configuration parameter <b>Wrap on overflow</b> to warning or error.</p>
<b>Hardware Implementation</b>	
<p>Configuration parameter <b>Production hardware signed integer division rounds to</b> is set to Undefined</p>	<p>Set <b>Production hardware signed integer division rounds to</b> to Zero or Floor.</p>
<p>Configuration parameter <b>Shift right on a signed integer as arithmetic shift</b> is selected.</p>	<p>Clear <b>Shift right on a signed integer as arithmetic shift</b> .</p>
<b>Simulation Target</b>	
<p>Configuration parameter <b>Compile-time recursion limit for MATLAB functions</b> is set to a value other than 0 .</p>	<p>Set <b>Compile-time recursion limit for MATLAB functions</b> to 0 .</p>
<p>Configuration parameter <b>Dynamic memory allocation in MATLAB functions</b> is selected.</p>	<p>Clear <b>Dynamic memory allocation in MATLAB functions</b>.</p>
<p>Configuration parameter <b>Enable run-time recursion for MATLAB functions</b> is selected.</p>	<p>Clear <b>Enable run-time recursion for MATLAB functions</b>.</p>
<b>Code Generation</b>	

Condition	Recommended Action
<p>Configuration parameter <b>Bitfield declarator type specifier</b> is set to <code>uchar_T</code> when any of these parameters are selected:</p> <ul style="list-style-type: none"> <li>• <b>Pack Boolean data into bitfields</b></li> <li>• <b>Use bitsets for storing state configuration</b></li> <li>• <b>Use bitsets for storing Boolean data</b></li> </ul>	<p>Set <b>Bitfield declarator type specifier</b> to <code>uint_T</code>.</p>
<p>Configuration parameter <b>Casting Modes</b> is not set to Standards Compliant.</p>	<p>Set <b>Casting Modes</b> to Standards Compliant.</p>
<p>Configuration parameter <b>Code replacement library</b> is not set to None or AUTOSAR 4.0.</p>	<p>Set <b>Code replacement library</b> to None or AUTOSAR 4.0</p>
<p>Configuration parameter <b>External mode</b> is selected.</p>	<p>Clear <b>External mode</b>.</p>
<p>Configuration parameter <b>Generate shared constants</b> is selected.</p>	<p>Clear <b>Generate shared constants</b>.</p>
<p>Configuration parameter <b>Include comments</b> is cleared.</p>	<p>Select <b>Include comments</b>.</p>
<p>Configuration parameter <b>MAT-file logging</b> is selected.</p>	<p>Clear <b>MAT-file logging</b></p>
<p>For ERT-based target systems, configuration parameter <b>MATLAB user comments</b> is cleared.</p>	<p>Select <b>MATLAB user comments</b>.</p>
<p>A value for configuration parameter <b>Maximum identifier length</b> is not provided.</p>	<p>Set the value to the implementation-dependent limit. The default is 31.</p>
<p>Configuration parameter <b>Parenthesis level</b> is not set to Maximum (Specify precedence with parentheses).</p>	<p>Set <b>Parentheses level</b> to Maximum (Specify precedence with parentheses).</p>



Condition	Recommended Action
For ERT-based target systems, configuration parameter <b>Preserve static keyword in function declarations</b> is cleared when <b>File packaging format</b> is set to Compact or Compact (with separate data file)	Select <b>Preserve static keyword in function declarations</b> .
Configuration parameter <b>Replace multiplications by powers of two with signed bitwise shifts</b> is selected.	Clear <b>Replace multiplications by powers of two with signed bitwise shifts</b> .
Configuration parameter <b>Shared code placement</b> is set to Auto.	Set <b>Shared code placement</b> to Shared location
For ERT-based target systems, configuration parameter <b>Support continuous time</b> is selected	Clear <b>Support continuous time</b> .
For ERT-based target systems, configuration parameter <b>Support non-inlined S-functions</b> is selected	Clear <b>Support non-inlined S-functions</b> .
Configuration parameter <b>System-generated identifiers</b> is set to Classic.	Set <b>System-generated identifiers</b> to Shortened.
Configuration parameter <b>System target file</b> is set to a GRT-based target.	Set <b>System target file</b> to an ERT-based target.
Configuration parameter <b>Use dynamic memory allocation for model initialization</b> is selected when <b>Code Interface Packaging</b> is set to Reusable Function.	Clear <b>Use dynamic memory allocation for model initialization</b> .  <b>Note</b> Select only when <b>Code Interface Packaging</b> is set to Reusable Function.

### Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

---

**Note** When you click **Modify All** for models with a GRT-based target, the Model Advisor does not update the **System target file** configuration parameter to an ERT-based system.

---

Parameter subchecks depend on the results of the parameter noted with **D** in the results table. When the result is *D-Warning*, the **Current Value** column in the results table states *Prerequisite constraint not met* for the subchecks. After you change the parameter, rerun the check.

---

**Note** Some subchecks are specific to configuration parameters for ERT-based systems. These parameters are not updated when you click **Modify All** unless you change the model to an ERT-based system.

---

### Capabilities and Limitations

This check does not review referenced models.

#### See Also

- hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)

## Check for blocks not recommended for MISRA C:2012

**Check ID:** `mathworks.misra.BlkSupport`

Identify blocks that are not supported or recommended for MISRA C:2012 compliant code generation.

### Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• 1-D Lookup Table</li> <li>• 2-D Lookup Table</li> <li>• n-D Lookup Table</li> </ul>	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• Lookup Table</li> <li>• Lookup Table (2-D)</li> </ul>	Consider replacing the deprecated Lookup Table blocks.
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.
String blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• Compose String</li> <li>• Scan String</li> <li>• String to Single</li> <li>• String to Double</li> <li>• To String</li> </ul>	Consider replacing the String blocks with blocks recommended for production.

## Capabilities and Limitations

You can:

- Run this check on your library models.

- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **See Also**

- hisl\_0020: Blocks not recommended for MISRA C:2012 compliance
- na\_0027: Use of only standard library blocks
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Model Advisor Exclusion Overview”

## IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks

### In this section...

“IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks” on page 2-149

“Display model metrics and complexity report” on page 2-150

“Check for unconnected objects” on page 2-151

“Check safety-related code generation settings” on page 2-152

“Check usage of Math Operations blocks” on page 2-156

“Check usage of Logic and Bit Operations blocks” on page 2-158

“Check usage of Ports and Subsystems blocks” on page 2-160

“Display configuration management data” on page 2-163

### IEC 61508, IEC 62304, ISO 26262, and EN 50128 Checks

IEC 61508, IEC 62304, ISO 26262, and EN 50128 checks facilitate designing and troubleshooting models, subsystems, and the corresponding generated code for applications to comply with IEC 61508-3, IEC 62304, ISO 26262-6, or EN 50128.

The Model Advisor performs a checkout of the Simulink Check license when you run the IEC 61508, IEC 62304, ISO 26262, or EN 50128 checks.

These checks are certified by the IEC Certification Kit for use in development processes that must comply with IEC 61508, ISO 26262, EN 50128, or derivative standards.

#### Tips

If your model uses model referencing, run the IEC 61508, IEC 62304, ISO 26262, or EN 50128 checks on all referenced models before running them on the top-level model.

#### See Also

- “Run Model Advisor Checks and Review Results”
- IEC 61508-3 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements
- IEC 62304 Medical device software - Software life cycle processes
- ISO 26262-6 Road vehicles - Functional safety - Part 6: Product development: Software level

- EN 50128 Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems
- Embedded Coder documentation:
  - “IEC 61508 Standard” (Embedded Coder)
  - “IEC 62304 Standard” (Embedded Coder)
  - “ISO 26262 Standard” (Embedded Coder)
  - “EN 50128 Standard” (Embedded Coder)

### **Display model metrics and complexity report**

**Check ID:** `mathworks.iec61508.MdlMetricsInfo`

Display number of elements and name, level, and depth of subsystems for the model or subsystem.

#### **Description**

The IEC 61508, ISO 26262, and EN 50128 standards recommend the usage of size and complexity metrics to assess the software under development. This check provides metrics information for the model. The provided information can be used to inspect whether the size or complexity of the model or subsystem exceeds given limits. The check displays:

- A block count for each Simulink block type contained in the given model, including library linked blocks.
- A count of Stateflow constructs in the given model (if applicable).
- Name, level, and depth of the subsystems contained in the given model (if applicable).
- The maximum subsystem depth of the given model.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
N/A	This summary is provided for your information. No action is required.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

## See Also

- IEC 61508-3, Table B.9 (1) - Software module size limit, Table B.9 (2) - Software complexity control
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1a) - Enforcement of low complexity, Table 3 (a) - Hierarchical structure of software components, Table 3 (b) - Restricted size of software components, and Table 3 (c) - Restricted size of interfaces
- EN 50128, Table A.12 (8) - Limited size and complexity of Functions, Subroutines and Methods and (9) Limited number of subroutine parameters
- `sldiagnostics` in the Simulink documentation
- “Cyclomatic Complexity for Stateflow Charts” (Simulink Coverage)

## Check for unconnected objects

**Check ID:** `mathworks.iec61508.Unconnected0bjects`

Identify unconnected lines, input ports, and output ports in the model.

### Description

Unconnected objects are likely to cause problems propagating signal attributes such as data, type, sample time, and dimensions.

Ports connected to Ground or Terminator blocks pass this check.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
There are unconnected lines, input ports, or output ports in the model or subsystem.	<ul style="list-style-type: none"><li>• Double-click an element in the list of unconnected items to locate the item in the model diagram.</li><li>• Connect the objects identified in the results.</li></ul>

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- IEC 61508-3, Table A.3 (3) - Language subset
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques
- EN 50128, Table A.4 (11) - Language Subset
- “Signal Basics” (Simulink)

## Check safety-related code generation settings

**Check ID:** `mathworks.do178.CodeSet`

Check model configuration for code generation settings that can impact safety.

### Description

This check verifies that model configuration parameters for code generation are set optimally for a safety-related application.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
The option to include comments in the generated code is cleared. Comments provide good traceability between the code and the model.	Select <b>Include comments</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>GenerateComments</code> to on.
The option to include comments that describe the code for blocks is cleared. Comments provide good traceability between the code and the model.	Select “Simulink block comments” (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>SimulinkBlockComments</code> to on.
The option to include comments that describe the code for blocks eliminated from a model is cleared. Comments provide good traceability between the code and the model.	Select <b>Show eliminated blocks</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ShowEliminatedStatement</code> to on.
The option to include the names of parameter variables and source blocks as comments in the model parameter structure declaration in <code>model_prm.h</code> is cleared. Comments provide good traceability between the code and the model.	Select <b>Verbose comments for 'Model default' storage class</b> (Simulink Coder) on the <b>Code Generation &gt; Comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ForceParamTrailComments</code> to on.
The option to include requirement descriptions assigned to Simulink blocks as comments is cleared. Comments provide good traceability between the code and the model.	Select <b>Requirements in block comments</b> (Simulink Coder) on the <b>Code Generation &gt; Custom comments</b> pane in the Configuration Parameters dialog box or set the parameter <code>ReqsInCode</code> to on.
The option to generate nonfinite data and operations is selected. Support for nonfinite numbers is inappropriate for real-time embedded systems.	Clear <b>Support: non-finite numbers</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportNonFinite</code> to off.
The option to generate and maintain integer counters for absolute and elapsed time is selected. Support for absolute time is inappropriate for real-time safety-related systems.	Clear <b>Support: absolute time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportAbsoluteTime</code> to off.

Condition	Recommended Action
The option to generate code for blocks that use continuous time is selected. Support for continuous time is inappropriate for real-time safety-related systems.	Clear <b>Support: continuous time</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SupportContinuousTime</code> to off.
The option to generate code for noninlined S-functions is selected. This option requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems.	Clear <b>Support: non-inlined S-functions</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter <code>SupportNonInlinedSFcns</code> to off.
The option to generate model function calls compatible with the main program module of the pre-R2012a GRT target is selected. This option is inappropriate for real-time safety-related systems.	Clear <b>Classic call call interface</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>GRTInterface</code> to off.
The option to generate the <code>model_update</code> function is cleared. Having a single call to the output and update functions simplifies the interface to the real-time operating system (RTOS) and simplifies verification of the generated code.	Select <b>Single output/update function</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>CombineOutputUpdateFcns</code> to on.
The option to generate the <code>model_terminate</code> function is selected. This function deallocates dynamic memory, which is unsuitable for real-time safety-related systems.	Clear <b>Terminate function</b> (Simulink Coder) on the <b>Code Generation</b> pane in the Configuration Parameters dialog box or set the parameter <code>IncludeMdlTerminateFcn</code> to off.
The option to log or monitor error status is cleared. If you do not select this option, the Simulink Coder product generates extra code that might not be reachable for testing.	Select <b>Remove error status field in real-time model data structure</b> (Simulink Coder) on the <b>Code Generation &gt; Interface</b> pane in the Configuration Parameters dialog box or set the parameter <code>SuppressErrorStatus</code> to on.
MAT-file logging is selected. This option adds extra code for logging test points to a MAT-file, which is not supported by embedded targets. Use this option only in test harnesses.	Clear <b>MAT-file logging</b> (Simulink Coder) in the Configuration Parameters dialog box or set the parameter <code>MatFileLogging</code> to off.

Condition	Recommended Action
The option that specifies the style for parenthesis usage is set to <b>Minimum (Rely on C/C++ operators precedence)</b> or to <b>Nominal (Optimize for readability)</b> . For safety-related applications, explicitly specify precedence with parentheses.	Set parameter <code>ParenthesesLevel</code> to <b>Maximum (Specify precedence with parentheses)</b> .
The option that specifies whether to preserve operand order is cleared. This option increases the traceability of the generated code.	Set parameter <code>PreserveExpressionOrder</code> to <b>on</b> .
The option that specifies whether to preserve empty primary condition expressions in <code>if</code> statements is cleared. This option increases the traceability of the generated code.	Set parameter <code>PreserveIfCondition</code> to <b>on</b> .
The minimum number of characters specified for generating name mangling strings is less than four. You can use this option to minimize the likelihood that parameter and signal names will change during code generation when the model changes. Use of this option assists with minimizing code differences between file versions, decreasing the effort to perform code reviews.	Set <b>Minimum mangle length</b> (Simulink Coder) on the <b>Code Generation &gt; Symbols</b> pane in the Configuration Parameters dialog box or the parameter <code>MangleLength</code> to a value of 4 or greater.

### Action Results

Clicking **Modify Settings** configures model code generation settings that can impact safety.

Subchecks depend on the results of the subchecks noted with **D** in the results table in the Model Advisor window.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
- EN 50128, Table A.4 (11) 'Language Subset'
- "hisl\_0038: Configuration Parameters > Code Generation > Comments" (Simulink)
- "hisl\_0039: Configuration Parameters > Code Generation > Interface" (Simulink)
- "hisl\_0047: Configuration Parameters > Code Generation > Code Style" (Simulink)
- "hisl\_0049: Configuration Parameters > Code Generation > Identifiers" (Simulink)
- "Model Configuration Parameters: Comments" (Simulink Coder)
- "Model Configuration Parameters: Comments" (Simulink Coder)
- "Model Configuration Parameters: Code Generation Identifiers" (Simulink Coder)
- "Model Configuration Parameters: Code Generation Interface" (Simulink Coder)
- "Model Configuration Parameters: Code Style" (Embedded Coder)

### Check usage of Math Operations blocks

**Check ID:** `mathworks.iec61508.MathOperationsBlocksUsage`

Identify usage of Math Operation blocks that might impact safety.

#### Description

This check inspects the usage of the following blocks:

- Abs
- Assignment
- Gain

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains an Absolute Value block that is operating on one of the following:</p> <ul style="list-style-type: none"> <li>• A boolean or an unsigned input data type. This condition results in unreachable simulation pathways through the model and might result in unreachable code</li> <li>• A signed integer value with the <b>Saturate on integer overflow</b> check box not selected. For signed data types, the absolute value of the most negative value is problematic because it is not representable by the data type. This condition results in an overflow in the generated code.</li> </ul>	<p>If the identified Absolute Value block is operating on a boolean or unsigned data type, do one of the following:</p> <ul style="list-style-type: none"> <li>• Change the input of the Absolute Value block to a signed input type.</li> <li>• Remove the Absolute Value block from the model.</li> </ul> <p>If the identified Absolute Value block is operating on a signed data type, in the <b>Block Parameters &gt; Signal Attributes</b> dialog box, select <b>Saturate on integer overflow</b>.</p>
<p>The model or subsystem contains Gain blocks with a of value 1 or an identity matrix.</p>	<p>If you are using Gain blocks as buffers, consider replacing them with Signal Conversion blocks.</p>
<p>The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter <b>Action if any output element is not assigned</b> set to <b>Error</b> or <b>Warning</b>.</p>	<p>Set block parameter <b>Action if any output element is not assigned</b> to one of the recommended values:</p> <ul style="list-style-type: none"> <li>• <b>Error</b>, if Assignment block is not in an Iterator subsystem.</li> <li>• <b>Warning</b>, if Assignment block is in an Iterator subsystem.</li> </ul>

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (3) - Defensive programming, Table A.3 (2) - Strongly typed programming language, Table B.8 (3) - Control Flow Analysis
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques, Table 9 (1e) - Control flow analysis
- EN 50128, Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming, EN 50128, Table A.4 (8) - Strongly Typed Programming Language, Table A.19 (3) - Control Flow Analysis
- MISRA C:2012, Dir 4.1
- MISRA C:2012, Rule 9.1
- hisl\_0001: Usage of Abs block
- hisl\_0002: Usage of Math Function blocks (rem and reciprocal)
- hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)
- hisl\_0029: Usage of Assignment blocks
- hisl\_0066: Usage of Gain blocks

### Check usage of Logic and Bit Operations blocks

**Check ID:** `mathworks.iec61508.LogicBlockUsage`

Identify usage of Logical Operator and Bit Operations blocks that might impact safety.

#### Description

This check inspects the usage of:

- Blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare To Zero, Detect Change, and If blocks
- Logical Operator blocks

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains a block computing a relational operator that is operating on different data types. The condition can lead to unpredictable results in the generated code.	For the identified blocks, use common data types as inputs. You can use Data Type Conversion blocks to change input data types.
The model or subsystem contains a block computing a relational operator that does not have Boolean output. The condition can lead to unpredictable results in the generated code.	For the specified blocks, on the Block Parameters > Signal Attributes pane, set the <b>Output data type</b> to boolean.
The model or subsystem contains a block computing a relational operator that uses the == or ~= operator to compare floating-point signals. The use of these operators on floating-point signals is unreliable and unpredictable because of floating-point precision issues. These operators can lead to unpredictable results in the generated code.	For the identified block, do one of the following: <ul style="list-style-type: none"> <li>• Change the signal data type.</li> <li>• Rework the model to eliminate using == or ~= operators on floating-point signals.</li> </ul>
The model or subsystem contains a Logical Operator block that has inputs or outputs that are not Boolean inputs or outputs. The block might result in floating-point equality or inequality comparisons in the generated code.	<ul style="list-style-type: none"> <li>• Modify the Logical Operator block so that all inputs and outputs are Boolean. On the Block Parameters &gt; Signal Attributes pane, consider selecting <b>Require all inputs to have the same data type</b> and setting <b>Output data type</b> to boolean.</li> <li>• In the Configuration Parameters dialog box, consider selecting the <b>Implement logic signals as boolean data (vs. double)</b>.</li> </ul>

## Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'
- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 61508-3, Table A.4 (3) 'Defensive programming'
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) 'Use of language subsets'
- ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing'
- EN 50128, Table A.4 (11) 'Language Subset'
- EN 50128, Table A.4 (8) 'Strongly Typed Programming Language'
- EN 50128, Table A.3 (1) 'Defensive Programming'
- MISRA C:2012, Dir 1.1
- MISRA C:2012, Rule 10.1
- "hisl\_0016: Usage of blocks that compute relational operators" (Simulink)
- "hisl\_0017: Usage of blocks that compute relational operators (2)" (Simulink)
- "hisl\_0018: Usage of Logical Operator block" (Simulink)

## Check usage of Ports and Subsystems blocks

**Check ID:** `mathworks.iec61508.PortsSubsystemsUsage`

Identify usage of Ports and Subsystems blocks that might impact safety.

### Description

This check inspects the usage of:

- For Iterator blocks
- While Iterator blocks
- If blocks
- Switch Case blocks

The check does not flag Switch Case blocks that do not use integer data types or enumeration values for inputs. To comply with "hisl\_0011: Usage of Switch Case blocks



and Action Subsystem blocks” (Simulink) - C, use an integer data type or an enumeration value for the inputs to Switch Case blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
<p>The model or subsystem contains a For Iterator block that has variable iterations. This condition can lead to unpredictable execution times or infinite loops in the generated code.</p>	<p>For the identified For Iterator blocks, do one of the following:</p> <ul style="list-style-type: none"> <li>• Set the <b>Iteration limit source</b> parameter to <code>internal</code>.</li> <li>• If the <b>Iteration limit source</b> parameter must be <code>external</code>, use a Constant, Probe, or Width block as the source.</li> <li>• Clear the <b>Set next i (iteration variable) externally</b> check box.</li> <li>• Consider selecting the <b>Show iteration variable</b> check box and observe the iteration value during simulation.</li> </ul>
<p>The model or subsystem contains a While Iterator block that has unlimited iterations. This condition can lead to infinite loops in the generated code. mo</p>	<p>For the identified While Iterator blocks:</p> <ul style="list-style-type: none"> <li>• Set the <b>Maximum number of iterations (-1 for unlimited)</b> parameter to a positive integer value.</li> <li>• Consider selecting the <b>Show iteration number port</b> check box and observe the iteration value during simulation.</li> </ul>
<p>The model or subsystem contains an If block with an If expression or Elseif expressions that might cause floating-point equality or inequality comparisons in generated code.</p>	<p>Modify the expressions in the If block to avoid floating-point equality or inequality comparisons in generated code.</p>
<p>The model or subsystem contains an If block using Elseif expressions without an Else condition.</p>	<p>In the If block Block Parameters dialog box, select <b>Show else condition</b>. Connect the resulting Else output port to an If Action Subsystem block.</p>

<b>Condition</b>	<b>Recommended Action</b>
The model or subsystem contains an If block with output ports that do not connect to If Action Subsystem blocks.	Verify that output ports of the If block connect to If Action Subsystem blocks.
The model or subsystem contains an Switch Case block without a default case.	In the Switch Case block Block Parameters dialog box, select <b>Show default case</b> . Connect the resulting default output port to a Switch Case Action Subsystem block.
The model or subsystem contains a Switch Case block with an output port that does not connect to a Switch Case Action Subsystem block.	Verify that output ports of the Switch Case blocks connect to Switch Case Action Subsystem blocks.
<p>The model or subsystem contains one of the following time-dependent blocks in a For Iterator or While Iterator subsystem:</p> <ul style="list-style-type: none"> <li>• Discrete Filter</li> <li>• Discrete FIR Filter</li> <li>• Discrete State-Space</li> <li>• Discrete Transfer Fcn</li> <li>• Discrete Zero-Pole</li> <li>• Transfer Fcn First Order</li> <li>• Transfer Fcn Lead or Lag</li> <li>• Transfer Fcn Real Zero</li> <li>• Discrete Derivative</li> <li>• Discrete Transfer Fcn (with initial outputs)</li> <li>• Discrete Transfer Fcn (with initial states)</li> <li>• Discrete Zero-Pole (with initial outputs)</li> <li>• Discrete Zero-Pole (with initial states)</li> </ul>	In the model or subsystem, consider removing the time-dependent blocks.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- IEC 61508-3, Table A.3 (3) - Language subset, Table A.4 (3) - Defensive programming
- IEC 62304, 5.5.3 - Software Unit acceptance criteria
- ISO 26262-6, Table 1 (1b) - Use of language subsets, Table 1 (1d) - Use of defensive implementation techniques
- EN 50128 - Table A.4 (11) - Language Subset, Table A.3 (1) - Defensive Programming
- MISRA C:2012, Rule 14.2
- MISRA C:2012, Rule 16.4
- MISRA C:2012, Dir 4.1
- “hisl\_0006: Usage of While Iterator blocks” (Simulink)
- “hisl\_0007: Usage of For Iterator or While Iterator subsystems” (Simulink)
- “hisl\_0008: Usage of For Iterator Blocks” (Simulink)
- “hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks” (Simulink)

## Display configuration management data

**Check ID:** `mathworks.iec61508.MdlVersionInfo`

Display model configuration and checksum information.

### Description

This informer check displays the following information for the current model:

- Model version number
- Model author
- Date

- Model checksum

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Could not retrieve model version and checksum information.	This summary is provided for your information. No action is required.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- IEC 61508-3, Table A.8 (5) - Software configuration management
- IEC 62304-8 - Software configuration management process
- ISO 26262-8, Clause 7 - Configuration management
- EN 50128, Table A.9 (5) - Software Configuration Management
- “How Simulink Helps You Manage Model Versions” (Simulink).
- Model Change Log in the Simulink Report Generator™ documentation
- `Simulink.BlockDiagram.getChecksum` in the Simulink documentation
- `Simulink.SubSystem.getChecksum` in the Simulink documentation

## MathWorks Automotive Advisory Board Checks

### In this section...

- "MathWorks Automotive Advisory Board Checks" on page 2-167
- "Check font formatting" on page 2-168
- "Check transition orientations in flow charts" on page 2-170
- "Check for nondefault block attributes" on page 2-171
- "Check usage of merge blocks" on page 2-172
- "Check signal line labels" on page 2-174
- "Check for propagated signal labels" on page 2-176
- "Check default transition placement in Stateflow charts" on page 2-177
- "Check return value assignments in Stateflow graphical functions" on page 2-178
- "Check entry formatting in State blocks in Stateflow charts" on page 2-179
- "Check usage of return values from Stateflow graphical functions" on page 2-180
- "Check for pointers in Stateflow charts" on page 2-181
- "Check for event broadcasts in Stateflow charts" on page 2-182
- "Check transition actions in Stateflow charts" on page 2-183
- "Check for MATLAB expressions in Stateflow charts" on page 2-184
- "Check for indexing in blocks" on page 2-185
- "Check file names" on page 2-187
- "Check folder names" on page 2-188
- "Check for prohibited blocks in discrete controllers" on page 2-189
- "Check for prohibited sink blocks" on page 2-190
- "Check positioning and configuration of ports" on page 2-192
- "Check for matching port and signal names" on page 2-193
- "Check whether block names appear below blocks" on page 2-194
- "Check for mixing basic blocks and subsystems" on page 2-195
- "Check for unconnected ports and signal lines" on page 2-196
- "Check position of Trigger and Enable blocks" on page 2-197
- "Check usage of tunable parameters in blocks" on page 2-198

**In this section...**

- "Check Stateflow data objects with local scope" on page 2-200
- "Check for Strong Data Typing with Simulink I/O" on page 2-201
- "Check usage of exclusive and default states in state machines" on page 2-202
- "Check Implement logic signals as Boolean data (vs. double)" on page 2-204
- "Check model diagnostic parameters" on page 2-204
- "Check the display attributes of block names" on page 2-207
- "Check display for port blocks" on page 2-209
- "Check subsystem names" on page 2-210
- "Check port block names" on page 2-211
- "Check character usage in signal labels" on page 2-213
- "Check Simulink bus signal names" on page 2-214
- "Check character usage in block names" on page 2-216
- "Check Trigger and Enable block names" on page 2-219
- "Check for Simulink diagrams using nonstandard display attributes" on page 2-220
- "Check MATLAB code for global variables" on page 2-222
- "Check visibility of block port names" on page 2-223
- "Check orientation of Subsystem blocks" on page 2-224
- "Check usage of Relational Operator blocks" on page 2-225
- "Check usage of Switch blocks" on page 2-226
- "Check usage of buses and Mux blocks" on page 2-227
- "Check for bitwise operations in Stateflow charts" on page 2-228
- "Check fundamental logical and numerical operations" on page 2-230
- "Check logical expressions in If blocks" on page 2-232
- "Check for comparison operations in Stateflow charts" on page 2-235
- "Check usage of restricted variable names" on page 2-236
- "Check unused ports in Variant Subsystems" on page 2-237
- "Check usage of character vector inside MATLAB Function block" on page 2-238
- "Check usage of recommended patterns for Switch/Case statements" on page 2-239
- "Check use of default variants" on page 2-240

**In this section...**

- “Check use of single variable variant conditionals” on page 2-242
- “Check nested states in Stateflow charts” on page 2-244
- “Check use of Simulink in Stateflow charts” on page 2-245
- “Check number of Stateflow states per container” on page 2-246
- “Check usage of unary minus operations in Stateflow charts” on page 2-247
- “Check usage of floating-point expressions in Stateflow charts” on page 2-248
- “Check input and output settings of MATLAB Functions” on page 2-249
- “Check MATLAB Function metrics” on page 2-250
- “Check for names of Stateflow ports and associated signals” on page 2-252
- “Check scope of From and Goto blocks” on page 2-253
- “Check the number of function calls in MATLAB Function blocks” on page 2-254
- “Check usage of Goto and From blocks between Subsystems” on page 2-254
- “Check reuse of variables within a Stateflow scope” on page 2-255
- “Check for Stateflow transition appearance” on page 2-256
- “Check usage of enumerated values” on page 2-258
- “Check Simulink signal appearance” on page 2-258
- “Check usage of non-compliant blocks” on page 2-260

## MathWorks Automotive Advisory Board Checks

MathWorks Automotive Advisory Board (MAAB) and checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run the MAAB checks.

### See Also

- “Run Model Advisor Checks and Review Results”
- “MAAB Control Algorithm Modeling” (Simulink) guidelines
- *The Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow* MAAB guidelines on the MathWorks website

### Check font formatting

**Check ID:** `mathworks.maab.db_0043`

Check for difference in font and font sizes.

#### Description

With the exception of free text annotations within a model, text elements, such as block names, block annotations, and signal labels, must have the same font style and font size. Select a font style and font size that is legible and portable (convertible between platforms), such as Arial or Times New Roman 12 point. To specify font rules for a Simulink session, from the Simulink editor select **Diagram > Format > Font Styles for Model**.

Available with Simulink Check.

#### Input Parameters

##### Font Name

Apply the specified font to all text elements. When you specify **Common** (default), the check identifies different fonts used in your model. Although you can specify other fonts, the fonts available from the drop-down list are **Arial**, **Courier New**, **Georgia**, **Times New Roman**, **Arial Black**, and **Verdana**.

##### Font Size

Apply the specified font size to all text elements. When you specify **Common** (default), the check identifies different font sizes used in your model. Although you can specify other font sizes, the font sizes available from the drop-down list are **6**, **8**, **9**, **10**, **12**, **14**, **16**.

##### Font Style

Apply the specified font style to all text elements. When you specify **Common** (default), the check identifies different font styles used in your model. The font styles available from the drop-down list are **normal**, **bold**, **italic**, and **bold italic**.



## Results and Recommended Actions

Condition	Recommended Action
The fonts or font sizes for text elements in the model are not consistent or portable.	Specify values for the font parameters and in the right pane of the Model Advisor, click <b>Modify all Fonts</b> , or manually change the fonts and font sizes of text elements in the model so they are consistent and portable.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag Stateflow state labels and box name font and font style settings. (Sub ID: c)
  - The check does not flag Stateflow state labels and box name font size settings. (Sub ID: d)
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

## Action Results

In the right pane of the Model Advisor, clicking **Modify all Fonts** changes the font and font size of all text elements in the model according to the values you specify in the input parameters.

For the input parameters, if you specify **Common**, clicking **Modify all Fonts** changes the font and font sizes of all text elements in the model to the most commonly used fonts, font sizes, or font styles.

## See Also

- MAAB guideline, Version 3.0: db\_0043: Simulink font and font size in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0043: Model font and font size

## Check transition orientations in flow charts

**Check ID:** `mathworks.maab.db_0132`

Check transition orientations in flow charts.

### Description

The following rules apply to transitions in flow charts:

- Draw transition conditions horizontally.
- Draw transitions with a condition action vertically.
- Junctions in flow charts should have a default exit transition.
- Transitions in flow charts should not combine condition and action.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The model includes a transition with a condition that is not drawn horizontally or a transition action that is not drawn vertically.	Modify the model.
Junction does not have a default exit transition	Add a default exit transition to the junction.
Transition has condition and action	Split up condition and action into separate transitions

### Capabilities and Limitations

- MAAB guideline, Version 3.0 limitation: Although `db_0132`: Transitions in flow charts has an exception for loop constructs, the check does flag flow charts containing loop constructs if the transition violates the orientation rule.
- JMAAB guideline, Version 5.1 limitation: The check only flags flow charts containing loop constructs if the transition violates the orientation rule.

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **See Also**

- MAAB guideline, Version 3.0: db\_0132: Transitions in flow charts in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0132: Transitions in Flow Charts

## **Check for nondefault block attributes**

**Check ID:** `mathworks.maab.db_0140`

Identify blocks that use nondefault block parameter values that are not displayed in the model diagram.

### **Description**

Model diagrams should display block parameters that have values other than default values. One way of displaying this information is by using the **Block Annotation** tab in the Block Properties dialog box. To automatically fix warnings associated with this check, see “Automatically Fix Display of Nondefault Block Parameters”.

To customize the list of nondefault block parameters that are flagged by the check, see “Customize Model Advisor Check for Nondefault Block Attributes”.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Block parameters that have values other than default values, and the values are not in the model display.	In the Block Properties dialog box, use the <b>Block Annotation</b> tab to add block parameter annotations.

## Capabilities and Limitations

- Only customizable for block parameters in `IntrinsicDialogParameters`. See “Common Block Properties” (Simulink)
- JMAAB guideline, Version 5.1 limitation: The check flags masked blocks that display parameter information but do not use block annotations. JMAAB 5.1 guidelines allow masked blocks to display parameter information.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialog boxes.
- Allows exclusions of blocks and charts.

## Tip

If you use the `add_block` function with `'built-in/blocktype'` as a source block path name for Simulink built-in blocks, some default parameter values of some blocks are different from the defaults that you get if you added those blocks interactively by using Simulink.

## See Also

- MAAB guideline, Version 3.0: `db_0140`: Display of basic block parameters.
- JMAAB guideline, Version 5.1: `db_0140`: Display of block parameters
- For a list of block parameter default values, see “Block-Specific Parameters” (Simulink).
- `add_block`.

## Check usage of merge blocks

**Check ID:** `mathworks.maab.na_0032`

Check usage of Merge blocks.

### Description

Identifies instances where buses are not identical in the Merge block, including number of elements, element names, element order, element data type, and element size.

---

**Note** These Merge block constraints are not covered by this check. They are flagged via compile-time errors.

- Signals and buses entering the Merge block cannot branch off to other blocks.
  - Buses must be either all virtual or all nonvirtual.
- 

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Block parameter <b>Allow unequal port widths</b> is selected.	Clear block parameter <b>Allow unequal port widths</b> /
Elements for bus input to a Merge block are different.	Bus input to a Merge block must be equivalent, including the same hierarchy with identical names and attributes for all elements. Consider updating the elements so that all buses are identical.
Data types for input signals to a Merge block are different.	The Merge block accepts real or complex signals of any data type that Simulink supports, including fixed-point and enumerated data types. Inputs must be of the same data type and numeric type. Consider updating the signals so data types are the same.
Dimensions of input signals to a Merge block are different.	The Merge block accepts only inputs of equal dimensions. Consider updating the signals inputs so they have the same dimensions.

Condition	Recommended Action
Unequal number of inputs on the Merge block.	Connect a Merge block to at least two input signals and verify that input signals have the same sample time. Consider making the number of inputs signal and input ports the same.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0032: Use of merge blocks in the Simulink documentation.

## Check signal line labels

**Check ID:** mathworks.maab.na\_0008

Check the labeling on signal lines.

### Description

Use a label to identify:

- Signals originating from the following blocks (the block icon exception noted below applies to all blocks listed, except Inport, Bus Selector, Demux, and Selector):
  - Bus Selector block (tool forces labeling)
  - Chart block (Stateflow)
  - Constant block
  - Data Store Read block
  - Demux block
  - From block

- Inport block
- Selector block
- Subsystem block

---

**Block Icon Exception** If a signal label is visible in the display of the icon for the originating block, you do not have to display a label for the connected signal unless the signal label is required elsewhere due to a rule for signal destinations.

---

- Signals connected to one of the following destination blocks (directly or indirectly with a basic block that performs an operation that is not transformative):
  - Bus Selector block (tool forces labeling)
  - Chart block (Stateflow)
  - Data Store Write block
  - Goto block
  - Mux block
  - Output block
  - Subsystem block
- Any signal of interest.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Signals coming from Bus Selector, Chart, Constant, Data Store Read, Demux, From, Inport, or Selector blocks are not labeled.	Label the signal.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: na\_0008: Display of labels on signals in the Simulink documentation.
- “Signal Names and Labels” (Simulink).

## Check for propagated signal labels

**Check ID:** `mathworks.maab.na_0009`

Check for propagated labels on signal lines.

### Description

You should propagate a signal label from its source rather than enter the signal label explicitly (manually) if the signal originates from:

- An Inport block in a nested subsystem. However, if the nested subsystem is a library subsystem, you can explicitly label the signal coming from the Inport block to accommodate reuse of the library block.
- A basic block that performs a nontransformative operation.
- A Subsystem or Stateflow Chart block. However, if the connection originates from the output of an instance of the library block, you can explicitly label the signal to accommodate reuse of the library block.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model includes signal labels that were entered explicitly, but should be propagated.	Use the open angle bracket (<) character to mark signal labels that should be propagated and remove the labels that were entered explicitly.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.



- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: na\_0009: Entry versus propagation of signal labels in the Simulink documentation.
- “Signal Names and Labels” (Simulink).

## Check default transition placement in Stateflow charts

**Check ID:** `mathworks.maab.jc_0531`

Check default transition placement in Stateflow charts.

### Description

In a Stateflow chart, you should connect the default transition at the top of the state and place the destination state of the default transition above other states in the hierarchy. There should be only one default transition.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The default transition for a Stateflow chart is not connected at the top of the state.	Move the default transition to the top of the Stateflow chart.
The destination state of a Stateflow chart default transition is lower than other states in the same hierarchy.	Adjust the position of the default transition destination state so that the state is above other states in the same hierarchy.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not detect default transitions that exceed the state boundary.

- The check does not detect missing unconditional default transitions.
- The check does not detect the horizontal placement (left position) of the default transition.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0531: Placement of the default transition in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0531: Default transition
- “Syntax for States and Transitions” (Stateflow)

## Check return value assignments in Stateflow graphical functions

**Check ID:** `mathworks.maab.jc_0511`

Identify graphical functions with multiple assignments of return values in Stateflow charts.

### Description

The return value from a Stateflow graphical function must be set in only one place.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The return value from a Stateflow graphical function is assigned in multiple places.	Modify the specified graphical function so that its return value is set in one place.

## Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## See Also

- MAAB guideline, Version 3.0: jc\_0511: Setting the return value from a graphical function in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0511: Setting the return value from a graphical function
- “Reusable Functions in Charts” (Stateflow).

## Check entry formatting in State blocks in Stateflow charts

**Check ID:** mathworks.maab.jc\_0501

Identify missing line breaks between entry action (en), during action (du), and exit action (ex) entries in states. Identify missing line breaks after semicolons (;) in statements.

### Description

Start a new line after the entry, during, and exit entries, and after the completion of a statement “;”.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
An entry (en) is not on a new line.	Add a new line after the entry.
A during (du) is not on a new line.	Add a new line after the during.
An exit (ex) is not on a new line.	Add a new line after the exit.
Multiple statements found on one line.	Add a new line after each statement.

**Capabilities and Limitations**

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**Edit-Time Checking**

This check is supported by edit-time checking.

**See Also**

MAAB guideline, Version 3.0: jc\_0501: Format of entries in a State block in the Simulink documentation.

**Check usage of return values from Stateflow graphical functions**

**Check ID:** `mathworks.maab.jc_0521`

Identify calls to graphical functions in conditional expressions.

**Description**

Do not use the return value of a graphical function in a comparison operation.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Conditional expressions contain calls to graphical functions.	Assign return values of graphical functions to intermediate variables. Use these intermediate variables in the specified conditional expressions.

**Capabilities and Limitations**

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- MAAB guideline, Version 3.0: jc\_0521: Use of the return value from graphical functions in the Simulink documentation.
- “Reusable Functions in Charts” (Stateflow).
- “Reuse Logic Patterns by Defining Graphical Functions” (Stateflow).

**Check for pointers in Stateflow charts**

**Check ID:** mathworks.maab.jm\_0011

Identify pointer operations on custom code variables.

**Description**

Pointers to custom code variables are not allowed.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
Custom code variables use pointer operations.	Modify the specified chart to remove the dependency on pointer operations.

**Capabilities and Limitations**

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jm\_0011: Pointers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 5.1: jm\_0011: Pointers in Stateflow

## Check for event broadcasts in Stateflow charts

**Check ID:** `mathworks.maab.jm_0012`

Identify undirected event broadcasts that might cause recursion during simulation and generate inefficient code.

### Description

Event broadcasts in Stateflow charts must be directed.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Event broadcasts are undirected.	Rearchitect the diagram to use directed event broadcasting. Use the send syntax or qualified event names to direct the event to a particular state. Use multiple send statements to direct an event to more than one state.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check cannot be configured to flag event used other than a Stateflow chart output. (Sub ID: a1)
- Runs on library models.

- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jm\_0012: Event broadcasts in the Simulink documentation.
- JMAAB guideline, Version 5.1: jm\_0012: Usage restrictions of events and broadcasting events
- “Broadcast Local Events to Synchronize Parallel States” (Stateflow).

## Check transition actions in Stateflow charts

**Check ID:** `mathworks.maab.db_0151`

Identify missing line breaks between transition actions.

### Description

For readability, start each transition action on a new line.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Multiple transition actions are on a single line.	Verify that each transition action begins on a new line.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**Edit-Time Checking**

This check is supported by edit-time checking.

**See Also**

- MAAB guideline, Version 3.0: db\_0151: State machine patterns for transition actions in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0151: State machine patterns for transition actions.
- “Syntax for States and Transitions” (Stateflow)

**Check for MATLAB expressions in Stateflow charts**

**Check ID:** `mathworks.maab.db_0127`

Identify Stateflow objects that use MATLAB expressions that are not suitable for code generation.

**Description**

Do not use MATLAB functions, instructions, and operators in Stateflow objects.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Stateflow objects use MATLAB expressions.	Replace MATLAB expressions in Stateflow objects.

**Capabilities and Limitations**

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.



**Edit-Time Checking**

This check is supported by edit-time checking.

**See Also**

- MAAB guideline, Version 3.0: db\_0127: MATLAB commands in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0127: Limitation on MATLAB commands in Stateflow blocks
- “Access MATLAB Functions and Workspace Data in C Charts” (Stateflow).

**Check for indexing in blocks**

**Check ID:** `mathworks.maab.db_0112`

Check that blocks use consistent vector indexing.

**Description**

Check that blocks use consistent vector indexing. When possible, use zero-based indexing to improve code efficiency.

Available with Simulink Check.

This check requires a Stateflow license.

The check verifies consistent indexing for the following objects:

Object	Indexing
<ul style="list-style-type: none"> <li>• Assignment block</li> <li>• For Iterator block</li> <li>• Find block</li> <li>• Multiport Switch block</li> <li>• Selector block</li> </ul>	<ul style="list-style-type: none"> <li>• Zero-based indexing ([0, 1, 2, ...])</li> <li>• One-based indexing ([1, 2, 3,...])</li> </ul>
<ul style="list-style-type: none"> <li>• Stateflow charts with C action language</li> </ul>	Zero-based indexing ([0, 1, 2, ...])

Object	Indexing
<ul style="list-style-type: none"> <li>• MATLAB Function block</li> <li>• Fcn block</li> <li>• MATLAB System blocks</li> <li>• Truth tables</li> <li>• State transition tables</li> <li>• Stateflow charts with MATLAB action language</li> <li>• MATLAB functions inside Stateflow charts</li> </ul>	One-based indexing ([1, 2, 3,...])

**Results and Recommended Actions**

Condition	Recommended Action
Objects in your model use one-based indexing, but can be configured for zero-based indexing.	Configure objects for zero-based indexing.
Objects in your model use inconsistent indexing.	If possible, configure objects for zero-based indexing. If your model contains objects that cannot be configured for zero-based indexing, configure objects for one-based indexing.

**Capabilities and Limitations**

- JMAAB guideline, Version 5.1 limitation: This check cannot be configured to support one-based indexing. (Sub ID: a2)
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- MAAB guideline, Version 3.0: db\_0112: Indexing in the Simulink documentation.

- JMAAB guideline, Version 5.1: db\_0112: Usage of index

## Check file names

**Check ID:** mathworks.maab.ar\_0001

Checks the names of all files residing in the same folder as the model

### Description

A file name conforms to constraints.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The file name contains illegal characters.	Rename the file. Allowed characters are a-z, A-Z, 0-9, and underscore (_).
The file name starts with a number.	Rename the file.
The file name starts with an underscore ("_").	Rename the file.
The file name ends with an underscore ("_").	Rename the file.
The file extension contains one (or more) underscores.	Change the file extension.
The file name has consecutive underscores.	Rename the file.
The file name contains more than one dot (".").	Rename the file.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag conflicts with MATLAB reserved keywords. (Sub ID: f)
  - The check does not flag identical file names on the MATLAB path. (Sub ID: g)
- MAAB guideline, Version 3.0 limitation: The check does not flag conflicts with C++ keywords.

- Runs on library models.
- Does not allow exclusions of blocks or charts.

**See Also**

- MAAB guideline, Version 3.0: ar\_0001: Filenames in the Simulink documentation.
- JMAAB guideline, Version 5.1: ar\_0001: Usable characters for file names

**Check folder names****Check ID:** `mathworks.maab.ar_0002`

Checks model directory and subdirectory names for invalid characters.

**Description**

A directory name conforms to constraints.

Available with Simulink Check.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
The directory name contains illegal characters.	Rename the directory. Allowed characters are a-z, A-Z, 0-9, and underscore (_).
The directory name starts with a number.	Rename the directory.
The directory name starts with an underscore ("_").	Rename the directory.
The directory name ends with an underscore ("_").	Rename the directory.
The directory name has consecutive underscores.	Rename the directory.

**Capabilities and Limitations**

- JMAAB guideline, Version 5.1 limitation: The check does not flag conflicts with MATLAB reserved keywords. (Sub ID: f)
- Runs on library models.

- Does not allow exclusions of blocks or charts.
- Analyzes the full path of the model.
- Analyzes subdirectories in the same directory as the model.

### See Also

- MAAB guideline, Version 3.0: ar\_0002: Directory names in the Simulink documentation.
- JMAAB guideline, Version 5.1: ar\_0002: Usable characters for folder names

## Check for prohibited blocks in discrete controllers

**Check ID:** `mathworks.maab.jm_0001`

Check for prohibited blocks in discrete controllers.

### Description

The check identifies continuous blocks in discrete controller models.

Available with Simulink Check.

### Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check for prohibited blocks in discrete controllers**.
- 2 In the **Input Parameters** pane, to:
  - Prohibit the blocks as specified in MAAB 3.0, from **Standard**, select **MAAB 3.0**. The **Block type list** table provides the blocks that MAAB 3.0 prohibits inside controllers.
  - To specify blocks to either allow or prohibit, from **Standard**, select **Custom**. In **Treat blocktype list as**, select **Allowed** or **Prohibited**. In the **Block type list** table, you can add or remove blocks.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

## Results and Recommended Actions

Condition	Recommended Action
Continuous blocks — Derivative, Integrator, State-Space, Transfer Fcn, Transfer Delay, Variable Time Delay, Variable Transport Delay, and Zero-Pole — are not permitted in models representing discrete controllers.	Replace continuous blocks with the equivalent blocks discretized in the s-domain. Use the Discretizing library, as described in “Discretize Blocks from the Simulink Model” (Simulink).

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- MAAB guideline, Version 3.0: jm\_0001: Prohibited Simulink standard blocks inside controllers in the Simulink documentation.
- JMAAB guideline, Version 5.1: jm\_0001: Prohibited Simulink standard blocks inside controllers.
- “Overview of the Model Advisor Configuration Editor”

## Check for prohibited sink blocks

**Check ID:** mathworks.maab.hd\_0001

Check for prohibited Simulink sink blocks.

### Description

You must design controller models from discrete blocks. Sink blocks, such as the Scope block, are not allowed in controller models.

Available with Simulink Check.

## Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check for prohibited sink blocks**.
- 2 In the **Input Parameters** pane, to:
  - Prohibit the blocks as specified by MAAB 3.0, from **Standard**, select MAAB 3.0. The **Block type list** table provides the sink blocks that MAAB 3.0 prohibits.
  - To specify blocks to either allow or prohibit, from **Standard**, select Custom. In **Treat blocktype list as**, select Allowed or Prohibited. In the **Block type list** table, you can add or remove blocks.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

## Results and Recommended Actions

Condition	Recommended Action
Sink blocks are not permitted in discrete controllers.	Remove sink blocks from the model.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- MAAB guideline, Version 3.0: hd\_0001: Prohibited Simulink sinks in the Simulink documentation.

- JMAAB guideline, Version 5.1: hd\_0001: Prohibited Simulink sinks.
- “Overview of the Model Advisor Configuration Editor”

## Check positioning and configuration of ports

**Check ID:** `mathworks.maab.db_0042`

Check whether the model contains ports with invalid position and configuration.

### Description

In models, ports must comply with the following rules:

- Place Inport blocks on the left side of the diagram. It is acceptable to move the Inport block to the right only to prevent signal crossings.
- Place Outport blocks on the right side of the diagram. It is acceptable to move the Outport block to the left only to prevent signal crossings.
- Avoid using duplicate Inport blocks at the subsystem level if possible.
- Do not use duplicate Inport blocks at the root level.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Inport blocks are too far to the right and result in left-flowing signals.	Move the specified Inport blocks to the left.
Outport blocks are too far to the left and result in left-flowing signals.	Move the specified Output blocks to the right.
Ports do not have the default orientation.	Modify the model diagram such that signal lines for output ports enter the side of the block and signal lines for input ports exit the right side of the block.



Condition	Recommended Action
Ports are duplicate Inport blocks.	<ul style="list-style-type: none"> <li>• If the duplicate Inport blocks are in a subsystem, remove them where possible.</li> <li>• If the duplicate Inport blocks are at the root level, remove them.</li> </ul>

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag when Inports are not placed on the left side of the diagram. (Sub ID: a)
  - The check does not flag when Outports are not placed on the right side of the diagram. (Sub ID: b)
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Does not analyze signal crossings

### See Also

- MAAB guideline, Version 3.0: db\_0042: Port block in Simulink models in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0042: Usage of Inport and Outport blocks

## Check for matching port and signal names

**Check ID:** mathworks.maab.jm\_0010

Check for mismatches between names of ports and corresponding signals.

### Description

Use matching names for ports and their corresponding signals.

Available with Simulink Check.

### Prerequisite

Prerequisite MAAB guidelines, Version 3.0, for this check are:

- db\_0042: Port block in Simulink models
- na\_0005: Port block name visibility in Simulink models

### Results and Recommended Actions

Condition	Recommended Action
Ports have names that differ from their corresponding signals.	Change the port name or the signal name to match the name for the signal.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jm\_0010: Port block names in Simulink models in the Simulink documentation.

## Check whether block names appear below blocks

**Check ID:** `mathworks.maab.db_0142`

Check whether block names appear below blocks.

### Description

If shown, the name of the block should appear below the block.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Blocks have names that do not appear below the blocks.	Set the name of the block to appear below the blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- MAAB guideline, Version 3.0: db\_0142: Position of block names in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0142: Position of block names.

## Check for mixing basic blocks and subsystems

**Check ID:** `mathworks.maab.db_0143`

Check for systems that mix primitive blocks and subsystems.

### Description

You must design each level of a model with building blocks of the same type, for example, only subsystems or only primitive (basic) blocks. If you mask your subsystem and set MaskType to a nonempty string, the Model Advisor treats the subsystem as a basic block.

Available with Simulink Check.

### Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check for mixing basic blocks and subsystems**.
- 2 In the **Input Parameters** pane, to:
  - Allow the blocks specified by MAAB 3.0, from **Standard**, select MAAB 3.0. The **Block type list** table provides the blocks that MAAB 3.0 allows at any model level.
  - To specify blocks to either allow or prohibit, from **Standard**, select Custom. In **Treat blocktype list as**, select Allowed or Prohibited. In the **Block type list** table, you can add or remove blocks.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

### Results and Recommended Actions

Condition	Recommended Action
A level in the model includes subsystem blocks and primitive blocks.	Move nonvirtual blocks into the subsystem.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0143: Similar block types on the model levels in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0143: Usage block types in model hierarchy
- “Overview of the Model Advisor Configuration Editor”

## Check for unconnected ports and signal lines

**Check ID:** `mathworks.maab.db_0081`

Check whether model has unconnected input ports, output ports, or signal lines.

### Description

Unconnected inputs should be connected to ground blocks. Unconnected outputs should be connected to terminator blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Blocks have unconnected inputs or outputs.	Connect unconnected lines to blocks specified by the design or to Ground or Terminator blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0081: Unconnected signals, block inputs and block outputs in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0081: Unconnected signals / block

## Check position of Trigger and Enable blocks

**Check ID:** mathworks.maab.db\_0146

Check the position of Trigger and Enable blocks.

### Description

Locate blocks that define subsystems as conditional or iterative at the top of the subsystem diagram.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Trigger, Enable, and Action Port blocks are not at the top of the subsystem diagram.	Move the Trigger, Enable, and Action Port blocks to the top of the subsystem diagram.
For Each, For Iterator, and While Iterator blocks are not in the same location on the subsystem diagram.	Move the For Each, For Iterator, and While Iterator blocks so they are at a uniform location on the subsystem diagram.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not verify that For Each or For Iterator blocks are uniformly located.
- Runs on library models.
- Analyzes content of library linked blocks.
- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0146: Triggered, enabled, conditional Subsystems in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0146: Block layout in conditional subsystems

## Check usage of tunable parameters in blocks

**Check ID:** `mathworks.maab.db_0110`

Check whether tunable parameters specify expressions, data type conversions, or indexing operations.

### Description

To make a parameter tunable, you must enter the basic block without the use of MATLAB calculations or scripting. For example, omit:

- Expressions
- Data type conversions

- Selections of rows or columns

Supported blocks include:

- Backlash
- Bias
- Combinatorial Logic
- Constant
- Dead Zone
- Derivative
- Discrete-Time Integrator
- Gain
- Hit Crossing
- Initial Condition (IC)
- Integrator
- n-D Lookup Table
- Magnitude-Angle to Complex
- Memory
- Permute Dimensions
- Quantizer
- Rate Limiter
- Rate Transition
- Real-Imag to Complex
- Relay
- Saturation
- Sine
- State-Space
- Switch
- Transport Delay
- Unit Delay
- Transport Delay

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Blocks have a tunable parameter that specifies an expression, data type conversion, or indexing operation.	In each case, move the calculation outside of the block, for example, by performing the calculation with a series of Simulink blocks, or precompute the value as a new variable.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not evaluate mask parameters.
- Allows exclusions of blocks and charts.

## See Also

- MAAB guideline, Version 3.0: db\_0110: Tunable parameters in basic blocks in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0110: Guidelines for block parameters

## Check Stateflow data objects with local scope

**Check ID:** `mathworks.maab.db_0125`

Check whether Stateflow data objects with local scope are defined at the chart level or below.

### Description

This check flags Stateflow data whose local scope is not defined at the Chart level or below, regardless of whether the data is used or not.

You must define local data of a Stateflow block on the chart level or below in the object hierarchy. You cannot define local variables on the machine level; however, parameters and constants are allowed at the machine level.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
Local data is not defined in the Stateflow hierarchy at the chart level or below.	Define local data at the chart level or below.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag Constant local data defined at the machine level. (Sub ID: b)
  - The check does not flag duplicate data definition and usage within a parent-child relationships. (Sub ID: c)
  - The check does not flag Parameter local data defined at the machine level. (Sub ID: d)
  - The check does not detect if local data has the same name within charts or states that have parent-child relationships.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

## See Also

- MAAB guideline, Version 3.0: db\_0125: Scope of internal signals and local auxiliary variables in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0125: Stateflow local data

## Check for Strong Data Typing with Simulink I/O

**Check ID:** `mathworks.maab.db_0122`

Check whether labeled Stateflow and Simulink input and output signals are strongly typed.

### Description

Strong data typing between Stateflow and Simulink input and output signals is required.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
A Stateflow chart does not use strong data typing with Simulink.	Select the <b>Use Strong Data Typing with Simulink I/O</b> check box for the specified block.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks and charts.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0122: Stateflow and Simulink interface signals and parameters in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0122: Stateflow and Simulink interface signals and parameters
- “Syntax for States and Transitions” (Stateflow)

## Check usage of exclusive and default states in state machines

**Check ID:** `mathworks.maab.db_0137`

Check states in state machines.

### Description

In state machines:

- There must be at least two exclusive states.
- A state cannot have only one substate.

- The initial state of a hierarchical level with exclusive states is clearly defined by a default transition.

Available with Simulink Check.

This check requires a Stateflow license.

### Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is db\_0149: Flow chart patterns for condition actions.

### Results and Recommended Actions

Condition	Recommended Action
Chart has only one exclusive (OR) state.	Make the state a parallel state, or add another exclusive (OR) state.
Chart does not have a default state defined.	Define a default state.
Chart has multiple default states defined.	Define only one default state. Make the others nondefault.
State has only one exclusive (OR) substate.	Make the state a parallel state, add another exclusive (OR) state, or replace the state with a flow chart.
State does not have a default substate defined.	Define a default substate.
State has multiple default substates defined.	Define only one default substate, make the others nondefault.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

MAAB guideline, Version 3.0: db\_0137: States in state machines in the Simulink documentation.

## Check Implement logic signals as Boolean data (vs. double)

**Check ID:** mathworks.maab.jc\_0011

Check the optimization parameter for Boolean data types.

### Description

Optimization for Boolean data types is required

Available with Simulink Check.

### Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is na\_0002: Appropriate implementation of fundamental logical and numerical operations.

### Results and Recommended Actions

Condition	Recommended Action
Configuration setting for <b>Implement logic signals as boolean data (vs. double)</b> is not set.	Select the <b>Implement logic signals as boolean data (vs. double)</b> check box in the Configuration Parameters dialog box.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0011: Optimization parameters for Boolean data types in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0011: Optimization parameters for Boolean data types.

## Check model diagnostic parameters

**Check ID:** mathworks.maab.jc\_0021

Check the model diagnostics configuration parameter settings.

## Description

You should enable the following diagnostics:

**Algebraic loop**

**Minimize algebraic loop**

**Inf or NaN block output**

**Duplicate data store names**

**Unconnected block input ports**

**Unconnected block output ports**

**Unconnected line**

**Unspecified bus object at root Output block**

**Element name mismatch**

**Invalid function-call connection**

Diagnostics not listed in the Results and Recommended Actions section below can be set to any value.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<b>Algebraic loop</b> is set to none.	Set <b>Algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box to error or warning. Otherwise, Simulink might attempt to automatically break the algebraic loops, which can impact the execution order of the blocks.
<b>Minimize algebraic loop</b> is set to none.	Set <b>Minimize algebraic loop</b> on the <b>Diagnostics &gt; Solver</b> pane in the Configuration Parameters dialog box to error or warning. Otherwise, Simulink might attempt to automatically break the algebraic loops for reference models and atomic subsystems, which can impact the execution order for those models or subsystems.

Condition	Recommended Action
<b>Inf or NaN block output</b> is set to none, which can result in numerical exceptions in the generated code.	Set <b>Inf or NaN block output</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Duplicate data store names</b> is set to none, which can result in nonunique variable naming in the generated code.	Set <b>Duplicate data store names</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Unconnected block input ports</b> is set to none, which prevents code generation.	Set <b>Unconnected block input ports</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Unconnected block output ports</b> is set to none, which can lead to dead code.	Set <b>Unconnected block output ports</b> on the <b>Diagnostics &gt; Data Validity &gt; Signals</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Unconnected line</b> is set to none, which prevents code generation.	Set <b>Unconnected line</b> on the <b>Diagnostics &gt; Connectivity &gt; Signals</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Unspecified bus object at root Output block</b> is set to none, which can lead to an unspecified interface if the model is referenced from another model.	Set <b>Unspecified bus object at root Output block</b> on the <b>Diagnostics &gt; Connectivity &gt; Buses</b> pane in the Configuration Parameters dialog box to error or warning.
<b>Element name mismatch</b> is set to none, which can lead to an unintended interface in the generated code.	Set <b>Element name mismatch</b> on the <b>Diagnostics &gt; Connectivity &gt; Buses</b> pane in the Configuration Parameters dialog box to error or warning.

**Capabilities and Limitations**

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

**See Also**

- MAAB guideline, Version 3.0: jc\_0021: Model diagnostic settings in the Simulink documentation.

**Check the display attributes of block names**

**Check ID:** mathworks.maab.jc\_0061

Check the display attributes of subsystem and block names.

**Description**

When the subsystem and block names provide descriptive information, display the names. If the block function is known from its appearance, do not display the name. Blocks with names that are obvious from the block appearance:

- From
- Goto
- Ground
- Logic
- MinMax
- ModelReference
- MultiPortSwitch
- Product
- Relational Operator
- Saturate
- Switch
- Terminator
- Trigonometry
- Unit Delay
- Sum
- Compare To Constant
- Compare To Zero

Available with Simulink Check.

### Input Parameters

To specify the custom blocks and masks to include in this check, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check the display attributes of block names**. In the **Input Parameters** pane, select Custom.
- 2 Use the **Block type list** table to include or exclude blocks and masks from the check.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
Name is displayed and obvious from the block appearance.	Hide the block name by selecting <b>Format &gt; Hide Automatic Block Name</b> .
Name is not descriptive. Specifically, the block name is: <ul style="list-style-type: none"> <li>• Not obvious from the block appearance.</li> <li>• The default name appended with an integer.</li> </ul>	Modify the block name to provide descriptive information, or hide the block name by selecting <b>Format &gt; Hide Automatic Block Name</b> .
Name is descriptive and not displayed. Descriptive names are: <ul style="list-style-type: none"> <li>• Provided for blocks that are not obvious from the block appearance.</li> <li>• Not a default name appended with an integer.</li> </ul>	Modify the blocks to show the block name by deselecting <b>Format &gt; Hide Automatic Block Name</b> .
Check does not evaluate my custom blocks and masks.	Use the Model Configuration Editor to specify your custom checks and blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.



- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0061: Display of block names in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0061: Display of block names

## Check display for port blocks

**Check ID:** mathworks.maab.jc\_0081

Check the **Icon display** setting for Inport and Outport blocks.

### Description

The **Icon display** setting is required.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The <b>Icon display</b> setting is not set.	Set the <b>Icon display</b> to Port number for the specified Inport and Outport blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.



- 2 Click **Apply**.
- 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
The subsystem names do not comply with the naming standard specified in the input parameters.	Update the subsystem names to comply with your own guidelines or the MAAB guidelines.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not flag conflicts with MATLAB reserved keywords. (Sub ID: f)
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### Tips

Use underscores to separate parts of a subsystem name instead of spaces.

### See Also

- MAAB guideline, Version 3.0: jc\_0201: Usable characters for Subsystem names in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0201: Usable characters for Subsystem names

## Check port block names

**Check ID:** mathworks.maab.jc\_0211

Check whether Inport and Outport block names include invalid characters.

### Description

The names of all Inport and Outport blocks are checked for invalid characters.

Available with Simulink Check.

### Input Parameters

To control the naming convention for blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check port block names**. In the **Input Parameter** pane:
  - Use **Naming standard** to select MAAB 3.0 or Custom. When you select MAAB 3.0, the check uses the regular expression  $([\^a-zA-Z_\theta-9])|(\^d)|(\^ )|(\_)|(\^_)|(\_\$)$  to verify that names:
    - Use these characters: a-z, A-Z,  $\theta$ -9, and the underscore ( $\_$ ).
    - Do not start with a number.
    - Do not use underscores at the beginning or end of a string.
    - Do not use more than one consecutive underscore.

When you select Custom, you can enter your own **Regular expression for prohibited names**. For example, if you want to allow more than one consecutive underscore, enter  $([\^a-zA-Z_\theta-9])|(\^d)|(\^ )|(\^_)|(\_\$)$ .

- 2 Click **Apply**.
- 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
The block names do not comply with the naming standard specified in the input parameters.	Update the block names to comply with your own guidelines or the MAAB guidelines.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not flag conflicts with MATLAB reserved keywords. (Sub ID: f)

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **Tips**

Use underscores to separate parts of a block name instead of spaces.

### **See Also**

- MAAB guideline, Version 3.0: jc\_0211: Usable characters for Inport blocks and Output blocks in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0211: Usable characters for Inport block and Output block

## **Check character usage in signal labels**

**Check ID:** mathworks.maab.jc\_0221

Check whether signal line names include invalid characters.

### **Description**

The names of all signal lines are checked for invalid characters.

Available with Simulink Check.

### **Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
The signal line name contains illegal characters.	Rename the signal line. Allowed characters include a-z, A-Z, 0-9, and underscore (_).
The signal line name starts with a number.	Rename the signal line.

<b>Condition</b>	<b>Recommended Action</b>
The signal line name starts with an underscore ("_").	Rename the signal line.
The signal line name ends with an underscore ("_").	Rename the signal line.
The signal line name has consecutive underscores.	Rename the signal line.
The signal line name has blank spaces.	Rename the signal line.
The signal line name has control characters.	Rename the signal line.

### **Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### **Tips**

Use underscores to separate parts of a signal line name instead of spaces.

### **See Also**

- MAAB guideline, Version 3.0: jc\_0221: Usable characters for signal line names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc\_0222: Usable characters for signal line and bus names.

## **Check Simulink bus signal names**

**Check ID:** `mathworks.maab.na_0030`

Checks Simulink bus signal names.

## Description

This check verifies that Simulink bus signal names comply with your own modeling guidelines or MAAB modeling guideline na\_0030. MAAB naming guidelines are as follows.

## Form

Simulink bus signal names:

- Should not start with a number
- Should not have blank spaces
- Carriage returns are not allowed

## Allowed Characters

Simulink bus signal names can contain characters:

a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 \_

## Underscores

Simulink bus signal names:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

Available with Simulink Check.

## Input Parameters

To specify the naming standard for Simulink bus signal names, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check Simulink bus signal names**. In the **Input Parameters** pane, enter:
  - MAAB 3.0 to use the MAAB naming standard. When you select MAAB 3.0, the check uses the regular expression  $([\text{^a-zA-Z}_0-9])|(\text{^d})|(\text{^ })|(\text{__})|(\text{^_})|(\text{_}$)$  to verify that names:

- Use these characters: a-z, A-Z, 0-9, and the underscore (\_).
  - Do not start with a number.
  - Do not use underscores at the beginning or end of a string.
  - Do not use more than one consecutive underscore.
  - Custom to use your own naming standard. When you select **Custom**, you can enter your own **Regular expression for prohibited names**. For example, if you want to allow more than one consecutive underscore, enter `(^a-zA-Z_0-9)]|^(\d)|(^ )|^_|(_$)`
- 2 Click **Apply**.
  - 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
The Simulink bus signal names do not comply with the naming standard specified in the input parameters.	Update the Simulink bus signal names to comply with your own or the MAAB guidelines.

### Capabilities and Limitations

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Allows syntax highlighting.

### See Also

- MAAB guideline, Version 3.0: na\_0030: Usable characters for Simulink Bus names in the Simulink documentation.
- JMAAB guideline, Version 4.0: jc\_0222: Usable characters for signal line and bus names.

## Check character usage in block names

**Check ID:** `mathworks.maab.jc_0231`

Check whether block names include invalid characters.



## Description

The check reports invalid characters in all block names, except:

- Inports and Outports
- Unmasked subsystems

MAAB guideline, Version 3.0, jc\_0231: Usable characters for block names does not apply to subsystem blocks.

Available with Simulink Check.

## Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is jc\_0201: Usable characters for Subsystem names.

## Input Parameters

To control the naming convention for blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check character usage in block names**. In the **Input Parameter** pane:

- Use **Naming standard** to select MAAB 3.0 or Custom. When you select MAAB 3.0, the check uses the regular expression  $([a-zA-Z_0-9\n\r ] | (^d) | (^ ))$  to verify that names:
  - Use these characters: a-z, A-Z, 0-9, underscore (`_`), and blank space.
  - Do not start with a number or blank space.
  - Do not have double byte characters.

When you select Custom, you can enter your own **Regular expression for prohibited names**. For example, if you do not want to allow underscores (`_`) in a block name, enter  $([a-zA-Z0-9\n\r ] | (^d) | (^ ))$ .

- 2 Click **Apply**.
- 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

## Results and Recommended Actions

Condition	Recommended Action
The block names do not comply with the naming standard specified in the input parameters.	Update the block names to comply with your own guidelines or the MAAB guidelines.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag the Basic block names, apart from [Inport] and [Outport], that use underscores at the beginning. (Sub ID: a)
  - The check does not flag the Basic block names, apart from [Inport] and [Outport], that use underscores at the end. (Sub ID: b)
  - The check does not flag the Basic block names, apart from [Inport] and [Outport], that use consecutive underscores. (Sub ID: c)
  - The check does not flag the Basic block names, apart from [Inport] and [Outport], that solely consists of a single reserved MATLAB word. (Sub ID: d)
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

## Edit-Time Checking

This check is supported by edit-time checking.

## Tips

Carriage returns are allowed in block names.

## See Also

- MAAB guideline, Version 3.0: jc\_0231: Usable characters for block names in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0231: Usable characters for block names

## Check Trigger and Enable block names

**Check ID:** mathworks.maab.jc\_0281

Check Trigger and Enable block port names.

### Description

Block port names should match the name of the signal triggering the subsystem. The check does not flag Trigger or Enable block names if the associated signal does not have a label.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Trigger block does not match the name of the signal to which it is connected.	Match Trigger block names to the connecting signal.
Enable block does not match the name of the signal to which it is connected.	Match Enable block names to the connecting signal.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: This check only flags Trigger and Enable blocks names.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0281: Naming of Trigger Port block and Enable Port block in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0281: Trigger signal names

## Check for Simulink diagrams using nonstandard display attributes

**Check ID:** mathworks.maab.na\_0004

Check model appearance setting attributes.

### Description

Model appearance settings are required to conform to the guidelines when the model is released.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The toolbar is not visible.	Select <b>Modeling &gt; Environment &gt; Toolstrip</b> .
The status bar is not visible.	Select <b>Modeling &gt; Environment &gt; Status Bar</b> .
<b>Sample Time &gt; Colors</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Colors</b> .
<b>Wide Nonscalar Lines</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Nonscalar Signals</b> .
<b>Viewer Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Viewers</b> .
<b>Testpoint Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Log &amp; Testpoint</b> .
<b>Port Data Types</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Port Data Type</b> .
<b>Storage Class</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Storage Class</b> .
<b>Signal Dimensions</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Signal Dimensions</b> .

<b>Condition</b>	<b>Recommended Action</b>
<b>Execution Context Indicator</b> is selected.	Clear <b>Display &gt; Signals &amp; Ports &gt; Execution Context Indicator</b> .
<b>Model Browser</b> is selected.	Clear <b>Modeling &gt; Environment &gt; Model Browser</b> .
<b>Sorted Execution Order</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Execution Order</b> .
<b>Model Block Version</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Ref. Model Version</b> .
<b>Model Block I/O Mismatch</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Ref. Model I/O Mismatch</b> .
<b>Library Links</b> is set Disabled, User Defined, or All.	Select <b>Debug &gt; Information Overlays &gt; Show All Links</b> .
<b>Linearization Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Linearization Indicators</b> .
Block backgrounds are not white.	For each listed diagram, click the canvas and select <b>Format &gt; Background</b> .
Block foregrounds are not black.	Select <b>Format &gt; Foreground Color &gt; Black</b> .
Diagrams do not have white backgrounds.	Select <b>Diagram &gt; Format &gt; Canvas Color &gt; White</b> .
Diagrams do not have zoom factor set to 100%.	For each listed diagram, select <b>Modeling &gt; Environment &gt; Zoom &gt; Normal View (100%)</b> .

### Action Results

Clicking **Modify** updates the display attributes to conform to the guideline.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: na\_0004: Simulink model appearance in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0004: Simulink model appearance settings

## Check MATLAB code for global variables

**Check ID:** mathworks.maab.na\_0024

Check for global variables in MATLAB code.

### Description

Verifies that global variables are not used in any of the following:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Global variables are used in one or more of the following: <ul style="list-style-type: none"><li>• MATLAB code in MATLAB Function blocks</li><li>• MATLAB functions defined in Stateflow charts</li><li>• Called MATLAB functions</li></ul>	Replace global variables with signal lines, function arguments, or persistent data.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.

- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: na\_0024: Global Variables in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0024: Shared data in MATLAB functions

## Check visibility of block port names

**Check ID:** mathworks.maab.na\_0005

Check the visibility of port block names.

### Description

An organization applying the MAAB guideline, Version 3.0, must select one of the following alternatives to enforce:

- The names of port blocks are not hidden.
- The name of port blocks must be hidden.

When the check is configured to display all port names, the check do not flag the port names that are hidden due to the **Hide Automatic Names** feature in Simulink.

Available with Simulink Check.

### Input Parameters

#### All Port names should be shown (Format/Show Name)

Select this check box if all ports should show the name, including subsystems.

### Results and Recommended Actions

Condition	Recommended Action
Blocks do not show their name and the <b>All Port names should be shown (Format/Show Name)</b> check box is selected.	Change the format of the specified blocks to show names according to the input requirement.

Condition	Recommended Action
Blocks show their name and the <b>All Port names should be shown (Format/Show Name)</b> check box is cleared.	Change the format of the specified blocks to hide names according to the input requirement.
Subsystem blocks do not show their port names.	Set the subsystem parameter <b>Show port labels</b> to a value other than none.
Subsystem blocks show their port names.	Set the subsystem parameter <b>Show port labels</b> to none.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content in masked subsystems.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

MAAB guideline, Version 3.0: na\_0005: Port block name visibility in Simulink models in the Simulink documentation.

## Check orientation of Subsystem blocks

**Check ID:** `mathworks.maab.jc_0111`

Check the orientation of subsystem blocks.

### Description

Subsystem inputs must be located on the left side of the block, and outputs must be located on the right side of the block.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
Subsystem blocks are not using the right orientation	Rotate the subsystem so that inputs are on the left side of block and outputs are on the right side of the block.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not flag the rotation of subsystems.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- MAAB guideline, Version 3.0: jc\_0111: Direction of Subsystem in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0111: Direction of Subsystem.

## Check usage of Relational Operator blocks

**Check ID:** `mathworks.maab.jc_0131`

Check the position of Constant blocks used in Relational Operator blocks.

## Description

When the relational operator is used to compare a signal to a constant value, the constant input should be the second, lower input.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Relational Operator blocks have a Constant block on the first, upper input.	Move the Constant block to the second, lower input.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0131: Use of Relational Operator block in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0131: Usage of Relational Operators

## Check usage of Switch blocks

**Check ID:** `mathworks.maab.jc_0141`

Check usage of Switch blocks.

### Description

Verifies that the Switch block control input (the second input) is a Boolean value and that the block is configured to pass the first input when the control input is nonzero.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The Switch block control input (second input) is not a Boolean value.	Change the data type of the control input to Boolean.

Condition	Recommended Action
The Switch block is not configured to pass the first input when the control input is nonzero.	Set the block parameter <b>Criteria for passing first input</b> to <code>u2 ~= 0</code> .

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking. However, edit-time checking for this check does not verify that the data type of the control input is a Boolean value.

### See Also

- MAAB guideline, Version 3.0: jc\_0141: Use of the Switch block in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0141: Use of the Switch block
- Switch block

## Check usage of buses and Mux blocks

**Check ID:** `mathworks.maab.na_0010`

Check usage of buses and Mux blocks.

### Description

This check verifies the usage of buses and Mux blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The individual scalar input signals for a Mux block do not have common functionality, data types, dimensions, and units.	Modify the scalar input signals such that the specifications match.
The output of a Mux block is not a vector.	Change the output of the Mux block to a vector.
The input for a Bus Selector block is not a bus signal.	Make sure that the input for all Bus Selector blocks is a bus signal.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not flag when blocks other than Demux and Mux are used to generate or decompose a vector. For example, a Selector block used instead of Demux or a Vector Concatenate used instead of Mux block. (Sub ID: a)
- Does not run on library models.
- Does not allow exclusions of blocks or charts.
- Does not flag non-scalar inputs as described in MAAB guideline na\_0010: Grouping data flows into signals.

### See Also

- MAAB guideline, Version 3.0: na\_0010: Grouping data flows into signals in the Simulink documentation.
- “Composite Signals” (Simulink)

## Check for bitwise operations in Stateflow charts

**Check ID:** `mathworks.maab.na_0001`

Identify bitwise operators (&, |, and ^) in Stateflow charts. If you select **Enable C-bit operations** for a chart, only bitwise operators in expressions containing Boolean data types are reported. Otherwise, all bitwise operators are reported for the chart.

## Description

Do not use bitwise operators in Stateflow charts, unless you enable bitwise operations.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Stateflow charts with <b>Enable C-bit operations</b> selected use bitwise operators (&,  , and ^) in expressions containing Boolean data types.	Do not use Boolean data types in the specified expressions.
The Model Advisor could not determine the data types in expressions with bitwise operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.
Stateflow charts with <b>Enable C-bit operations</b> cleared use bitwise operators (&,  , and ^).	To fix this issue, do either of the following: <ul style="list-style-type: none"> <li>• Modify the expressions to replace bitwise operators.</li> <li>• If not using Boolean data types, consider enabling bitwise operations. In the Chart properties dialog box, select <b>Enable C-bit operations</b>.</li> </ul>

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag when operators other than "!" are used for logical negation. (Sub ID: a)
  - The check cannot be configured to select between "~=", "!=", and "<>" for inequality operations. (Sub ID: b1, b2, b3)
  - The check does not detect when the "^" and "~" bitwise operators are used for Boolean operations. (Sub ID: c)
- Applies only to charts that use C as the action language.

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- hisf\_0003: Usage of bitwise operations
- MAAB guideline, Version 3.0: na\_0001: Bitwise Stateflow operators in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0001: Standard usage of Stateflow operators
- “Binary Operations” (Stateflow).

## Check fundamental logical and numerical operations

**Check ID:** `mathworks.maab.na_0002`

Checks data types in numerical and logic blocks.

### Description

Checks the data types for logical and numerical blocks and identifies when the data type does is not appropriate for the operation.

The data type for logical blocks should be Boolean. Logic blocks include:

- Logical Operator (AND, OR, NOT)
- Enable (port)
- Trigger (port)

The data type for numerical blocks should be non-boolean. Numerical blocks include:

- Sum
- Complex to Real-Imag
- Product
- Dot Product
- Gain

- Sign
- Slider Gain

---

**Note** These blocks also accept numeric input but are not included as part of this check:

- Compare To Constant
  - Compare To Zero
  - Detect Fall Negative
  - Detect Fall Nonpositive
  - Detect Rise Positive
  - Detect Rise Nonnegative
  - Detect Change
  - Detect Decrease
  - Detect Increase
  - Relational Operator
- 

Available with Simulink Check.

### **Input Parameters**

To specify the custom blocks and masks to include in this check, use the Model Advisor Configuration Editor.

- 1** Open the Model Configuration Editor and navigate to **Check fundamental logical and numerical operations**. In the **Input Parameters** pane, select Custom.
- 2** Use the **Blocks for Numerical Operations** and **Blocks for Logical Operations** tables to include or exclude blocks and masks from the check.
- 3** Click **Apply**.
- 4** Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
Data type for a numerical block is Boolean.	Consider changing the data type to a non-boolean.
Data type for a logical block is not Boolean.	Consider changing the data type to a Boolean.
Check does not evaluate my custom blocks and masks.	Enter your custom blocks and masks using the input parameters in the Model Configuration Editor.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0002: Appropriate implementation of fundamental logical and numerical operations in the Simulink documentation.

## Check logical expressions in If blocks

**Check ID:** `mathworks.maab.na_0003`

Check If blocks for inappropriate construct of primary expressions in a logical expression.

### Description

Identifies instances in an If block where primary expressions are complex.

Primary expressions are defined as:

- An input
- A constant
- A constant parameter



- A parenthesized expression containing no operators other than zero or  $<$ ,  $>$ ,  $>=$ ,  $<=$ ,  $=$ ,  $\sim$ ,  $|$ ,  $\&$ , and  $\sim$

Examples of primary expressions include:

- $u1$
- $5$
- $K$
- $(u1 > 0)$
- $(u1 <= G)$
- $(u1 > U2)$
- $(\sim u1)$

Examples of acceptable logical expressions exceptions include:

- $u1 | u2$
- $((u1 > 0) \& (u1 < 20))$
- $(u1 > 0) \& (u2 < u3)$
- $(u1 > 0) \& (\sim u2)$

This table provides examples of unacceptable logical expressions.

Primary Expression	Reasoning
$u1 \& u2   u3$	Too many primary expressions.
$u1 \& (u2   u3)$	Unacceptable operator within primary expression.
$(u1 > 0) \& (u1 < 20) \& (u2 > 5)$	Too many primary expressions that are not inputs.
$(u1 > 0) \& ((2 * u2) > 6)$	Unacceptable operator within primary expression.

### Exception

A logical expression can contain more than two primary expressions when both these conditions are met:

- The primary expressions are all inputs.

- Only one type of logical operator is present.

Examples of acceptable exceptions include:

- `u1 | u2 | u3 | u4 | u5`
- `u1 & u2 & u3 & u4`

### Simple "If" Expressions

In the literal interpretation of guideline na\_0003, expression `u1 < u2` is a violation. However, the expression follows the commonly used "If" expression template (<Primary Expression><Operator><Primary Expression>). So, when logical operators are not used and only one relational operator is present, the expression satisfies guideline na\_0003 and `u1 < u2` is NOT a violation.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Logical expression contains more than two primary expressions that consist of a constant, constant parameter, and input.	Consider one of the following: <ul style="list-style-type: none"> <li>• Make primary expressions an input and either:                             <ul style="list-style-type: none"> <li>• Use parenthesized expressions with one relational operator type</li> <li>• Construct a simple "If" express using template                                  &lt;Primary Expression&gt;&lt;Logical Operator&gt;&lt;Primary Expression&gt;</li> </ul> </li> <li>• Reduce the number of primary expressions to two or less.</li> <li>• Construct the logical expression using logical blocks other than the If block.</li> </ul>

Condition	Recommended Action
Logical expression contains more than two parenthesized expressions that use multiple relational operators	Consider one of the following: <ul style="list-style-type: none"> <li>• Use only one type of relational operator. Acceptable logical operators include &lt;, &gt;, &gt;=, &lt;=, ==, ~=,  , &amp;, and ~. The primary expression must consist of inputs only.</li> <li>• Reduce the number of parenthesized expressions to two or less.</li> <li>• Construct the logical expression using logical blocks other than the If block.</li> </ul>
Parenthesized expression includes a relational operator other than zero or <, >, >=, <=, ==, ~=,  , &, or ~.	Consider one of the following: <ul style="list-style-type: none"> <li>• Use relational operator &lt;, &gt;, &gt;=, &lt;=, ==, ~=,  , &amp;, or ~ within the parenthesized expression.</li> <li>• Construct the logical expression using logical blocks other than the If block.</li> </ul>

### Capabilities and Limitations

- Does not flag logical expressions that use only one of these relative operators <, >, >=, <=, ==, ~=, |, &, and ~
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

### See Also

- MAAB guideline, Version 3.0: na\_0003: Simple logical expressions in If Condition block in the Simulink documentation.

## Check for comparison operations in Stateflow charts

**Check ID:** mathworks.maab.na\_0013

Identify comparison operations with different data types in Stateflow objects.

**Description**

Comparisons should be made between variables of the same data types.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Comparison operations with different data types were found.	Revisit the specified operations to avoid comparison operations with different data types.
The Model Advisor could not determine the data types in expressions with comparison operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

**Capabilities and Limitations**

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- MAAB guideline, Version 3.0: na\_0013: Comparison operation in Stateflow in the Simulink documentation.

**Check usage of restricted variable names**

**Check ID:** mathworks.maab.na\_0019

Check for use of reserved keywords in MATLAB Function block variable names.

**Description**

Identifies variable names in MATLAB Function blocks that conflict with reserved C and C++ keywords. For a complete list of reserved keywords, see “Reserved Keywords” (Simulink Coder).

Avoid using variable names that conflict with MATLAB Functions, such as `conv`.

This check is case insensitive. For example, the check flags keywords `true`, `True`, `TRUE`, and `tRue`.

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
Variable name conflicts with reserved keyword.	Consider using a different variable name that does not conflict with the reserved keywords.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

**See Also**

- MAAB guideline, Version 3.0: `na_0019`: Restricted Variable Names in the Simulink documentation.
- `hisl_0032`: Model object names

**Check unused ports in Variant Subsystems**

**Check ID:** `mathworks.maab.na_0020`

Checks variant subsystems for unused ports.

**Description**

Checks variant subsystems for unused ports and provides the action to terminate the unused inputs with a Terminator block.

**Results and Recommended Actions**

Condition	Recommended Action
Variant subsystem has unused ports	Consider connecting the unused ports to Terminator blocks.

Check action connects unused ports to Terminator blocks.

Available with Simulink Check.

**Capabilities and Limitations**

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag when the number of inputs to the subordinate subsystem are different. (Sub ID: a)
  - The check does not flag when the number of inputs to the subordinate model are different. (Sub ID: b)
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Allows syntax highlighting.

**See Also**

- MAAB guideline, Version 3.0: na\_0020: Number of inputs to variant subsystems in the Simulink documentation.

**Check usage of character vector inside MATLAB Function block**

**Check ID:** `mathworks.maab.na_0021`

Check for use of character vectors in MATLAB Function blocks.

### Description

Identifies character vectors that are used in MATLAB Function blocks.

MATLAB Functions store strings as character arrays. Due to lack of dynamic memory allocation, the arrays cannot be re-sized to accommodate a string value of different length. Strings are not a supported data type in Simulink, so MATLAB Function blocks cannot pass the string data outside the block.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
MATLAB Function block contains a character vector.	Consider using enumerations instead of character vectors.

### Capabilities and Limitations

- Does not flag strings in MATLAB
- Does not flag character vectors that are hard-coded into the class definition.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

### See Also

- MAAB guideline, Version 3.0: na\_0021: Strings in the Simulink documentation.

## Check usage of recommended patterns for Switch/Case statements

**Check ID:** mathworks.maab.na\_0022

Check for use of non-constant variables in Switch/Case statements.

### Description

In generated code, MATLAB Function block inputs are passed as functional arguments. This check evaluates the Switch/Case statements in the generated code to determine if non-constant values are being used in the Case argument.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Non-constant variables are used in the Switch/Case statement.	Consider defining the input variable as a constant.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

### See Also

- MAAB guideline, Version 3.0: na\_0022: Recommended patterns for Switch/Case statements in the Simulink documentation.

## Check use of default variants

**Check ID:** `mathworks.maab.na_0036`

Check use of default variants in a variant subsystem.

### Description

Checks Variant Subsystem, Variant Source, Variant Sink, and variant Model blocks in a variant subsystem for a default variant.

Available with Simulink Check.



## Input Parameters

To set the active variant as the default variant, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check use of default variants**. In the **Input Parameters** pane, select **Check use of 'Allow zero active variant controls' option**.

---

**Note** When using **Allow zero active variant controls** option, set the following on output ports of the variant subsystem:

- Set **Specify output when source is unconnected** to true
  - Provide a valid value in **Constant value**
  - Set **Output Data type** to Inherit: auto
- 

- 2 Click **Apply**.
- 3 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

## Results and Recommended Actions

Condition	Recommended Action
The subsystem does not contain a default variant.	Set block parameter <b>Variant control</b> to (default)

Condition	Recommended Action
Block parameter <b>Variant Control</b> is set to Variant.	<p>To set the active variant as the default variant.</p> <ol style="list-style-type: none"> <li><b>1</b> <b>Variant Control</b> is set to Variant</li> <li><b>2</b> Open the variant block and select block parameter <b>Allow zero active variant controls</b>.</li> <li><b>3</b> For output ports of the variant subsystem: <ul style="list-style-type: none"> <li>• Set <b>Specify output when source is unconnected</b> to true</li> <li>• Provide a valid value in <b>Constant value</b></li> <li>• Set <b>Output Data type</b> to Inherit: auto</li> </ul> </li> </ol>

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts
- Allows syntax highlighting

**See Also**

- MAAB guideline, Version 3.0: na\_0036: Default variant in the Simulink documentation.

**Check use of single variable variant conditionals**

**Check ID:** mathworks.maab.na\_0037

Check use of single variables in conditional expressions

## Description

Checks Variant Subsystem, Variant Source, Variant Sink, and variant Model blocks for conditional expressions that have more than one variable.

**Note** Guideline na\_0037 states that default variants are an exception to the recommendation of writing variant conditional expressions using multiple variable with a single condition. You can define a default by:

- Selecting (default) in the block parameter **Variant control**.
- Specifying an exhaustive condition.

This check cannot differentiate between defaults that are defined using an exhaustive condition.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Conditional expression contains more than one condition variable.	Consider updating your model so that only one variant is used.
Conditional expression variable or <code>Simulink.Variant</code> object is not found in the workspace.	Consider defining a variant in your model.
Check does not execute on my variant subsystem.	Clear <b>Override variant conditions and use following variant</b> for the variant subsystem.

## Capabilities and Limitations

- Does not check default variants.
- Does not run on the Variant subsystem when you select **Override variant conditions and use following variant**
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

**See Also**

- MAAB guideline, Version 3.0: na\_0037: Use of single variable variant conditionals in the Simulink documentation.

**Check nested states in Stateflow charts****Check ID:** `mathworks.maab.na_0038`

Checks the depth of nested states in Stateflow charts.

**Description**

Checks the depth of nested states in Stateflow charts and identifies states that exceed the defined nesting level threshold.

MAAB guideline na\_0038 recommends three levels.

Available with Simulink Check.

**Input Parameters**

To specify the threshold for nesting levels of Stateflow states, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check nested states in Stateflow charts**.
- 2 In the **Input Parameters** pane, enter the nesting level threshold. The default value is 3.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Level of nested Stateflow states exceeds the defined threshold.	Consider encapsulating Stateflow states in subcharts so level of nested states does not exceed the threshold.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Allows syntax highlighting.

### See Also

- MAAB guideline, Version 3.0: na\_0038: Levels in Stateflow charts in the Simulink documentation.

## Check use of Simulink in Stateflow charts

**Check ID:** mathworks.maab.na\_0039

Checks for Stateflow charts that are nested in Simulink functions used in the root Stateflow chart.

### Description

Checks Simulink functions in the root Stateflow chart and identifies Stateflow charts that are nested within these functions.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The Simulink function has a nested Stateflow chart.	Consider modifying your root Stateflow chart so the Simulink function does not contain a nested Stateflow chart.

### Capabilities and Limitations

- Does not check Stateflow states.
- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Allows syntax highlighting.

### See Also

- MAAB guideline, Version 3.0: na\_0039: Use of Simulink in Stateflow charts in the Simulink documentation.

## Check number of Stateflow states per container

**Check ID:** `mathworks.maab.na_0040`

Checks the number of viewable states within a Stateflow container.

### Description

Checks the number of viewable states in a Stateflow container and identifies containers that exceed the defined threshold. A state is considered visible if it is not within an atomic subchart, function, or subchart.

MAAB guideline na\_0040 recommends six to 10 states per container.

Available with Simulink Check.

### Input Parameters

To specify the threshold for the number of viewable states in a Stateflow container, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check number of Stateflow states per container**.
- 2 In the **Input Parameters** pane, enter the threshold for the number of viewable states per container. The default value is 10.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

## Results and Recommended Actions

Condition	Recommended Action
Number of viewable states in the Stateflow container exceeds the defined threshold.	Consider encapsulating Stateflow states in subcharts so the number of states per container does not exceed the threshold.

## Capabilities and Limitations

- Atomic subcharts are considered as states and included in the check.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Allows syntax highlighting.

## See Also

- MAAB guideline, Version 3.0: na\_0040: Number of states per container in the Simulink documentation.

## Check usage of unary minus operations in Stateflow charts

**Check ID:** `mathworks.maab.jc_0451`

Identify unary minus operations applied to unsigned integers in Stateflow objects.

### Description

Do not perform unary minus operations on unsigned integers in Stateflow objects.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Unary minus operations are applied to unsigned integers in Stateflow objects.	Modify the specified objects to remove dependency on unary minus operations.
The Model Advisor could not determine the data types in expressions with unary minus operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

### Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0451: Use of unary minus on unsigned integers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0451: Use of unary minus on unsigned integers

## Check usage of floating-point expressions in Stateflow charts

**Check ID:** `mathworks.maab.jc_0481`

Identify equal to operations (==) in expressions where at least one side of the expression is a floating-point variable or constant.

### Description

Do not use equal to operations with floating-point data types. You can use equal to operations with integer data types.

Available with Simulink Check.

This check requires a Stateflow license.



## Results and Recommended Actions

Condition	Recommended Action
Expressions use equal to operations (==) where at least one side of the expression is a floating-point variable or constant.	Modify the specified expressions to avoid equal to operations between floating-point expressions. If an equal to operation is required, a margin of error should be defined and used in the operation.
The Model Advisor could not determine the data types in expressions with equality operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

## Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## See Also

MAAB guideline, Version 3.0: jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow in the Simulink documentation.

## Check input and output settings of MATLAB Functions

**Check ID:** mathworks.maab.na\_0034

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

### Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	<p>Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Function identified in the results.</p> <p>If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink):</p> <ul style="list-style-type: none"> <li>• Change <b>Complexity</b> from Inherited to On or Off.</li> <li>• Change <b>Type</b> from Inherit: Same as Simulink to an explicit type.</li> <li>• Change <b>Size</b> from -1 (Inherited) to an explicit size.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0034: MATLAB Function block input/output settings in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0034: MATLAB Function block input/output settings

## Check MATLAB Function metrics

**Check ID:** mathworks.maab.himl\_0003

Display complexity and code metrics for MATLAB Functions. Report metric violations.

**Description**

This check provides complexity and code metrics for MATLAB Functions. The check additionally reports metric violations.

A results table provides links to MATLAB Functions that violate the complexity input parameters.

Available with Simulink Check.

**Input Parameters****Maximum effective lines of code per function**

Provide the maximum effective lines of code per function. Effective lines do not include empty lines, comment lines, or lines with a function end keyword.

**Minimum density of comments**

Provide minimum density of comments. Density is ratio of comment lines to total lines of code.

**Maximum cyclomatic complexity per function**

Provide maximum cyclomatic complexity per function. Cyclomatic complexity is the number of linearly independent paths through the source code.

**Results and Recommended Actions**

Condition	Recommended Action
MATLAB Function violates the complexity input parameters.	For the MATLAB Function: <ul style="list-style-type: none"> <li>• If effective lines of code is too high, further divide the MATLAB Function.</li> <li>• If comment density is too low, add comment lines.</li> <li>• If cyclomatic complexity per function is too high, further divide the MATLAB Function.</li> </ul>

**Capabilities and Limitations**

- Runs on library models.

- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0016: Source lines of MATLAB Functions in the Simulink documentation.
- MAAB guideline, Version 3.0: na\_0018: Number of nested if/else and case statement in the Simulink documentation.

## Check for names of Stateflow ports and associated signals

**Check ID:** `mathworks.maab.db_0123`

Check for mismatches between Stateflow ports and associated signal names.

### Description

The name of Stateflow input and output should be the same as the corresponding signal. The check does not flag:

- Name mismatches for reusable Stateflow charts in libraries.
- Stateflow ports if the corresponding signal does not have a label.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Signals have names that differ from the corresponding Stateflow ports.	Change the names of either the signals or the Stateflow ports.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts. Exclusions will not work for library linked charts.

### See Also

- MAAB guideline, Version 3.0: db\_0123: Stateflow port names in the Simulink documentation.

## Check scope of From and Goto blocks

**Check ID:** mathworks.maab.na\_0011

Check the scope of From and Goto blocks.

### Description

You can use global scope for controlling flow. However, From and Goto blocks must use local scope for signal flows.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
From and Goto blocks are not configured with local scope.	<ul style="list-style-type: none"> <li>• Make sure that the ports are connected.</li> <li>• Change the scope of the specified blocks to local.</li> </ul>

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0011: Scope of Goto and From blocks in the Simulink documentation.

## Check the number of function calls in MATLAB Function blocks

**Check ID:** mathworks.maab.na\_0017

### Description

Checks whether number of function calls in MATLAB Function blocks is less than the set threshold. By default, the limit is set to three.

### Results and Recommended Actions

Condition	Recommended Action
Number of function calls in MATLAB Function blocks is greater than the set threshold. by default the set threshold is three. The set threshold can be modified by using the input parameter <b>Function Call Level</b> in the configuration editor.	Reduce the number of function calls from MATLAB Function blocks to be less than the set threshold.

### Capabilities and Limitations

- Recursive function calls are only counted once.
- Inline class methods are not analyzed.
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- MAAB guideline, Version 3.0: na\_0017: Number of called function levels in the Simulink documentation.

## Check usage of Goto and From blocks between Subsystems

**Check ID:** mathworks.maab.jc\_0171

### Description

Checks the signal flow when using Goto and From blocks.

This check identifies the subsystems connected to each other that use Goto and From blocks in feed-forward and feedback loops that do not have at least one signal line for each direction.

### Results and Recommended Actions

Condition	Recommended Action
Subsystems connected to each other using Goto and From blocks in feed-forward and feedback loops do not have at least one signal line for each direction.	Make sure to add at least one direct signal line in between the connected subsystems.

### Capabilities and Limitations

- The check allows the connections made from the referenced subsystems with in another subsystem.
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- MAAB guideline, Version 3.0: jc\_0171: Maintaining signal flow when using Goto and From blocks

## Check reuse of variables within a Stateflow scope

**Check ID:** mathworks.maab.jc\_0491

### Description

Checks if the defined variables have multiple meanings. The same variable must not have multiple usages within a single Stateflow state.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Variables with same name are used in a single subsystem.	Make sure that all the defined variables are unique including the temporary loop control variables.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

This check does not run on:

- MATLAB function block
- Truth tables
- Graphical functions
- If/switch in action language in Stateflow states.

### See Also

- MAAB guideline, Version 3.0: jc\_0491: Reuse of variables within a single Stateflow scope

## Check for Stateflow transition appearance

**Check ID:** mathworks.maab.db\_0129

### Description

Checks and reports Stateflow transitions that are visually overlapping other Stateflow objects.

Available with Simulink Check.

This check requires a Stateflow license.



## Results and Recommended Actions

Condition	Recommended Action
Stateflow transitions overlap with other Stateflow transitions.	Avoid having Stateflow transitions overlap with other Stateflow transitions.
Stateflow transitions overlap with labels of other Stateflow transitions.	Avoid having Stateflow transitions overlap with labels of other Stateflow transitions.
Stateflow transitions overlap with junctions.	Avoid having Stateflow transitions overlap with junctions.
Stateflow transitions overlap with states.	Avoid having Stateflow transitions overlap with states.
Stateflow transitions overlap with boxes.	Avoid having Stateflow transitions overlap with boxes.
Stateflow transitions overlap with Simulink functions.	Avoid having Stateflow transitions overlap with Simulink functions.
Stateflow transitions overlap with MATLAB functions.	Avoid having Stateflow transitions overlap with MATLAB functions.
Stateflow transitions overlap with annotations.	Avoid having Stateflow transitions overlap with annotations.
Stateflow transitions overlap with truth tables.	Avoid having Stateflow transitions overlap with truth tables.
Stateflow transitions labels are not visually associated to the corresponding transition.	Transition labels must be visually associated to the corresponding transition.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag transition lines that are not drawn vertically or horizontally. (Sub ID: d)
  - The check does not flag unnecessary connective junctions. (Sub ID: e)
  - This check does not report the Stateflow transitions that overlap with the text inside a state that overflows the label of a state.
- Runs on library models.
- Analyzes content in all masked subsystems.

- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- MAAB guideline, Version 3.0: db\_0129: Stateflow transition appearance

## Check usage of enumerated values

**Check ID:** `mathworks.maab.na_0031`

### Description

Identifies the enumeration classes used in the model without a default value specification.

### Results and Recommended Actions

Condition	Recommended Action
Enumeration classes in the model do not have a default value specification.	Make sure that the enumeration classes used in the model have a <b>getDefaultValue</b> method implementation as a static method of the class.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- MAAB guideline, Version 3.0: na\_0031: Definition of default enumerated value

## Check Simulink signal appearance

**Check ID:** `mathworks.maab.db_0032`

## Description

Checks if the Simulink signals that are intersecting and overlapping adhere to recommended guidelines.

## Results and Recommended Actions

Condition	Recommended Action
Simulink signals intersect each other.	Make sure that the signals do not intersect with each other.
Simulink signals are not drawn with right angles.	Make sure that the signals are drawn with right angles throughout the model.
Simulink signals overlap each other.	Make sure that the signals do not overlap with each other.
Simulink signals are drawn over a Simulink block.	Make sure that the signals are not drawn over any Simulink blocks.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not verify the Line crossing style property setting. (Sub ID: a2)
  - The check does not flag lines that split into more than two lines at a single branching point. (Sub ID: d)
- Signal hop preference is not considered.
- Block label overlaps are not analyzed.
- Simulink signals that split into more than two signals at a single branch are not considered.
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of subsystems.
- Analyzes content of library linked blocks.

## See Also

- MAAB guideline, Version 3.0: db\_0032: Simulink signal appearance

## Check usage of non-compliant blocks

**Check ID:** mathworks.maab.na\_0027

### Description

Identifies the blocks that are not listed in the compliant block list for a model.

### Results and Recommended Actions

Condition	Recommended Action
One or more Simulink blocks that are not listed in the compliant block list are used in the model.	Choose Simulink blocks from the compliant block list in configuration parameters dialog box.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- MAAB guideline, Version 3.0: na\_0027: Use of only standard library blocks in the Simulink documentation.

“Blocks and Products Supported for Code Generation” (Simulink Coder)

## Japan MATLAB Automotive Advisory Board Checks

### In this section...

- "Japan MATLAB Automotive Advisory Board Checks" on page 2-264
- "Check for mixing basic blocks and subsystems" on page 2-266
- "Check Implement logic signals as Boolean data (vs. double)" on page 2-267
- "Check for Simulink diagrams using nonstandard display attributes" on page 2-268
- "Check font formatting" on page 2-270
- "Check positioning and configuration of ports" on page 2-272
- "Check whether block names appear below blocks" on page 2-274
- "Check the display attributes of block names" on page 2-275
- "Check position of Trigger and Enable blocks" on page 2-277
- "Check for nondefault block attributes" on page 2-278
- "Check for unconnected ports and signal lines" on page 2-279
- "Check usage of Switch blocks" on page 2-280
- "Check usage of Relational Operator blocks" on page 2-281
- "Check for indexing in blocks" on page 2-282
- "Check usage of tunable parameters in blocks" on page 2-284
- "Check usage of Discrete-Time Integrator block" on page 2-286
- "Check usage of fixed-point data type with non-zero bias" on page 2-287
- "Check input and output datatype for Switch blocks" on page 2-287
- "Check signs of input signals in product blocks" on page 2-288
- "Check transition orientations in flow charts" on page 2-289
- "Check return value assignments in Stateflow graphical functions" on page 2-290
- "Check for Strong Data Typing with Simulink I/O" on page 2-291
- "Check Stateflow data objects with local scope" on page 2-292
- "Check for MATLAB expressions in Stateflow charts" on page 2-294
- "Check for pointers in Stateflow charts" on page 2-295
- "Check for event broadcasts in Stateflow charts" on page 2-296
- "Check for bitwise operations in Stateflow charts" on page 2-297

**In this section...**

- "Check usage of unary minus operations in Stateflow charts" on page 2-298
- "Check input and output settings of MATLAB Functions" on page 2-299
- "Check MATLAB code for global variables" on page 2-300
- "Check usage of Memory and Unit Delay blocks" on page 2-301
- "Check block orientation" on page 2-302
- "Check usage of internal transitions in Stateflow states" on page 2-303
- "Check usage of transition conditions in Stateflow transitions" on page 2-304
- "Check usable characters for signal names and bus names" on page 2-305
- "Check usable characters for parameter names" on page 2-306
- "Check length of model file name" on page 2-307
- "Check length of folder name at every level of model path" on page 2-308
- "Check length of subsystem names" on page 2-308
- "Check length of Inport and Outport names" on page 2-309
- "Check length of signal and bus names" on page 2-310
- "Check length of parameter names" on page 2-311
- "Check length of block names" on page 2-312
- "Check if blocks are shaded in the model" on page 2-313
- "Check operator order of Product blocks" on page 2-313
- "Check icon shape of Logical Operator blocks" on page 2-314
- "Check if tunable block parameters are defined as named constants" on page 2-315
- "Check default/else case in Switch Case blocks and If blocks" on page 2-316
- "Check for parentheses in Fcn block expressions" on page 2-317
- "Check undefined initial output for conditional subsystems" on page 2-318
- "Check usage of the Saturation blocks" on page 2-318
- "Check type setting by data objects" on page 2-319
- "Check prohibited comparison operation of logical type signals" on page 2-320
- "Check usage of Stateflow comments" on page 2-321
- "Check updates to variables used in state transition conditions" on page 2-322
- "Check for unexpected backtracking in state transitions" on page 2-323

**In this section...**

- "Check usage of parentheses in Stateflow transitions" on page 2-324
- "Check condition actions and transition actions in Stateflow" on page 2-325
- "Check starting point of internal transition in Stateflow" on page 2-326
- "Check prohibited combination of state action and flow chart" on page 2-327
- "Check usage of Lookup Tables" on page 2-328
- "Check Signed Integer Division Rounding mode" on page 2-329
- "Check usage of Merge block" on page 2-330
- "Check for unused data in Stateflow Charts" on page 2-331
- "Check usable number for first index" on page 2-332
- "Check execution timing for default transition path" on page 2-333
- "Check for parallel Stateflow state used for grouping" on page 2-333
- "Check scope of data in parallel states" on page 2-334
- "Check uniqueness of State names" on page 2-335
- "Check usage of State names" on page 2-336
- "Check uniqueness of Stateflow State and Data names" on page 2-337
- "Check repetition of Action types" on page 2-338
- "Check indentation of code in Stateflow states" on page 2-339
- "Check placement of Label String in Transitions" on page 2-340
- "Check duplication of Simulink data names" on page 2-341
- "Comparing floating point types in Simulink" on page 2-342
- "Check for avoiding algebraic loops between subsystems" on page 2-343
- "Check default transition placement in Stateflow charts" on page 2-343
- "Check settings for data ports in Multiport Switch blocks" on page 2-345
- "Check for usage of text inside states" on page 2-346
- "Check for unconnected objects in Stateflow Charts" on page 2-346
- "Check Stateflow chart action language" on page 2-347
- "Check usable characters for Stateflow data names" on page 2-348
- "Check length of Stateflow data name" on page 2-349
- "Check signal line labels" on page 2-350

### **In this section...**

- “Check for propagated signal labels” on page 2-351
- “Check trigger signal names” on page 2-352
- “Check for consistency in model element names” on page 2-354
- “Check for sample time setting” on page 2-355
- “Check output data type of operation blocks” on page 2-356
- “Check usage of numeric literals in Stateflow” on page 2-357
- “Check usage of transitions to external states” on page 2-358
- “Check order of state action types” on page 2-358
- “Check if state action type 'exit' is used in the model” on page 2-359
- “Check position of comments in transition labels” on page 2-360
- “Check usage of unconditional transitions in flow charts” on page 2-361
- “Check for comments in unconditional transitions” on page 2-362
- “Check terminal junctions in Stateflow” on page 2-363
- “Check unused data in Simulink Model” on page 2-364
- “Check for use of C-style comment symbols” on page 2-365
- “Check for implicit type casting in Stateflow” on page 2-366
- “Check diagnostic settings for incorrect calculation results” on page 2-367
- “Check block names” on page 2-368
- “Check Model Description” on page 2-369
- “Usable characters for file names” on page 2-370
- “Usable characters for folder names” on page 2-371
- “Usable characters for Subsystem names” on page 2-373
- “Usable characters for Inport block and Outport block” on page 2-374

## **Japan MATLAB Automotive Advisory Board Checks**

Japan MATLAB Automotive Advisory Board (JMAAB) checks facilitate designing and troubleshooting models from which code is generated for automotive applications.

The Model Advisor performs a checkout of the Simulink Check license when you run JMAAB checks.



JMAAB checks are classified in to the following groups:

### **Naming Conventions**

Japan MATLAB Automotive Advisory Board (JMAAB) checks related to naming conventions.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### **Model Architecture**

Japan MATLAB Automotive Advisory Board (JMAAB) checks related to model architecture.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### **Model Configuration Options**

Japan MATLAB Automotive Advisory Board (JMAAB) checks related to model configuration options.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### **Simulink**

Japan MATLAB Automotive Advisory Board (JMAAB) checks related to Simulink Blocks and components.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### **Stateflow**

Japan MATLAB Automotive Advisory Board (JMAAB) checks related Stateflow charts and components.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### MATLAB Functions

Japan MATLAB Automotive Advisory Board (JMAAB) checks related to MATLAB Functions.

For more information, see “Model Checks for Japan MATLAB Automotive Advisory Board (JMAAB) Guideline Compliance”.

### See Also

- “Run Model Advisor Checks and Review Results”
- The *Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow* JMAAB guidelines on the MathWorks website

## Check for mixing basic blocks and subsystems

**Check ID:** `mathworks.maab.db_0143`

Check for systems that mix primitive blocks and subsystems.

### Description

You must design each level of a model with building blocks of the same type, for example, only subsystems or only primitive (basic) blocks. If you mask your subsystem and set `MaskType` to a nonempty string, the Model Advisor treats the subsystem as a basic block.

Available with Simulink Check.

### Input Parameters

To change the list of blocks that the check flags, you can use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check for mixing basic blocks and subsystems**.
- 2 In the **Input Parameters** pane, to:
  - Allow the blocks specified by MAAB 3.0, from **Standard**, select **MAAB 3.0**. The **Block type list** table provides the blocks that MAAB 3.0 allows at any model level.

- To specify blocks to either allow or prohibit, from **Standard**, select Custom. In **Treat blocktype list as**, select Allowed or Prohibited. In the **Block type list** table, you can add or remove blocks.
- 3 Click **Apply**.
  - 4 Save the configuration. When you run the check using this configuration, the check uses the specified input parameters.

### Results and Recommended Actions

Condition	Recommended Action
A level in the model includes subsystem blocks and primitive blocks.	Move nonvirtual blocks into the subsystem.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0143: Similar block types on the model levels in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0143: Usage block types in model hierarchy
- “Overview of the Model Advisor Configuration Editor”

## Check Implement logic signals as Boolean data (vs. double)

**Check ID:** `mathworks.maab.jc_0011`

Check the optimization parameter for Boolean data types.

### Description

Optimization for Boolean data types is required

Available with Simulink Check.

### Prerequisite

A prerequisite MAAB guideline, Version 3.0, for this check is na\_0002: Appropriate implementation of fundamental logical and numerical operations.

### Results and Recommended Actions

Condition	Recommended Action
Configuration setting for <b>Implement logic signals as boolean data (vs. double)</b> is not set.	Select the <b>Implement logic signals as boolean data (vs. double)</b> check box in the Configuration Parameters dialog box.

### Capabilities and Limitations

- Does not run on library models.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0011: Optimization parameters for Boolean data types in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0011: Optimization parameters for Boolean data types.

## Check for Simulink diagrams using nonstandard display attributes

**Check ID:** `mathworks.maab.na_0004`

Check model appearance setting attributes.

### Description

Model appearance settings are required to conform to the guidelines when the model is released.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The toolbar is not visible.	Select <b>Modeling &gt; Environment &gt; Toolstrip</b> .
The status bar is not visible.	Select <b>Modeling &gt; Environment &gt; Status Bar</b> .
<b>Sample Time &gt; Colors</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Colors</b> .
<b>Wide Nonscalar Lines</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Nonscalar Signals</b> .
<b>Viewer Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Viewers</b> .
<b>Testpoint Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Log &amp; Testpoint</b> .
<b>Port Data Types</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Port Data Type</b> .
<b>Storage Class</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Storage Class</b> .
<b>Signal Dimensions</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Signal Dimensions</b> .
<b>Execution Context Indicator</b> is selected.	Clear <b>Display &gt; Signals &amp; Ports &gt; Execution Context Indicator</b> .
<b>Model Browser</b> is selected.	Clear <b>Modeling &gt; Environment &gt; Model Browser</b> .
<b>Sorted Execution Order</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Execution Order</b> .
<b>Model Block Version</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Ref. Model Version</b> .
<b>Model Block I/O Mismatch</b> is selected.	Clear <b>Debug &gt; Information Overlays &gt; Ref. Model I/O Mismatch</b> .
<b>Library Links</b> is set Disabled, User Defined, or All.	Select <b>Debug &gt; Information Overlays &gt; Show All Links</b> .

Condition	Recommended Action
<b>Linearization Indicators</b> is cleared.	Select <b>Debug &gt; Information Overlays &gt; Linearization Indicators</b> .
Block backgrounds are not white.	For each listed diagram, click the canvas and select <b>Format &gt; Background</b> .
Block foregrounds are not black.	Select <b>Format &gt; Foreground Color &gt; Black</b> .
Diagrams do not have white backgrounds.	Select <b>Diagram &gt; Format &gt; Canvas Color &gt; White</b> .
Diagrams do not have zoom factor set to 100%.	For each listed diagram, select <b>Modeling &gt; Environment &gt; Zoom &gt; Normal View (100%)</b> .

### Action Results

Clicking **Modify** updates the display attributes to conform to the guideline.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- MAAB guideline, Version 3.0: na\_0004: Simulink model appearance in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0004: Simulink model appearance settings

### Check font formatting

**Check ID:** `mathworks.maab.db_0043`

Check for difference in font and font sizes.

## Description

With the exception of free text annotations within a model, text elements, such as block names, block annotations, and signal labels, must have the same font style and font size. Select a font style and font size that is legible and portable (convertible between platforms), such as Arial or Times New Roman 12 point. To specify font rules for a Simulink session, from the Simulink editor select **Diagram > Format > Font Styles for Model**.

Available with Simulink Check.

## Input Parameters

### Font Name

Apply the specified font to all text elements. When you specify Common (default), the check identifies different fonts used in your model. Although you can specify other fonts, the fonts available from the drop-down list are Arial, Courier New, Georgia, Times New Roman, Arial Black, and Verdana.

### Font Size

Apply the specified font size to all text elements. When you specify Common (default), the check identifies different font sizes used in your model. Although you can specify other font sizes, the font sizes available from the drop-down list are 6, 8, 9, 10, 12, 14, 16.

### Font Style

Apply the specified font style to all text elements. When you specify Common (default), the check identifies different font styles used in your model. The font styles available from the drop-down list are normal, bold, italic, and bold italic.

## Results and Recommended Actions

Condition	Recommended Action
The fonts or font sizes for text elements in the model are not consistent or portable.	Specify values for the font parameters and in the right pane of the Model Advisor, click <b>Modify all Fonts</b> , or manually change the fonts and font sizes of text elements in the model so they are consistent and portable.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag Stateflow state labels and box name font and font style settings. (Sub ID: c)
  - The check does not flag Stateflow state labels and box name font size settings. (Sub ID: d)
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### Action Results

In the right pane of the Model Advisor, clicking **Modify all Fonts** changes the font and font size of all text elements in the model according to the values you specify in the input parameters.

For the input parameters, if you specify **Common**, clicking **Modify all Fonts** changes the font and font sizes of all text elements in the model to the most commonly used fonts, font sizes, or font styles.

### See Also

- MAAB guideline, Version 3.0: db\_0043: Simulink font and font size in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0043: Model font and font size

## Check positioning and configuration of ports

**Check ID:** `mathworks.maab.db_0042`

Check whether the model contains ports with invalid position and configuration.

### Description

In models, ports must comply with the following rules:



- Place Inport blocks on the left side of the diagram. It is acceptable to move the Inport block to the right only to prevent signal crossings.
- Place Outport blocks on the right side of the diagram. It is acceptable to move the Outport block to the left only to prevent signal crossings.
- Avoid using duplicate Inport blocks at the subsystem level if possible.
- Do not use duplicate Inport blocks at the root level.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Inport blocks are too far to the right and result in left-flowing signals.	Move the specified Inport blocks to the left.
Output blocks are too far to the left and result in left-flowing signals.	Move the specified Output blocks to the right.
Ports do not have the default orientation.	Modify the model diagram such that signal lines for output ports enter the side of the block and signal lines for input ports exit the right side of the block.
Ports are duplicate Inport blocks.	<ul style="list-style-type: none"> <li>• If the duplicate Inport blocks are in a subsystem, remove them where possible.</li> <li>• If the duplicate Inport blocks are at the root level, remove them.</li> </ul>

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag when Inports are not placed on the left side of the diagram. (Sub ID: a)
  - The check does not flag when Outports are not placed on the right side of the diagram. (Sub ID: b)
- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Does not analyze signal crossings

### See Also

- MAAB guideline, Version 3.0: db\_0042: Port block in Simulink models in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0042: Usage of Inport and Outport blocks

## Check whether block names appear below blocks

**Check ID:** `mathworks.maab.db_0142`

Check whether block names appear below blocks.

### Description

If shown, the name of the block should appear below the block.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Blocks have names that do not appear below the blocks.	Set the name of the block to appear below the blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

**See Also**

- MAAB guideline, Version 3.0: db\_0142: Position of block names in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0142: Position of block names.

**Check the display attributes of block names**

**Check ID:** mathworks.maab.jc\_0061

Check the display attributes of subsystem and block names.

**Description**

When the subsystem and block names provide descriptive information, display the names. If the block function is known from its appearance, do not display the name. Blocks with names that are obvious from the block appearance:

- From
- Goto
- Ground
- Logic
- MinMax
- ModelReference
- MultiPortSwitch
- Product
- Relational Operator
- Saturate
- Switch
- Terminator
- Trigonometry
- Unit Delay
- Sum
- Compare To Constant
- Compare To Zero

Available with Simulink Check.

### Input Parameters

To specify the custom blocks and masks to include in this check, use the Model Advisor Configuration Editor.

- 1 Open the Model Configuration Editor and navigate to **Check the display attributes of block names**. In the **Input Parameters** pane, select Custom.
- 2 Use the **Block type list** table to include or exclude blocks and masks from the check.
- 3 Click **Apply**.
- 4 Save the configuration. When you run the check using this configuration, the check uses the input parameters that you specified.

### Results and Recommended Actions

Condition	Recommended Action
Name is displayed and obvious from the block appearance.	Hide the block name by selecting <b>Format &gt; Hide Automatic Block Name</b> .
Name is not descriptive. Specifically, the block name is: <ul style="list-style-type: none"> <li>• Not obvious from the block appearance.</li> <li>• The default name appended with an integer.</li> </ul>	Modify the block name to provide descriptive information, or hide the block name by selecting <b>Format &gt; Hide Automatic Block Name</b> .
Name is descriptive and not displayed. Descriptive names are: <ul style="list-style-type: none"> <li>• Provided for blocks that are not obvious from the block appearance.</li> <li>• Not a default name appended with an integer.</li> </ul>	Modify the blocks to show the block name by deselecting <b>Format &gt; Hide Automatic Block Name</b> .
Check does not evaluate my custom blocks and masks.	Use the Model Configuration Editor to specify your custom checks and blocks.

### Capabilities and Limitations

- Runs on library models.

- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0061: Display of block names in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0061: Display of block names

## Check position of Trigger and Enable blocks

**Check ID:** `mathworks.maab.db_0146`

Check the position of Trigger and Enable blocks.

### Description

Locate blocks that define subsystems as conditional or iterative at the top of the subsystem diagram.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Trigger, Enable, and Action Port blocks are not at the top of the subsystem diagram.	Move the Trigger, Enable, and Action Port blocks to the top of the subsystem diagram.
For Each, For Iterator, and While Iterator blocks are not in the same location on the subsystem diagram.	Move the For Each, For Iterator, and While Iterator blocks so they are at a uniform location on the subsystem diagram.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not verify that For Each or For Iterator blocks are uniformly located.
- Runs on library models.
- Analyzes content of library linked blocks.

- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0146: Triggered, enabled, conditional Subsystems in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0146: Block layout in conditional subsystems

## Check for nondefault block attributes

**Check ID:** `mathworks.maab.db_0140`

Identify blocks that use nondefault block parameter values that are not displayed in the model diagram.

### Description

Model diagrams should display block parameters that have values other than default values. One way of displaying this information is by using the **Block Annotation** tab in the Block Properties dialog box. To automatically fix warnings associated with this check, see “Automatically Fix Display of Nondefault Block Parameters”.

To customize the list of nondefault block parameters that are flagged by the check, see “Customize Model Advisor Check for Nondefault Block Attributes”.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Block parameters that have values other than default values, and the values are not in the model display.	In the Block Properties dialog box, use the <b>Block Annotation</b> tab to add block parameter annotations.

### Capabilities and Limitations

- Only customizable for block parameters in `IntrinsicDialogParameters`. See “Common Block Properties” (Simulink)

- JMAAB guideline, Version 5.1 limitation: The check flags masked blocks that display parameter information but do not use block annotations. JMAAB 5.1 guidelines allow masked blocks to display parameter information.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialog boxes.
- Allows exclusions of blocks and charts.

**Tip**

If you use the `add_block` function with `'built-in/blocktype'` as a source block path name for Simulink built-in blocks, some default parameter values of some blocks are different from the defaults that you get if you added those blocks interactively by using Simulink.

**See Also**

- MAAB guideline, Version 3.0: db\_0140: Display of basic block parameters.
- JMAAB guideline, Version 5.1: db\_0140: Display of block parameters
- For a list of block parameter default values, see “Block-Specific Parameters” (Simulink).
- `add_block`.

## Check for unconnected ports and signal lines

**Check ID:** `mathworks.maab.db_0081`

Check whether model has unconnected input ports, output ports, or signal lines.

**Description**

Unconnected inputs should be connected to ground blocks. Unconnected outputs should be connected to terminator blocks.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Blocks have unconnected inputs or outputs.	Connect unconnected lines to blocks specified by the design or to Ground or Terminator blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0081: Unconnected signals, block inputs and block outputs in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0081: Unconnected signals / block

## Check usage of Switch blocks

**Check ID:** `mathworks.maab.jc_0141`

Check usage of Switch blocks.

### Description

Verifies that the Switch block control input (the second input) is a Boolean value and that the block is configured to pass the first input when the control input is nonzero.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The Switch block control input (second input) is not a Boolean value.	Change the data type of the control input to Boolean.



Condition	Recommended Action
The Switch block is not configured to pass the first input when the control input is nonzero.	Set the block parameter <b>Criteria for passing first input</b> to <code>u2 ~= 0</code> .

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking. However, edit-time checking for this check does not verify that the data type of the control input is a Boolean value.

### See Also

- MAAB guideline, Version 3.0: jc\_0141: Use of the Switch block in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0141: Use of the Switch block
- Switch block

## Check usage of Relational Operator blocks

**Check ID:** `mathworks.maab.jc_0131`

Check the position of Constant blocks used in Relational Operator blocks.

### Description

When the relational operator is used to compare a signal to a constant value, the constant input should be the second, lower input.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Relational Operator blocks have a Constant block on the first, upper input.	Move the Constant block to the second, lower input.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Allows exclusions of blocks and charts.

## See Also

- MAAB guideline, Version 3.0: jc\_0131: Use of Relational Operator block in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0131: Usage of Relational Operators

## Check for indexing in blocks

**Check ID:** `mathworks.maab.db_0112`

Check that blocks use consistent vector indexing.

### Description

Check that blocks use consistent vector indexing. When possible, use zero-based indexing to improve code efficiency.

Available with Simulink Check.

This check requires a Stateflow license.

The check verifies consistent indexing for the following objects:

Object	Indexing
<ul style="list-style-type: none"> <li>• Assignment block</li> <li>• For Iterator block</li> <li>• Find block</li> <li>• Multiport Switch block</li> <li>• Selector block</li> </ul>	<ul style="list-style-type: none"> <li>• Zero-based indexing ([0, 1, 2, ...])</li> <li>• One-based indexing ([1, 2, 3,...])</li> </ul>
<ul style="list-style-type: none"> <li>• Stateflow charts with C action language</li> </ul>	Zero-based indexing ([0, 1, 2, ...])
<ul style="list-style-type: none"> <li>• MATLAB Function block</li> <li>• Fcn block</li> <li>• MATLAB System blocks</li> <li>• Truth tables</li> <li>• State transition tables</li> <li>• Stateflow charts with MATLAB action language</li> <li>• MATLAB functions inside Stateflow charts</li> </ul>	One-based indexing ([1, 2, 3,...])

### Results and Recommended Actions

Condition	Recommended Action
Objects in your model use one-based indexing, but can be configured for zero-based indexing.	Configure objects for zero-based indexing.
Objects in your model use inconsistent indexing.	If possible, configure objects for zero-based indexing. If your model contains objects that cannot be configured for zero-based indexing, configure objects for one-based indexing.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: This check cannot be configured to support one-based indexing. (Sub ID: a2)

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0112: Indexing in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0112: Usage of index

## Check usage of tunable parameters in blocks

**Check ID:** `mathworks.maab.db_0110`

Check whether tunable parameters specify expressions, data type conversions, or indexing operations.

### Description

To make a parameter tunable, you must enter the basic block without the use of MATLAB calculations or scripting. For example, omit:

- Expressions
- Data type conversions
- Selections of rows or columns

Supported blocks include:

- Backlash
- Bias
- Combinatorial Logic
- Constant
- Dead Zone
- Derivative
- Discrete-Time Integrator
- Gain

- Hit Crossing
- Initial Condition (IC)
- Integrator
- n-D Lookup Table
- Magnitude-Angle to Complex
- Memory
- Permute Dimensions
- Quantizer
- Rate Limiter
- Rate Transition
- Real-Imag to Complex
- Relay
- Saturation
- Sine
- State-Space
- Switch
- Transport Delay
- Unit Delay
- Transport Delay

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Blocks have a tunable parameter that specifies an expression, data type conversion, or indexing operation.	In each case, move the calculation outside of the block, for example, by performing the calculation with a series of Simulink blocks, or precompute the value as a new variable.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Does not evaluate mask parameters.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: db\_0110: Tunable parameters in basic blocks in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0110: Guidelines for block parameters

## Check usage of Discrete-Time Integrator block

**Check ID:** `mathworks.jmaab.jc_0627`

Check usage of Discrete-Time Integrator block.

### Description

For Discrete-Time Integrator blocks, check:

- Block parameter **Limit output** is selected.
- Saturation limits is defined using a `Simulink.Parameter` or `MPT.Parameter` object whose data type is `auto`.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Block parameter <b>Limit output</b> is cleared	Select the Discrete-Time Integrator block parameter <b>Limit output</b> .
Saturation limit is defined by a Parameter object whose data type is not <code>auto</code>	Change the data type for the Parameter object to <code>auto</code> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.

### See Also

- JMAAB guideline, Version 5.1: jc\_0627: Guideline for using the Discrete-Time Integrator block

## Check usage of fixed-point data type with non-zero bias

**Check ID:** mathworks.jmaab.jc\_0643

Check blocks with whose output signal data type is fixed-point and bias is not zero.

### Description

For blocks that have a fixed-point data type for their output signals, check that block parameter **Bias** is set to 0.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
In the Data Type Assistant, <b>Mode</b> is set to Fixed Point but the value for <b>Bias</b> is not 0.	Change block parameter <b>Bias</b> to 0.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

### See Also

- JMAAB guideline, Version 5.1: jc\_0643: Fixed-point setting

## Check input and output datatype for Switch blocks

**Check ID:** mathworks.jmaab.jc\_0650

Check whether the input and output data types for data ports are the same for switching function blocks

### Description

For Switch, Multiport Switch, and Index Vector blocks, check that the input and output data ports have the same data type.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Input and output data ports have different data types.	Change the input or output data port so the data type is the same for both.

### Capabilities and Limitations

- Analyzes content in all masked subsystems.

### See Also

- JMAAB guideline, Version 5.1: jc\_0650: Block input/output data type with switching function

## Check signs of input signals in product blocks

**Check ID:** `mathworks.jmaab.jc_0611`

Check the sign bit for the input signal data types in product blocks with division operators.

### Description

For product blocks with division operators, check that the same sign bit is used for input signal data types. Sign bits are either signed or unsigned.



## Results and Recommended Actions

Condition	Recommended Action
Input signal data types have different sign bits.	Update the production block so the sign bit for the input signal data types match.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.

## See Also

- JMAAB guideline, Version 5.1: jc\_0611: Input signal for multiplication and division blocks

## Check transition orientations in flow charts

**Check ID:** `mathworks.maab.db_0132`

Check transition orientations in flow charts.

### Description

The following rules apply to transitions in flow charts:

- Draw transition conditions horizontally.
- Draw transitions with a condition action vertically.
- Junctions in flow charts should have a default exit transition.
- Transitions in flow charts should not combine condition and action.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The model includes a transition with a condition that is not drawn horizontally or a transition action that is not drawn vertically.	Modify the model.
Junction does not have a default exit transition	Add a default exit transition to the junction.
Transition has condition and action	Split up condition and action into separate transitions

### Capabilities and Limitations

- MAAB guideline, Version 3.0 limitation: Although db\_0132: Transitions in flow charts has an exception for loop constructs, the check does flag flow charts containing loop constructs if the transition violates the orientation rule.
- JMAAB guideline, Version 5.1 limitation: The check only flags flow charts containing loop constructs if the transition violates the orientation rule.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- MAAB guideline, Version 3.0: db\_0132: Transitions in flow charts in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0132: Transitions in Flow Charts

## Check return value assignments in Stateflow graphical functions

**Check ID:** `mathworks.maab.jc_0511`

Identify graphical functions with multiple assignments of return values in Stateflow charts.

### Description

The return value from a Stateflow graphical function must be set in only one place.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
The return value from a Stateflow graphical function is assigned in multiple places.	Modify the specified graphical function so that its return value is set in one place.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jc\_0511: Setting the return value from a graphical function in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0511: Setting the return value from a graphical function
- “Reusable Functions in Charts” (Stateflow).

## Check for Strong Data Typing with Simulink I/O

**Check ID:** mathworks.maab.db\_0122

Check whether labeled Stateflow and Simulink input and output signals are strongly typed.

**Description**

Strong data typing between Stateflow and Simulink input and output signals is required.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
A Stateflow chart does not use strong data typing with Simulink.	Select the <b>Use Strong Data Typing with Simulink I/O</b> check box for the specified block.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks and charts.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

**See Also**

- MAAB guideline, Version 3.0: db\_0122: Stateflow and Simulink interface signals and parameters in the Simulink documentation.
- JMAAB guideline, Version 5.1: db\_0122: Stateflow and Simulink interface signals and parameters
- “Syntax for States and Transitions” (Stateflow)

**Check Stateflow data objects with local scope**

**Check ID:** `mathworks.maab.db_0125`

Check whether Stateflow data objects with local scope are defined at the chart level or below.

## Description

This check flags Stateflow data whose local scope is not defined at the Chart level or below, regardless of whether the data is used or not.

You must define local data of a Stateflow block on the chart level or below in the object hierarchy. You cannot define local variables on the machine level; however, parameters and constants are allowed at the machine level.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Local data is not defined in the Stateflow hierarchy at the chart level or below.	Define local data at the chart level or below.

## Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag Constant local data defined at the machine level. (Sub ID: b)
  - The check does not flag duplicate data definition and usage within a parent-child relationships. (Sub ID: c)
  - The check does not flag Parameter local data defined at the machine level. (Sub ID: d)
  - The check does not detect if local data has the same name within charts or states that have parent-child relationships.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

## See Also

- MAAB guideline, Version 3.0: db\_0125: Scope of internal signals and local auxiliary variables in the Simulink documentation.

- JMAAB guideline, Version 5.1: db\_0125: Stateflow local data

## Check for MATLAB expressions in Stateflow charts

**Check ID:** `mathworks.maab.db_0127`

Identify Stateflow objects that use MATLAB expressions that are not suitable for code generation.

### Description

Do not use MATLAB functions, instructions, and operators in Stateflow objects.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Stateflow objects use MATLAB expressions.	Replace MATLAB expressions in Stateflow objects.

### Capabilities and Limitations

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- MAAB guideline, Version 3.0: db\_0127: MATLAB commands in Stateflow in the Simulink documentation.

- JMAAB guideline, Version 5.1: db\_0127: Limitation on MATLAB commands in Stateflow blocks
- “Access MATLAB Functions and Workspace Data in C Charts” (Stateflow).

## Check for pointers in Stateflow charts

**Check ID:** mathworks.maab.jm\_0011

Identify pointer operations on custom code variables.

### Description

Pointers to custom code variables are not allowed.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Custom code variables use pointer operations.	Modify the specified chart to remove the dependency on pointer operations.

### Capabilities and Limitations

- Applies only to Stateflow charts that use C as the action language.
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: jm\_0011: Pointers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 5.1: jm\_0011: Pointers in Stateflow

## Check for event broadcasts in Stateflow charts

**Check ID:** `mathworks.maab.jm_0012`

Identify undirected event broadcasts that might cause recursion during simulation and generate inefficient code.

### Description

Event broadcasts in Stateflow charts must be directed.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Event broadcasts are undirected.	Rearchitect the diagram to use directed event broadcasting. Use the send syntax or qualified event names to direct the event to a particular state. Use multiple send statements to direct an event to more than one state.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check cannot be configured to flag event used other than a Stateflow chart output. (Sub ID: a1)
- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: `jm_0012`: Event broadcasts in the Simulink documentation.
- JMAAB guideline, Version 5.1: `jm_0012`: Usage restrictions of events and broadcasting events



- “Broadcast Local Events to Synchronize Parallel States” (Stateflow).

## Check for bitwise operations in Stateflow charts

**Check ID:** mathworks.maab.na\_0001

Identify bitwise operators (&, |, and ^) in Stateflow charts. If you select **Enable C-bit operations** for a chart, only bitwise operators in expressions containing Boolean data types are reported. Otherwise, all bitwise operators are reported for the chart.

### Description

Do not use bitwise operators in Stateflow charts, unless you enable bitwise operations.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Stateflow charts with <b>Enable C-bit operations</b> selected use bitwise operators (&,  , and ^) in expressions containing Boolean data types.	Do not use Boolean data types in the specified expressions.
The Model Advisor could not determine the data types in expressions with bitwise operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.
Stateflow charts with <b>Enable C-bit operations</b> cleared use bitwise operators (&,  , and ^).	To fix this issue, do either of the following: <ul style="list-style-type: none"> <li>• Modify the expressions to replace bitwise operators.</li> <li>• If not using Boolean data types, consider enabling bitwise operations. In the Chart properties dialog box, select <b>Enable C-bit operations</b>.</li> </ul>

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitations:
  - The check does not flag when operators other than "!" are used for logical negation. (Sub ID: a)
  - The check cannot be configured to select between "~=", "!=", and "<>" for inequality operations. (Sub ID: b1, b2, b3)
  - The check does not detect when the "^" and "~" bitwise operators are used for Boolean operations. (Sub ID: c)
- Applies only to charts that use C as the action language.
- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- hisf\_0003: Usage of bitwise operations
- MAAB guideline, Version 3.0: na\_0001: Bitwise Stateflow operators in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0001: Standard usage of Stateflow operators
- “Binary Operations” (Stateflow).

## Check usage of unary minus operations in Stateflow charts

**Check ID:** `mathworks.maab.jc_0451`

Identify unary minus operations applied to unsigned integers in Stateflow objects.

### Description

Do not perform unary minus operations on unsigned integers in Stateflow objects.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Unary minus operations are applied to unsigned integers in Stateflow objects.	Modify the specified objects to remove dependency on unary minus operations.
The Model Advisor could not determine the data types in expressions with unary minus operations.	To allow Model Advisor to determine the data types, consider explicitly typecasting the specified expressions.

## Capabilities and Limitations

- Does not run on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

## See Also

- MAAB guideline, Version 3.0: jc\_0451: Use of unary minus on unsigned integers in Stateflow in the Simulink documentation.
- JMAAB guideline, Version 5.1: jc\_0451: Use of unary minus on unsigned integers

## Check input and output settings of MATLAB Functions

**Check ID:** mathworks.maab.na\_0034

Identify MATLAB Functions that have inputs, outputs or parameters with inherited complexity or data type properties.

### Description

The check identifies MATLAB Functions with inherited complexity or data type properties. A results table provides links to MATLAB Functions that do not pass the check, along with conditions triggering the warning.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
MATLAB Functions have inherited interfaces.	<p>Explicitly define complexity and data type properties for inports, outports, and parameters of MATLAB Function identified in the results.</p> <p>If applicable, using the “MATLAB Function Block Editor” (Simulink), make the following modifications in the “Ports and Data Manager” (Simulink):</p> <ul style="list-style-type: none"> <li>• Change <b>Complexity</b> from Inherited to On or Off.</li> <li>• Change <b>Type</b> from Inherit: Same as Simulink to an explicit type.</li> <li>• Change <b>Size</b> from -1 (Inherited) to an explicit size.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- MAAB guideline, Version 3.0: na\_0034: MATLAB Function block input/output settings in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0034: MATLAB Function block input/output settings

## Check MATLAB code for global variables

**Check ID:** mathworks.maab.na\_0024

Check for global variables in MATLAB code.

**Description**

Verifies that global variables are not used in any of the following:

- MATLAB code in MATLAB Function blocks
- MATLAB functions defined in Stateflow charts
- Called MATLAB functions

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
Global variables are used in one or more of the following: <ul style="list-style-type: none"> <li>• MATLAB code in MATLAB Function blocks</li> <li>• MATLAB functions defined in Stateflow charts</li> <li>• Called MATLAB functions</li> </ul>	Replace global variables with signal lines, function arguments, or persistent data.

**Capabilities and Limitations**

- Runs on library models.
- Does not analyze content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

**See Also**

- MAAB guideline, Version 3.0: na\_0024: Global Variables in the Simulink documentation.
- JMAAB guideline, Version 5.1: na\_0024: Shared data in MATLAB functions

**Check usage of Memory and Unit Delay blocks**

**Check ID:** mathworks.jmaab.jc\_0623

Checks Memory and Unit Delay blocks with inappropriate sample time.

### Description

Identifies the Memory blocks with discrete sample time and Unit Delay blocks with a nondiscrete sample time.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Memory blocks have sample time that is not continuous.	Use Unit Delay block instead of Memory block.
Unit Delay blocks have nondiscrete sample time.	Use Memory block instead of Unit Delay block.

### Capabilities and Limitations

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0623: Use of continuous-time delay blocks and discrete-time blocks

### Check block orientation

**Check ID:** mathworks.jmaab.jc\_0110

Checks blocks with changed orientation.

## Description

Identifies the blocks that are reversed or with rotated orientation. This check excludes Unit Delay or Delay blocks.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Some blocks in the model have rotated or reversed orientation.	Flip or rotate these blocks to be oriented toward the right.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.

## See Also

- JMAAB guideline, Version 5.1: `jc_0110`: Direction of block

## Check usage of internal transitions in Stateflow states

**Check ID:** `mathworks.jmaab.jc_0763`

## Description

Identifies the Stateflow states that uses multiple internal transitions.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

	Condition	Recommended Action
a1	One or more Stateflow states have multiple internal transitions.	Remodel to avoid the use of multiple internal transitions.
a2	One or more Stateflow states have multiple internal transitions not placed in order of execution.	Consider placing internal transitions from top to bottom in the order of execution.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0763: Usage of multiple internal transitions

## Check usage of transition conditions in Stateflow transitions

**Check ID:** mathworks.jmaab.jc\_0772

### Description

Identifies the transitions sourced from a state and unconditional Stateflow transitions with higher priority than conditional transitions.

Available with Simulink Check.

This check requires a Stateflow license.



## Results and Recommended Actions

Condition	Recommended Action
Stateflow transitions found with higher priority than conditional transitions.	Change the execution order of the transitions or add an execution condition.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0772: Execution order and transition conditions of transition lines

## Check usable characters for signal names and bus names

**Check ID:** mathworks.jmaab.jc\_0222

### Description

Checks the signal and bus names in the model.

## Results and Recommended Actions

Condition	Recommended Action
The file name starts with a number.	Rename the file.
The file name starts with an underscore ("_").	Rename the file.
The file name ends with an underscore ("_").	Rename the file.

Condition	Recommended Action
The file extension contains one (or more) underscores.	Change the file extension.
The file name has consecutive underscores.	Rename the file.
The file name contains more than one dot (".").	Rename the file.
The file name contains illegal characters.	Rename the file. Allowed characters are a-z, A-Z, 0-9. and underscore (_).

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0222`: Usable characters for signal/bus names

## Check usable characters for parameter names

**Check ID:** `mathworks.jmaab.jc_0232`

### Description

Checks the parameter names in the model.

### Results and Recommended Actions

Condition	Recommended Action
The parameter name starts with a number.	Rename the parameter.
The parameter name starts with an underscore ("_").	Rename the parameter.

Condition	Recommended Action
The parameter name ends with an underscore ("_").	Rename the parameter.
The parameter name has consecutive underscores.	Rename the parameter.
The parameter name contains illegal characters.	Rename the parameter. Allowed characters are a-z, A-Z, 0-9. and underscore (_).

### Capabilities and Limitations

- Does not analyze content in masked subsystems.

### See Also

- JMAAB guideline, Version 5.1: jc\_0232: Usable characters for parameter names

## Check length of model file name

**Check ID:** mathworks.jmaab.jc\_0241

### Description

Checks if the length of the model file name adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Model file name does not adhere to the length restriction of 63 characters.	Modify the model file name so that the length of the file name is not more than 63 characters.

### Capabilities and Limitations

- Does not analyze content of library-linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0241: Length restrictions for model file names

**Check length of folder name at every level of model path****Check ID:** `mathworks.jmaab.jc_0242`**Description**

Checks the length of the folder names at every level of the model path to see if all the folders in the path adhere to the maximum length restriction of 63 characters.

Available with Simulink Check.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Length of the folder names at every level of the model path does not adhere to the length restriction of 63 characters.	Modify the folder names that do not meet the length restriction of 63 characters throughout the path.

**Capabilities and Limitations**

- Does not analyze content of library-linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0242: Length restrictions for folder names

**Check length of subsystem names****Check ID:** `mathworks.jmaab.jc_0243`**Description**

Checks if the length of the subsystem names in the model adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Subsystem names in the model does not adhere to the length restriction of 63 characters.	Modify the subsystem block names so that the length of the subsystem name is not more than 63 characters.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- JMAAB guideline, Version 5.1: `jc_0243`: Length restrictions for subsystem names

## Check length of Inport and Outport names

**Check ID:** `mathworks.jmaab.jc_0244`

## Description

Checks if the length of the inport and outport names adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Inport or output block names in the Model does not adhere to the length restriction of 63 characters.	Modify the inport or the output block names so that the length of the block name is not more than 63 characters.

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library-linked blocks.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- JMAAB guideline, Version 5.1: `jc_0244`: Length restrictions for Inport and Output names

## Check length of signal and bus names

**Check ID:** `mathworks.jmaab.jc_0245`

### Description

Checks if the length of the signal or bus names adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Signal or bus name in the model does not adhere to the length restriction of 63 characters.	Modify the signal or the bus names in the model so that the length of the names is not more than 63 characters.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

## See Also

- JMAAB guideline, Version 5.1: `jc_0245`: Length restrictions for signal and bus names

## Check length of parameter names

**Check ID:** `mathworks.jmaab.jc_0246`

## Description

Checks if the length of the parameter names in the model adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Parameter names in the model does not adhere to the length restriction of 63 characters.	Modify the parameter names so that the length of the parameter names is not more than 63 characters.

### Capabilities and Limitations

- Does not analyze content of library-linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0246: Length restrictions for parameter names

## Check length of block names

**Check ID:** `mathworks.jmaab.jc_0247`

### Description

Checks if the length of the block names in the model adheres to the maximum length restriction of 63 characters.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Block names in the model does not adhere to the length restriction of 63 characters.	Modify the block names so that the length of the block names is not more than 63 characters.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.



**See Also**

- JMAAB guideline, Version 5.1: jc\_0247: Length restrictions for block names

**Check if blocks are shaded in the model**

**Check ID:** mathworks.jmaab.jc\_0604

**Description**

Checks if block shading is used in the model.

**Results and Recommended Actions**

Condition	Recommended Action
<b>Block shading</b> is turned <b>on</b> .	Consider turning <b>off</b> the <b>DropShadow</b> property in blocks for better readability.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0604: Block shading

**Check operator order of Product blocks**

**Check ID:** mathworks.jmaab.jc\_0610

**Description**

Checks the operator order of product blocks.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0610_a: Check first input to Product block	Improper usage of operator order of Product blocks.	Change the first input in Product block to multiplication <sup>(*)</sup> .
jc_0610_b: Check number of inputs to Product blocks	Product blocks have invalid number of inputs.	Consider having not more than two inputs for the Product blocks.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking.

### See Also

- JMAAB guideline, Version 5.1: jc\_0610: Operator order for multiplication and division blocks

## Check icon shape of Logical Operator blocks

**Check ID:** `mathworks.jmaab.jc_0621`

**Description**

Checks icon shape of Logical Operator blocks. Icon shape of Logical Operator should be rectangular.

**Results and Recommended Actions**

Condition	Recommended Action
Improper setting of icon shape for Logical Operator blocks.	Change the icon shape of Logical Operator blocks to rectangular for readability.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

**Edit-Time Checking**

This check is supported by edit-time checking.

**See Also**

- JMAAB guideline, Version 5.1: `jc_0621`: Guideline for using the Logical Operator block

**Check if tunable block parameters are defined as named constants**

**Check ID:** `mathworks.jmaab.jc_0645`

**Description**

Checks if the tunable block parameters are defined as named constants.

### Results and Recommended Actions

Condition	Recommended Action
Improper usage of tunable block parameter values.	Change the tunable block parameter literal values to named constants.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0645`: Parameter definition for calibration

## Check default/else case in Switch Case blocks and If blocks

**Check ID:** `mathworks.jmaab.jc_0656`

### Description

Checks the **default/else** case in Switch Case blocks and If blocks.

### Results and Recommended Actions

Condition	Recommended Action
Improper usage of Switch and If blocks.	Consider setting the <b>default/else</b> case option in Switch Case blocks and If blocks to <b>on</b> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0656: Usage of Conditional Control block

## Check for parentheses in Fcn block expressions

**Check ID:** mathworks.jmaab.jc\_0622

### Description

Checks the use of parentheses in Fcn block expressions. Parentheses must be used to define the operator precedence.

### Results and Recommended Actions

Condition	Recommended Action
Improper usage of Fcn block expressions.	Resolve the operator precedence in Fcn block expressions by adding parentheses.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0622: Guideline for using the Fcn block

## Check undefined initial output for conditional subsystems

**Check ID:** `mathworks.jmaab.jc_0640`

### Description

Checks that the initial output value for all Outports and Merge blocks connected to a Conditional subsystem are explicitly defined.

### Results and Recommended Actions

Condition	Recommended Action
The initial output for all Outports and Merge blocks connected to a Conditional subsystem are not explicitly defined.	For a Conditional subsystem, explicitly define the initial output value for all Outports and Merge blocks connected to the Conditional subsystem.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0640`: Initial value settings for Outport blocks in conditional subsystems

## Check usage of the Saturation blocks

**Check ID:** `mathworks.jmaab.jc_0628`

### Description

This check identifies:

- The Saturation or Saturation Dynamic blocks with any type casting operations. The check compares that the compiled input and output data types match or checks that

**Output data type** is set to **Inherit: Same as input** and **Inherit: Same as second input** for Saturation and Saturation Dynamic blocks respectively.

- If the **upper limit** is set to the maximum value of the output data type (intmax, realmax).
- If the **lower limit** is set to the minimum value of the output data type (intmin, -realmax).

### Results and Recommended Actions

Condition	Recommended Action
The input and output data types are different.	Make sure that the <b>Output data type</b> is set to <b>Inherit: Same as input</b> and <b>Inherit: Same as second input</b> for Saturation and Saturation Dynamic blocks respectively.
The upper limit and lower limit values of the blocks are not set to adhered values.	<ul style="list-style-type: none"> <li>• Set the <b>upper limit</b> of the output data type to less than the maximum value.</li> <li>• Set the <b>lower limit</b> of the output data type to less than the minimum value.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0628: Guideline for using the Saturation Block

### Check type setting by data objects

**Check ID:** mathworks.jmaab.jc\_0644

### Description

Identifies the blocks in Simulink that violate the type setting if the signal objects are used (if signal data type is set in signal object, then it must not be set on the block side).

This check exempts:

- Data type conversion block.
- Type setting using **fixdt**.
- Double and Boolean types.
- Reusable internal part of a function (atomic subsystem).
- Block output data type set to **Inherit via backpropagation**.

### Results and Recommended Actions

Condition	Recommended Action
Signal data type is set to different types in signal objects and in the block.	Set the output data type of the blocks either to <b>auto</b> or <b>Inherit via back propagation</b> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to **graphical**.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to **on**.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0644: Guideline for type setting

## Check prohibited comparison operation of logical type signals

**Check ID:** mathworks.jmaab.jc\_0655



## Description

Identifies the Boolean type transitions in Stateflow charts that use either comparison with numbers or logical values (true or false), or use negation operators (! or ~) variably in the model.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Negation operators are used variably in the model.	The negation operator must be used consistently in the model.
Boolean type transactions are compared with numbers or logical values (true or false).	Make sure that the Boolean type transactions are not compared with numbers or logical values.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0655: Prohibition of logical value comparison in Stateflow

## Check usage of Stateflow comments

**Check ID:** mathworks.jmaab.jc\_0738

## Description

Identifies the comments that are nested or contain newline(s) in the middle in Stateflow.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0738_a: If the action language is "C", comment nesting must not be used	On or more comments in Stateflow objects are nested.	Comments in Stateflow must not be nested.
jc_0738_b: If the action language is "C", comments must not contain newline(s) in the middle	One or more comments in Stateflow objects contain newline(s) in the middle.	Comments in Stateflow must not contain newline(s) in the middle.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0738: Usage of Stateflow comments

## Check updates to variables used in state transition conditions

**Check ID:** `mathworks.jmaab.jc_0741`

### Description

Checks if the variables used in state transition conditions perform an update by "during" state action type.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more variables in the state transition condition performs an update by "during" state action type.	Make sure that the variables used in state transition conditions do not perform an update by "during" state action type.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0741`: Timing to update data used in state chart transition conditions

## Check for unexpected backtracking in state transitions

**Check ID:** `mathworks.jmaab.jc_0751`

### Description

Checks unexpected backtracking in state transitions. Configuration parameter for **Unexpected backtracking (SFUnexpectedBacktrackingDiag)** must be set to **error**.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Backtracking is undetected during the state transition.	Set configuration parameter for <b>Unexpected backtracking (SFUnexpectedBacktrackingDiag)</b> to <b>error</b> .

### Capabilities and Limitations

- Runs on library models.
- Does not analyze content of library-linked blocks.
- Does not analyze content in masked subsystems.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0751: Backtracking prevention in state transition

## Check usage of parentheses in Stateflow transitions

**Check ID:** mathworks.jmaab.jc\_0752

### Description

Checks if a new line is started before and after parentheses for condition actions in Stateflow transitions.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Condition actions in Stateflow transitions are written beside parenthesis.	Start new line before and after parentheses for condition actions in Stateflow transitions.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0752`: Format of condition action in transition label

## Check condition actions and transition actions in Stateflow

**Check ID:** `mathworks.jmaab.jc_0753`

### Description

Checks if the use of condition actions or transition actions are uniform within the same chart.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Condition actions and transition actions are mixed within the same chart.	Use of condition actions or transition actions must be uniform within the same chart.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check cannot be configured to only allow condition actions within a Stateflow chart. (Sub ID: a1)
- This check does not flag the transition actions used in state charts.
- Runs on library models.

- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Allows exclusions of blocks and charts.

**See Also**

- JMAAB guideline, Version 5.1: `jc_0753`: Condition actions and transition actions in Stateflow

**Check starting point of internal transition in Stateflow**

**Check ID:** `mathworks.jmaab.jc_0760`

**Description**

Identifies if in all state charts and flow charts, internal transitions from state boundaries must start from the left edge of the state.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
Starting point of one or more internal transitions from state boundaries of state charts or flow charts does not start from the left edge of the state.	Make sure that in all the state charts and flow charts, internal transitions from state boundaries must start from the left edge of the state.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.

- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0760: Starting point of internal transition

## Check prohibited combination of state action and flow chart

**Check ID:** mathworks.jmaab.jc\_0762

### Description

Checks if state actions within states and flow chart statements are used in combination.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Stateflow states combine state action and flow chart.	Separate state actions and flow chart statements into different states.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0762: Prohibited of state action and flow chart combination

## Check usage of Lookup Tables

**Check ID:** mathworks.jmaab.jc\_0626

Checks for the correct parameter settings in Lookup Tables to prevent unexpected results.

### Description

Checks n-D Lookup (1-D,2-D, and n-D) Tables for the following parameters to ensure that the values adhere to the corresponding recommendations.

- InterpMethod
- ExtrapMethod
- UseLastTableValue

Checks Dynamic Lookup Tables for the parameter **LookUpMeth** and ensures that the values adhere to the recommendation.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The parameter <b>InterpMethod</b> in the n-D lookup table is set to <b>Cubic spline</b> .	Update the parameter settings with one of the following recommended values <ul style="list-style-type: none"> <li>• Flat</li> <li>• Nearest</li> <li>• Linear point-slope</li> <li>• Linear Lagrange.</li> </ul>
The parameter <b>ExtrapMethod</b> in the n-D lookup table is set to <b>Cubic spline</b> or <b>Linear</b> .	Set the parameter to the recommended value <b>Clip</b> .
The parameter <b>UseLastTableValue</b> in the n-D lookup table is set to <b>off</b> .	Set the parameter to the recommended value <b>on</b> .



Condition	Recommended Action
The parameter <b>LookUpMeth</b> in the dynamic lookup table is set to other than <b>Interpolation-Use End Values</b> .	Set the parameter to the recommended value <b>Interpolation-Use End Values</b> .

### Capabilities and Limitations

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0626: Guideline for using the Lookup Table block

## Check Signed Integer Division Rounding mode

**Check ID:** mathworks.jmaab.jc\_0642

### Description

Identifies blocks whose parameter **Integer Rounding Mode** is set to **Simplest** when the configuration parameter **Signed Integer Division Rounds** is set to **Undefined**.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
If the parameter <b>Integer Rounding Mode</b> of the listed blocks is set to <b>Simplest</b> when the parameter <b>Signed Integer Division Rounds</b> is set to <b>Undefined</b> .	Set the parameter <b>Signed Integer Division Round</b> to a value that describes the rounding behavior of your production target or changing the <b>Integer Rounding Mode</b> of the listed blocks to a value other than <b>Simplest</b> .

### Capabilities and Limitations

- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0642: Integer rounding mode setting

## Check usage of Merge block

**Check ID:** `mathworks.jmaab.jc_0659`

Checks if there are any blocks present in between a conditional subsystem and a merge block.

### Description

Merge blocks must have direct connections from conditionally executed subsystems. While using a Merge block take the following into consideration:

- No blocks must be present in between the Merge and Conditionally executed subsystem blocks, including a virtual subsystem that does not affect the function of Merge block.
- The Merge block can be nested inside any number of subsystems, if the preceding condition is satisfied.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more blocks inserted in between a Merge and a Conditional Subsystem block.	Make direct connections from Conditional Subsystem blocks to Merge blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0659: Usage restrictions of signal lines inputted to Merge block

## Check for unused data in Stateflow Charts

**Check ID:** mathworks.jmaab.jc\_0700

Checks the state of the parameter **Unused data, events, messages and functions**.

### Description

Identifies if the parameter **Unused data, events, messages and functions** is set to **None**. Unused data and events cannot exist in the Stateflow block.

### Results and Recommended Actions

Condition	Recommended Action
In a Stateflow block, the parameter <b>Unused data, events, messages and functions</b> is set to <b>None</b> .	Make sure to set the parameter to either <b>Warning</b> or <b>Error</b> .

### Capabilities and Limitations

- Runs on library models.

### See Also

- JMAAB guideline, Version 5.1: jc\_0700: Unused data in Stateflow block

## Check usable number for first index

**Check ID:** `mathworks.jmaab.jc_0701`

### Description

Identifies if the first index of arrays in Stateflow is not set to either **0** or **1**.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0701_a1: Check if first index of Stateflow data is set to 0	One or more Stateflow data have first index set to a value other than 0.	Make Sure to set the first index value to 0.
jc_0701_a2: Check if first index of Stateflow data is set to 1	One or more Stateflow data have first index set to a value other than 1.	Make Sure to set the first index value to 1.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0701`: Usable number for first index

## Check execution timing for default transition path

**Check ID:** mathworks.jmaab.jc\_0712

### Description

Identifies the state of the parameter **Execute (enter) Chart At Initialization**. This parameter requires many other considerations to produce consistent results.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The parameter <b>Execute (enter) Chart At Initialization</b> is selected.	Make sure to clear the selection.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0712: Execution timing for default transition path

## Check for parallel Stateflow state used for grouping

**Check ID:** mathworks.jmaab.jc\_0721

### Description

Parallel states must not be used for the purpose of grouping that is the substates of parallel states must not be parallel states.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
Parallel states are only for grouping.	Substates of the parallel states must not be parallel (do not use for grouping).

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0721: Usage of parallel states

## Check scope of data in parallel states

**Check ID:** mathworks.jmaab.jc\_0722

### Description

The scope of local variables must be set as restricted to one parallel state unless that same data is required by two or more parallel states.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
The scope of Stateflow data (local variables) is not restricted to a parallel state when the same data is not required by multiple parallel states.	Restrict the scope of Stateflow data (local variables) to only one parallel state.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of charts.

## See Also

- JMAAB guideline, Version 5.1: `jc_0722`: Local data definition in parallel states

## Check uniqueness of State names

**Check ID:** `mathworks.jmaab.jc_0730`

### Description

State names must be unique in charts, with the exception of Atomic subcharts. I.e. Atomic Subcharts are treated as different container so they can share State Names with other states outside of the subchart.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
In a Stateflow chart, two or more Stateflow states have the same name.	Rename the Stateflow states so that there are no identical names in the Stateflow chart.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0730`: Unique state name in Stateflow blocks

## Check usage of State names

**Check ID:** `mathworks.jmaab.jc_0731`

Checks for slashes (/) in the state names.

### Description

Checks if slashes (/) are included in state names. After the state name is defined, add a new line for describing any executable statements. A slash (/) is required only when describing executable statements in continuation after state names.

Available with Simulink Check.



## Results and Recommended Actions

Condition	Recommended Action
Slash is in the state name.	Remove the slash from the state name and make sure to start a new line for any executable statements.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0731: State name format

## Check uniqueness of Stateflow State and Data names

**Check ID:** mathworks.jmaab.jc\_0732

### Description

Checks if in a single Stateflow chart, the Stateflow Data name and the Stateflow State name are the same.

Available with Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
If the Stateflow Data name and the Stateflow State name have the same name in a Stateflow Chart.	Rename either of the Stateflow Data name or Stateflow State name to not to be identical names.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0732: Distinction between state names, data names, and event names

## Check repetition of Action types

**Check ID:** mathworks.jmaab.jc\_0734

Identifies repeated Action types in a Stateflow state.

### Description

The action types (entry (en), during (du), exit (ex), en, du:, du, ex:, en, ex:, en, du, ex: ) must not be described two or more times in a Stateflow state.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more action types is used multiple times in a Stateflow state.	Merge the actions types so that each of the action types is defined only once in a Stateflow state.

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.

- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0734`: Number of state action types

## Check indentation of code in Stateflow states

**Check ID:** `mathworks.jmaab.jc_0736`

Check for uniform indentation of label Strings in Stateflow States and Transitions.

### Description

Checks if the indentations in the Stateflow blocks are described uniformly and adhere to the following recommendations:

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
<code>jc_0736_a</code> : Check for uniform indentation of action language in Stateflow states	One or more Stateflow states in the model do not have uniform indentation.	Consider deleting the blank spaces before state action types and adding the exact number of single-byte spaces as defined in the input parameter before executable statements.

Guideline Sub ID	Condition	Recommended Action
jc_0736_b: Check for uniform spacing of transition action types	One or more Stateflow transitions in the model do not have uniform spacing.	Consider not adding blank spaces before '[' of a transition condition, '{' of a transition action and '/' of the event in a transition.
jc_0736_c: Check for uniform spacing of transition actions	One or more Stateflow transitions in the model do not have uniform spacing.	Consider adding the exact number of single-byte spaces as defined in the input parameter after the '/' of a transition action.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to on.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to graphical.
- Allows exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0736: Uniform indentations in Stateflow blocks

## Check placement of Label String in Transitions

**Check ID:** mathworks.jmaab.jc\_0770

### Description

Checks the placement of the Stateflow Transition labels. The Stateflow signal label must always be at the origin of the signal or at the midpoint of the signal transition line.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0770_a1: Transition labels should be placed near the point of origin of the transition	One or more Stateflow transitions do not have label string placed near the origin of the transition.	Make sure to place the Stateflow labels near the point of origin of the transition.
jc_0770_a2: Transition labels should be placed near center of the transition	One or more Stateflow transitions do not have label string placed near the center of the transition.	Make sure to place the Stateflow labels near the mid-point (center) of the transition.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0770: Transition label layout

## Check duplication of Simulink data names

**Check ID:** mathworks.jmaab.jc\_0791

### Description

Checks for a duplicate definition of data names. Simulink data names must be unique across the base workspace, model workspace, and data dictionary.

### Results and Recommended Actions

Condition	Recommended Action
One or more Simulink data names have duplicate definitions.	Make sure that all the defined Simulink data names are unique.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0791: Duplicate data names

## Comparing floating point types in Simulink

**Check ID:** `mathworks.jmaab.jc_0800`

### Description

Checks if equivalence comparison is done on floating-point numbers.

### Results and Recommended Actions

Condition	Recommended Action
One or more instances of equivalence comparison for floating-point numbers is observed.	Make sure to avoid the use of equivalence comparisons for floating-point numbers.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0800: Comparing floating-point types in Simulink

**Check for avoiding algebraic loops between subsystems**

**Check ID:** mathworks.jmaab.jc\_0653

**Description**

Checks the placement of the Delay blocks in algebraic loops between subsystems.

**Results and Recommended Actions**

Condition	Recommended Action
Delay blocks in feedback loops are placed inside subsystem.	Delay blocks in feedback loops must be placed outside of their parent subsystem.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0653: Delay block layout in feedback loops

**Check default transition placement in Stateflow charts**

**Check ID:** mathworks.jmaab.jc\_0531

**Description**

Checks the default transition placement in Stateflow charts.

**Results and Recommended Actions**

Condition	Recommended Action
Parallel states in the model have default transitions.	Default transitions must not be used for parallel states.
One or more Stateflow states in the model do not have default transitions.	Make sure that there is a default transition at every level.
One or more Stateflow states in the model have multiple default transitions at the same level.	Multiple default transitions must not be included in the same level.
One or more default transitions in the model are not connected to the top state or junction.	Make sure that the default transitions are directly connected to the upper part of the state or junction.
One or more destination states or junctions of default transitions are not on the top.	Make sure that the transition destination state or transition destination junction for the default transition is positioned in the far upper left within the same level.
One or more default transitions in the model exceeds state boundaries.	Make sure that the default transition does not exceed state boundaries.
The parameter <b>No unconditional default transitions</b> in <b>Configuration &gt; Diagnostics &gt; Stateflow</b> is not set to <b>error</b> .	Set the parameter <b>No unconditional default transitions</b> to <b>error</b> .

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0531: Default transition



## Check settings for data ports in Multiport Switch blocks

**Check ID:** mathworks.jmaab.jc\_0630

### Description

Identifies the Multiport Switch blocks that violate data port settings.

### Results and Recommended Actions

Condition	Recommended Action
<b>Switch block</b> or a <b>Multiport Switch block</b> in the model have fewer than two data ports.	Make sure that the <b>Switch blocks</b> or a <b>Multiport Switch block</b> in the model have data ports that are greater than two.
Data type of <b>control port</b> on the <b>Multiport Switch</b> is not set to <b>unsigned integer</b> .	Change the data type of the <b>control port</b> on the <b>Multiport Switch</b> to <b>unsigned integer</b> .
<b>Multiport Switch block</b> setting <b>Data port for default case</b> is not set to <b>Additional data port</b> .	Set the Multiport Switch block setting <b>Data port for default case</b> to <b>Additional data port</b> .
Switch block setting <b>Diagnostics for default case</b> to <b>None</b> .	Set the Switch block setting <b>Diagnostics for default case</b> to <b>None</b> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### Edit-Time Checking

This check is supported by edit-time checking. However, edit-time checking for this check does not verify compliance with JMAAB guideline (v5.0) jc\_0630 Sub ID b: Set the Input to the [Multiport Switch] control port to an unassigned integer.

### See Also

- JMAAB guideline, Version 5.1: jc\_0630: Usage of Multiport Switch block

## Check for usage of text inside states

**Check ID:** `mathworks.jmaab.jc_0739`

### Description

Identifies the Stateflow states with text exceeding the boundary of the state.

Available with Simulink Check.

This check requires a Stateflow license.

### Results and Recommended Actions

Condition	Recommended Action
One or more Stateflow states in the model have text exceeding the boundary of the state.	Make sure that the text inside the Stateflow state does not go outside the boundary of the state.

### Capabilities and Limitations

- This check flags all the Stateflow states in the model that have a newline character in the text. This flag happens even if the text inside the Stateflow state is contained within the boundary of the state.
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: `jc_0739`: Guidelines for describing texts inside states

## Check for unconnected objects in Stateflow Charts

**Check ID:** `mathworks.jmaab.jc_0797`

**Description**

Checks for unconnected objects in Stateflow Charts and Identifies dangling transitions and unconnected Stateflow States and Junctions in Stateflow Charts.

Available with Simulink Check.

This check requires a Stateflow license.

**Results and Recommended Actions**

Condition	Recommended Action
One or more unconnected transitions, states, or junctions are found in Stateflow Charts.	Remodel to connect the dangling transitions and unconnected states and junctions.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of blocks and charts.
- Analyzes content of library linked blocks.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0797: Unconnected transition lines / states / connective junctions

**Check Stateflow chart action language**

**Check ID:** mathworks.jmaab.jc\_0790

**Description**

Checks if the action language of Stateflow charts is set to C.

### Results and Recommended Actions

Condition	Recommended Action
Action language of one or more stateflow charts is not set to C.	Set all Stateflow charts action language to C.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0790: Chart action language

## Check usable characters for Stateflow data names

**Check ID:** mathworks.jmaab.jc\_0795

### Description

Checks if the Stateflow data names in the model are using acceptable characters.

### Results and Recommended Actions

Condition	Recommended Action
The Stateflow data name starts with an underscore ("_").	Rename the Stateflow data.
The Stateflow data name ends with an underscore ("_").	Rename the Stateflow data.
The Stateflow data name has consecutive underscores.	Rename the Stateflow data.
The Stateflow data name is solely a reserved MATLAB word.	Consider using a different name for Stateflow data.

### Capabilities and Limitations

- JMAAB guideline, Version 5.1 limitation: The check does not flag when a Name starts with underscore. (Sub ID: a)
- Runs on library models.
- Analyzes content in all masked subsystems.
- Allows exclusions of charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0795: Usable characters for Stateflow data names

## Check length of Stateflow data name

**Check ID:** mathworks.jmaab.jc\_0796

### Description

Checks if the length of Stateflow data names are within the limit of 0 - 63 characters.

### Results and Recommended Actions

Condition	Recommended Action
Length of Stateflow data names are not within the limit of 0 - 63 characters.	Consider using a different name for each Stateflow data name.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of charts.
- Analyzes content of library linked blocks.

### See Also

- JMAAB guideline, Version 5.1: jc\_0796: Length restriction for Stateflow data names

### Check signal line labels

**Check ID:** `mathworks.jmaab.jc_0008`

Check the labeling on signal lines.

#### Description

Use a label to identify:

- Signals originating from the following blocks (the block icon exception noted below applies to all blocks listed, except Inport, Bus Selector, Demux, and Selector):
  - Bus Selector block (tool forces labeling)
  - Chart block (Stateflow)
  - Constant block
  - Data Store Read block
  - Demux block
  - From block
  - Inport block
  - Selector block
  - Subsystem block

---

**Block Icon Exception** If a signal label is visible in the display of the icon for the originating block, you do not have to display a label for the connected signal unless the signal label is required elsewhere due to a rule for signal destinations.

---

- Signals connected to one of the following destination blocks (directly or indirectly with a basic block that performs an operation that is not transformative):
  - Bus Selector block (tool forces labeling)
  - Chart block (Stateflow)
  - Data Store Write block
  - Goto block
  - Mux block
  - Outport block

- Subsystem block
- Any signal of interest.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Signals coming from Bus Selector, Chart, Constant, Data Store Read, Demux, From, Inport, or Selector blocks are not labeled.	Label the signal.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0008: Definition of Signal labels
- “Signal Names and Labels” (Simulink).

## Check for propagated signal labels

**Check ID:** mathworks.jmaab.jc\_0009

Check for propagated labels on signal lines.

### Description

You should propagate a signal label from its source rather than enter the signal label explicitly (manually) if the signal originates from:

- An Inport block in a nested subsystem. However, if the nested subsystem is a library subsystem, you can explicitly label the signal coming from the Inport block to accommodate reuse of the library block.
- A basic block that performs a nontransformative operation.

- A Subsystem or Stateflow Chart block. However, if the connection originates from the output of an instance of the library block, you can explicitly label the signal to accommodate reuse of the library block.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model includes signal labels that were entered explicitly, but should be propagated.	Use the open angle bracket (<) character to mark signal labels that should be propagated and remove the labels that were entered explicitly.

### Capabilities and Limitations

- Does not run on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems that have no workspaces and no dialogs.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0009: Signal name propagation
- “Signal Names and Labels” (Simulink).

## Check trigger signal names

**Check ID:** mathworks.jmaab.jc\_0281

### Description

Identify trigger blocks where the origin of the trigger signal and the destination have dissimilar names.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.



**Results and Recommended Actions**

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
jc_0281_a1: Check names of the origin of the trigger signal and the conditional input block	The name of the block at the origin of the trigger signal and the conditional input block name at the destination are dissimilar.	The name of the block at the origin of the trigger signal and the conditional input block name at the destination must include the same name.
jc_0281_a2: Check names of the trigger signal and the conditional subsystem	The name of the block at the origin of the trigger signal and the conditional subsystem name at the destination are dissimilar.	The name of the block at the origin of the trigger signal and the conditional subsystem name at the destination must include the same name.
jc_0281_a3: Check names of the origin of the trigger signal and the conditional input block	The name of the trigger signal and the conditional input block name at the destination are dissimilar.	The name of the trigger signal and the conditional input block name at the destination must include the same name.
jc_0281_a4: Check names of the trigger signal and the conditional subsystem	The name of the trigger signal and the conditional subsystem name at the destination must include the same name.	The name of the trigger signal and the conditional subsystem name at the destination are dissimilar.
jc_0281_b1: Check names of the origin of the trigger signal and the Stateflow block event	The name of the block at the origin of the trigger signal and the Stateflow block event name at the destination are dissimilar.	The name of the block at the origin of the trigger signal and the Stateflow block event name at the destination must include the same name.
jc_0281_b2: Check names of the origin of the trigger signal and the Chart name	The name of the block at the origin of the trigger signal and the Chart name at the destination are dissimilar.	The name of the block at the origin of the trigger signal and the Chart name at the destination must include the same name.

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
jc_0281_b3: Check names of trigger signal and the Stateflow block event	The name of the trigger signal and the Stateflow block event name at the destination are dissimilar.	The name of the trigger signal and the Stateflow block event name at the destination must include the same name.
jc_0281_b4: Check names of the trigger signal and the Chart	The name of the trigger signal and the Chart name at the destination are dissimilar.	The name of the trigger signal and the Chart name at the destination must include the same name.

Note: The Guideline Sub IDs A1 and B1 are selected by default.

### **Capabilities and Limitations**

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

### **See Also**

- JMAAB guideline, Version 5.1: jc\_0281: Trigger signal names

## **Check for consistency in model element names**

**Check ID:** mathworks.jmaab.jc\_0602

### **Description**

Checks if the model elements connected to a signal are following consistent naming.

The following names must be matched exactly when directly connected with signal lines:

- Inport block name
- Outport block name
- Structural subsystem input port label name

- Structural subsystem output port label name
- From tag name
- Goto tag name
- Signal line signal name

**Exception 1:** The name of a signal line connected to one of the below subsystems can have a different name to that of the subsystem port label name:

- Reusable subsystems
- Subsystems linked to a library

**Exception 2:** If a combination of Inport blocks, Outport blocks, and other blocks have the same block name, use a suffix or prefix for the Inport and Outport blocks for consistent naming. This can be configured through Model Advisor Configuration Editor.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more model elements are not consistent with the connected signal name.	Consider renaming the deviating model elements to match the signal name or to be consistent with Inport/Outport blocks.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0602: Consistency in model element names

## Check for sample time setting

**Check ID:** mathworks.jmaab.jc\_0641

### Description

Check if the sample time property of a block is set to -1 (inherited).

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Sample time property of a block is not set to -1 (inherited).	Change the sample time to -1 (inherited).

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library-linked blocks. By default, the input parameter **Follow links** is set to `on`.
- Analyzes content in masked subsystems. By default, the input parameter **Look under masks** is set to `graphical`.
- Allows exclusions of blocks and charts.

### See Also

- JMAAB guideline, Version 5.1: `jc_0641`: Sample time setting

## Check output data type of operation blocks

**Check ID:** `mathworks.jmaab.jc_0651`

### Description

Checks if the model adhere to the guidelines for implementing type conversion.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
One or more Operation blocks found that explicitly specify output data type.	Instead of explicitly specifying output data type on operation blocks, use Data Type Conversion block when changing the data type of the block output signal.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0651: Guideline for implementing a type conversion

## Check usage of numeric literals in Stateflow

**Check ID:** mathworks.jmaab.jc\_0702

### Description

Identifies the use of numeric literals in Stateflow states and transitions.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
One or more expressions in Stateflow states and Stateflow transitions contain numeric literals.	Consider remodeling to use named parameters and constants instead of numeric literals.

## Capabilities and Limitations

- Runs on library models.

- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0702: Use of named Stateflow parameters/constants

**Check usage of transitions to external states****Check ID:** `mathworks.jmaab.jc_0723`**Description**

Identifies transitions in Stateflow Charts that end on external child states.

Available with Simulink Check.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
One or more transitions end on external child states.	Consider remodeling to avoid use of transitions ending on external child states.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of charts.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0723: Prohibited direct transition from external state to child state

**Check order of state action types****Check ID:** `mathworks.jmaab.jc_0733`

## Description

Identifies state actions that are out of order in Stateflow states.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

## Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0733_a: Check basic state action types	One or more Stateflow states have basic state action types written out of order.	Consider ordering the state actions in the order of entry (en), during (du), and exit (ex).
jc_0733_b: Check combined state action types	One or more Stateflow states have combined state action types written out of order.	Consider ordering the state actions in the order of entry (en), during (du), and exit (ex).

## Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0733: Order of state action types

## Check if state action type 'exit' is used in the model

**Check ID:** mathworks.jmaab.jc\_0740

## Description

Checks if Stateflow exit actions are used in the model.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more Stateflow states use <b>exit</b> action type.	Consider removing state action type <b>exit</b> in Stateflow states.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0740: Limitation on use of exit state action

## Check position of comments in transition labels

**Check ID:** mathworks.jmaab.jc\_0771

### Description

Identifies comments in transition labels that are not positioned uniformly.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.



## Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0771_a1: Comments in transition labels must be uniformly positioned on top	One or more comments in transition labels are not positioned at the top.	Comments in transition labels must be positioned above transition conditions, condition actions, transition actions, and Stateflow events.
jc_0771_a2: Comments in transition labels must be uniformly positioned at the bottom	One or more comments in transition labels are not positioned at the bottom.	Comments in transition labels must be positioned below transition conditions, condition actions, transition actions, and Stateflow events.

## Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0771: Comment position in transition labels

## Check usage of unconditional transitions in flow charts

**Check ID:** mathworks.jmaab.jc\_0773

## Description

Identifies unconditional transitions in flow charts.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0773_a: Check for the presence of unconditional transition originating from a Stateflow junction with conditional transition	One or more Stateflow junctions do not have unconditional transitions originating from them.	Consider adding an unconditional transition to the junction.
jc_0773_b: Check execution order of unconditional transitions	One or more Stateflow junctions have unconditional transitions that are not executed last.	Consider setting the order of execution of the unconditional transition from the junction to the highest value.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0773: Unconditional transition of a flow chart

## Check for comments in unconditional transitions

**Check ID:** `mathworks.jmaab.jc_0774`

### Description

Identifies the comments in unconditional transitions without action statements.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
One or more unconditional transitions without action statements do not have comments.	Consider adding a comment explaining the unconditional transition.

## Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0774: Comments for through transition

## Check terminal junctions in Stateflow

**Check ID:** `mathworks.jmaab.jc_0775`

## Description

Identifies the usage of terminal junctions in flow charts.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

## Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0775_a1: Check for usage of only one terminal junction	One or more Stateflow containers have more than one terminal junction.	Consider using only one terminal junction.

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
jc_0775_a2: Check for usage of one terminal junction with one unconditional transition as input	One or more Stateflow containers have either more than one terminal junction or a terminal junction without one unconditional transition.	Consider using only one terminal junction with one unconditional transition as input.

**Capabilities and Limitations**

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0775: Terminal junction in flow charts

**Check unused data in Simulink Model**

**Check ID:** mathworks.jmaab.jc\_0792

**Description**

Identifies unused data in the model workspace and data dictionary.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

## Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0792_a: Check for unused data in Data Dictionary	One or more data variables in the data dictionary are unused.	Consider removing the unused data variables.
jc_0792_b: Check for unused data in Model Workspace	One or more data variables in the model workspace are unused.	Consider removing the unused data variables.

## Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Does not allow exclusions of blocks or charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0792: Unused data

## Check for use of C-style comment symbols

**Check ID:** mathworks.jmaab.jc\_0801

## Description

Identifies the usage of C-style (/\* and \*/) comments in CGT files and MPT objects.

Available with Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
One or more entities use C-style comments.	Consider removing C-style comments in CGT files and MPT objects.

## Capabilities and Limitations

- Does not run on library models.

- Does not analyze content of library linked blocks.
- Does not analyze content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0801: Prohibited use of the /\* and \*/ comment symbols

## Check for implicit type casting in Stateflow

**Check ID:** mathworks.jmaab.jc\_0802

### Description

Identifies implicit type casting in Stateflow.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
One or more operations and/or function calls in Stateflow charts have data type mismatch.	All operations and function calls must be made between variables of the same data type. If the data types are different, the variables need to be explicitly type casted to match data types.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0802: Prohibited use of implicit type casting in Stateflow

## Check diagnostic settings for incorrect calculation results

**Check ID:** mathworks.jmaab.jc\_0806

### Description

Identifies the status of the configuration parameters of the data validity diagnostic settings which detect incorrect calculation results.

Available with Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model configuration parameter <b>Division by singular matrix</b> (CheckMatrixSingularityMsg) is not set to <b>error</b> .	Set the input parameter value to <b>error</b> .
The model configuration parameter <b>Inf or NaN block output</b> (SignalInfNanChecking) is not set to <b>error</b> .	Set the input parameter value to <b>error</b> .
The model configuration parameter <b>Wrap on overflow</b> (IntegerOverflowMsg) is not set to <b>error</b> .	Set the input parameter value to <b>error</b> .
The model configuration parameter <b>Saturate on overflow</b> (IntegerSaturationMsg) is not set to <b>error</b> .	Set the input parameter value to <b>error</b> .

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Does not allow exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0806: Detecting incorrect calculation results

## Check block names

**Check ID:** mathworks.jmaab.jc\_0231

### Description

Identifies block names with incorrect characters.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0231_a: Characters allowed for block names	One or more blocks in the model have invalid names.	Use alphanumeric characters and underscores in block name.
jc_0231_b: Number at the beginning	One or more blocks in the model have numbers at the beginning of the block name.	Use alphabetic characters at the beginning of the block name.
jc_0231_c: Underscore at the beginning	One or more blocks in the model have underscores at the beginning of the block name.	Use alphabetic characters at the beginning of the block name.
jc_0231_d: Underscore at the end	One or more blocks in the model have underscores at the end of the block name.	Do not end block names with underscores.
jc_0231_e: Consecutive underscores	One or more blocks in the model have consecutive underscores in the block name.	Do not use consecutive underscores in block name.
jc_0231_f: Single reserved MATLAB word	One or more blocks in the model use reserved MATLAB words as the block name.	Do not use reserved MATLAB word as block name.



## Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

## See Also

- JMAAB guideline, Version 5.1: jc\_0231: Usable characters for block names

## Check Model Description

**Check ID:** mathworks.jmaab.jc\_0603

### Description

Identifies the layers in a model that have inconsistent description formatting for the following model elements:

- Annotations
- Model Info Block
- DocBlock Block

You can select the description block type and configure the description tags from the **Description Block Type** and **Description Tags** input parameters. The format for the **Description Tags** should be of a string type with comma separated tags.

By default, the **Description Block Type** is set to **Annotation** and the **Description Tags** parameter is **Input:,Description:,Output:**.

For example, considering the default values (Input:, Description:, Output:), each of the layer in the model should have the description format as following:

Input: <input information>

Description: <model description>

Output: <output information>

Available with Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
One or more layers in the model do not have model descriptions, or do not have consistent model description formatting.	Use a consistent format for the model descriptions at all the layers in the model.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0603: Model description

**Usable characters for file names**

**Check ID:** mathworks.jmaab.ar\_0001

**Description**

Checks weather the file names meet the guideline standards.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

**Results and Recommended Actions**

Guideline Sub ID	Condition	Recommended Action
ar_0001_a: Characters allowed for file names	One or more files have invalid names.	Use alphanumeric characters and underscores in file names.

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
ar_0001_b: Number at the beginning	One or more files have numbers at the beginning of the file name.	Use alphabetic characters at the beginning of the file names.
ar_0001_c: Underscore at the beginning	One or more files have underscores at the beginning of the file name.	Use alphabetic characters at the beginning of the file names.
ar_0001_d: Underscore at the end	One or more files have underscores at the end of the file name.	Do not end the file names with underscores.
ar_0001_e: Consecutive underscores	One or more files have consecutive underscores in the file name.	Do not use consecutive underscores in the file names.
ar_0001_f: Single Reserved MATLAB word	One or more files have reserved MATLAB words as the file name.	Do not use reserved MATLAB word as the file name.
ar_0001_g: Identical file names on path	One or more files have names which are identical to files present in MATLAB path.	Use unique file names.

### Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked files.
- Does not support exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: jc\_0775: Terminating junctions in Flow Charts

## Usable characters for folder names

**Check ID:** mathworks.jmaab.ar\_0002

**Description**

Checks folder names to meet the guideline standards.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

**Results and Recommended Actions**

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
ar_0002_a: Characters allowed for folder names	One or more folders have invalid names.	Use alphanumeric characters and underscores in folder names.
ar_0002_b: Number at the beginning	One or more folders have numbers at the beginning of the folder name.	Use alphabetic characters at the beginning of the folder names.
ar_0002_c: Underscore at the beginning	One or more folders have underscores at the beginning of the folder name.	Use alphabetic characters at the beginning of the folder names.
ar_0002_d: Underscore at the end	One or more folders have underscores at the end of the folder name.	Do not end the folder names with underscores.
ar_0002_e: Consecutive underscores	One or more folders have consecutive underscores in the folder name.	Do not use consecutive underscores in the folder names.
ar_0002_f: Single Reserved MATLAB word	One or more folders have reserved MATLAB words as the folder name.	Do not use reserved MATLAB word as the folder name.

**Capabilities and Limitations**

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in all masked subsystems.
- Does not support exclusions of blocks or charts.

### See Also

- JMAAB guideline, Version 5.1: ar\_0002: folder names

## Usable characters for Subsystem names

**Check ID:** mathworks.jmaab.jc\_0201

### Description

Identifies subsystem names with incorrect characters.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

### Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0201_a: Characters allowed for subsystem names	One or more subsystems have invalid names.	Use alphanumeric characters and underscores in subsystem names.
jc_0201_b: Number at the beginning	One or more subsystems have numbers at the beginning of the subsystem name.	Use alphabetic characters at the beginning of the subsystem names.
jc_0201_c: Underscore at the beginning	One or more subsystems have underscores at the beginning of the subsystem name.	Use alphabetic characters at the beginning of the subsystem names.
jc_0201_d: Underscore at the end	One or more subsystems have underscores at the end of the subsystem name.	Do not end the subsystem names with underscores.

<b>Guideline Sub ID</b>	<b>Condition</b>	<b>Recommended Action</b>
jc_0201_e: Consecutive underscores	One or more subsystems have consecutive underscores in the subsystem name.	Do not use consecutive underscores in the subsystem names.
jc_0201_f: Single Reserved MATLAB word	One or more subsystems have reserved MATLAB words as the subsystem name.	Do not use reserved MATLAB word as the subsystem name.

**Capabilities and Limitations**

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

**Edit-Time Checking.** This check is supported by edit-time checking.

**See Also**

- JMAAB guideline, Version 5.1: jc\_0201: Subsystem names

**Usable characters for Inport block and Outport block**

**Check ID:** mathworks.jmaab.jc\_0211

**Description**

Identifies Inport or Outport block names with incorrect characters.

This check contains sub-checks. You can choose to run single or multiple sub-checks by selecting sub-ids from the Model Advisor configuration editor.

Available with Simulink Check.

## Results and Recommended Actions

Guideline Sub ID	Condition	Recommended Action
jc_0211_a: Characters allowed for port block names	One or more port blocks have invalid names.	Use alphanumeric characters and underscores in port block names.
jc_0211_b: Number at the beginning	One or more port blocks have numbers at the beginning of the port block name.	Use alphabetic characters at the beginning of the port block names.
jc_0211_c: Underscore at the beginning	One or more port blocks have underscores at the beginning of the port block name.	Use alphabetic characters at the beginning of the port block names.
jc_0211_d: Underscore at the end	One or more port blocks have underscores at the end of the port block name.	Do not end the port block names with underscores.
jc_0211_e: Consecutive underscores	One or more port blocks have consecutive underscores in the port block name.	Do not use consecutive underscores in the port block names.
jc_0211_f: Single Reserved MATLAB word	One or more port blocks have reserved MATLAB words as the port block name.	Do not use reserved MATLAB word as the port block name.

## Capabilities and Limitations

- Supports selection of Guideline Sub IDs.
- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Supports exclusions of blocks or charts.

**Edit-Time Checking.** This check is supported by edit-time checking.

### **See Also**

- JMAAB guideline, Version 5.1: jc\_0211: port block names



## MISRA C:2012 Checks

### In this section...

“See Also” on page 2-377

“Check usage of Assignment blocks” on page 2-377

“Check for blocks not recommended for MISRA C:2012” on page 2-379

“Check for unsupported block names” on page 2-381

“Check configuration parameters for MISRA C:2012” on page 2-382

“Check for equality and inequality operations on floating-point values” on page 2-386

“Check for bitwise operations on signed integers” on page 2-387

“Check for recursive function calls” on page 2-388

“Check for switch case expressions without a default case” on page 2-389

“Check for blocks not recommended for C/C++ production code deployment” on page 2-390

“Check for missing error ports for AUTOSAR receiver interfaces” on page 2-391

“Check for missing const qualifiers in model functions” on page 2-393

“Check integer word length” on page 2-394

“Check bus object names that are used as bus element names” on page 2-395

You can check that your model or subsystem has a likelihood of generating MISRA C:2012 compliant code.

### See Also

- “Run Model Advisor Checks and Review Results”

### Check usage of Assignment blocks

**Check ID:** `mathworks.misra.AssignmentBlocks`

Identify Assignment blocks that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.

### Description

This check applies to the Assignment block that is available in the Simulink block library under **Simulink > Math Operations**.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter <b>Action if any output element is not assigned</b> set to <b>Error</b> or <b>Warning</b> .	Set block parameter <b>Action if any output element is not assigned</b> to one of the recommended values: <ul style="list-style-type: none"> <li>• <b>Error</b>, if Assignment block is not in an Iterator subsystem.</li> <li>• <b>Warning</b>, if Assignment block is in an Iterator subsystem.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking. However, the following check condition is not supported because edit-time checking is unable to determine whether the Assignment block is an Iterator subsystem.

Set block parameter **Action if any output element is not assigned** to one of the recommended values:

- **Error**, if Assignment block is not in an Iterator subsystem.

- **Warning**, if Assignment block is in an Iterator subsystem.

**See Also**

- MISRA C:2012, Rule 9.1
- ISO/IEC TS 17961: 2013, uninitref
- CERT C, EXP33-C
- CWE, CWE-908
- “hisl\_0029: Usage of Assignment blocks” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

**Check for blocks not recommended for MISRA C:2012**

**Check ID:** `mathworks.misra.BlkSupport`

Identify blocks that are not supported or recommended for MISRA C:2012 compliant code generation.

**Description**

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• 1-D Lookup Table</li> <li>• 2-D Lookup Table</li> <li>• n-D Lookup Table</li> </ul>	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• Lookup Table</li> <li>• Lookup Table (2-D)</li> </ul>	Consider replacing the deprecated Lookup Table blocks.
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.
String blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• Compose String</li> <li>• Scan String</li> <li>• String to Single</li> <li>• String to Double</li> <li>• To String</li> </ul>	Consider replacing the String blocks with blocks recommended for production.

**Capabilities and Limitations**

You can:

- Run this check on your library models.

- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **See Also**

- hisl\_0020: Blocks not recommended for MISRA C:2012 compliance
- na\_0027: Use of only standard library blocks
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Model Advisor Exclusion Overview”

## **Check for unsupported block names**

**Check ID:** `mathworks.misra.BlockNames`

Identify block names containing /.

### **Description**

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

### **Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
Block names containing / were found in the model or subsystem.	Remove / from the block name.

### **Capabilities and Limitations**

- Runs on library models.
- Analyzes content of library linked blocks.

- Analyzes content in masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

**Edit-Time Checking**

This check is supported by edit-time checking.

**See Also**

- MISRA C:2012, Rule 3.1
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)

**Check configuration parameters for MISRA C:2012**

**Check ID:** `mathworks.misra.CodeGenSettings`

Identify configuration parameters that can impact MISRA C:2012 compliant code generation.

**Description**

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications.

Available with Embedded Coder and Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
<b>Math and Data Types</b>	
Configuration parameter <b>Use division for fixed-point net slope computation</b> is not set to 0n or Use division for reciprocals of integers only.	Set <b>Use division for fixed-point net slope computation</b> to 0n or Use division for reciprocals of integers only.

Condition	Recommended Action
Configuration parameter <b>Inf or NaN block output</b> is set to None or error and <b>Support non-finite numbers</b> is set to on.	When <b>Support non-finite numbers</b> is:
Configuration parameter <b>Inf or NaN block output</b> is set to None and <b>Support non-finite numbers</b> is set to off.	<ul style="list-style-type: none"> <li>• on, set <b>Inf or NaN block output</b> to warning</li> <li>• off, set <b>Inf or NaN block output</b> to warning or error</li> </ul>
Configuration parameter <b>Model Verification block enabling</b> is set to Use local settings or Enable All.	Set <b>Model Verification block enabling</b> to Disable All.
Configuration parameter <b>Undirected event broadcasts</b> is set to none or warning.	Set <b>Undirected event broadcasts</b> to error.
Configuration parameter <b>Wrap on overflow</b> is set to None	Set configuration parameter <b>Wrap on overflow</b> to warning or error.
<b>Hardware Implementation</b>	
Configuration parameter <b>Production hardware signed integer division rounds to</b> is set to Undefined	Set <b>Production hardware signed integer division rounds to</b> to Zero or Floor.
Configuration parameter <b>Shift right on a signed integer as arithmetic shift</b> is selected.	Clear <b>Shift right on a signed integer as arithmetic shift</b> .
<b>Simulation Target</b>	
Configuration parameter <b>Compile-time recursion limit for MATLAB functions</b> is set to a value other than 0 .	Set <b>Compile-time recursion limit for MATLAB functions</b> to 0 .
Configuration parameter <b>Dynamic memory allocation in MATLAB functions</b> is selected.	Clear <b>Dynamic memory allocation in MATLAB functions</b> .
Configuration parameter <b>Enable run-time recursion for MATLAB functions</b> is selected.	Clear <b>Enable run-time recursion for MATLAB functions</b> .
<b>Code Generation</b>	

Condition	Recommended Action
<p>Configuration parameter <b>Bitfield declarator type specifier</b> is set to uchar_T when any of these parameters are selected:</p> <ul style="list-style-type: none"> <li>• <b>Pack Boolean data into bitfields</b></li> <li>• <b>Use bitsets for storing state configuration</b></li> <li>• <b>Use bitsets for storing Boolean data</b></li> </ul>	<p>Set <b>Bitfield declarator type specifier</b> to uint_T.</p>
<p>Configuration parameter <b>Casting Modes</b> is not set to Standards Compliant.</p>	<p>Set <b>Casting Modes</b> to Standards Compliant.</p>
<p>Configuration parameter <b>Code replacement library</b> is not set to None or AUTOSAR 4.0.</p>	<p>Set <b>Code replacement library</b> to None or AUTOSAR 4.0</p>
<p>Configuration parameter <b>External mode</b> is selected.</p>	<p>Clear <b>External mode</b>.</p>
<p>Configuration parameter <b>Generate shared constants</b> is selected.</p>	<p>Clear <b>Generate shared constants</b>.</p>
<p>Configuration parameter <b>Include comments</b> is cleared.</p>	<p>Select <b>Include comments</b>.</p>
<p>Configuration parameter <b>MAT-file logging</b> is selected.</p>	<p>Clear <b>MAT-file logging</b></p>
<p>For ERT-based target systems, configuration parameter <b>MATLAB user comments</b> is cleared.</p>	<p>Select <b>MATLAB user comments</b>.</p>
<p>A value for configuration parameter <b>Maximum identifier length</b> is not provided.</p>	<p>Set the value to the implementation-dependent limit. The default is 31.</p>
<p>Configuration parameter <b>Parenthesis level</b> is not set to Maximum (Specify precedence with parentheses).</p>	<p>Set <b>Parentheses level</b> to Maximum (Specify precedence with parentheses).</p>



Condition	Recommended Action
For ERT-based target systems, configuration parameter <b>Preserve static keyword in function declarations</b> is cleared when <b>File packaging format</b> is set to Compact or Compact (with separate data file)	Select <b>Preserve static keyword in function declarations</b> .
Configuration parameter <b>Replace multiplications by powers of two with signed bitwise shifts</b> is selected.	Clear <b>Replace multiplications by powers of two with signed bitwise shifts</b> .
Configuration parameter <b>Shared code placement</b> is set to Auto.	Set <b>Shared code placement</b> to Shared location
For ERT-based target systems, configuration parameter <b>Support continuous time</b> is selected	Clear <b>Support continuous time</b> .
For ERT-based target systems, configuration parameter <b>Support non-inlined S-functions</b> is selected	Clear <b>Support non-inlined S-functions</b> .
Configuration parameter <b>System-generated identifiers</b> is set to Classic.	Set <b>System-generated identifiers</b> to Shortened.
Configuration parameter <b>System target file</b> is set to a GRT-based target.	Set <b>System target file</b> to an ERT-based target.
Configuration parameter <b>Use dynamic memory allocation for model initialization</b> is selected when <b>Code Interface Packaging</b> is set to Reusable Function.	Clear <b>Use dynamic memory allocation for model initialization</b> .  <b>Note</b> Select only when <b>Code Interface Packaging</b> is set to Reusable Function.

### Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

---

**Note** When you click **Modify All** for models with a GRT-based target, the Model Advisor does not update the **System target file** configuration parameter to an ERT-based system.

---

Parameter subchecks depend on the results of the parameter noted with **D** in the results table. When the result is *D-Warning*, the **Current Value** column in the results table states *Prerequisite constraint not met* for the subchecks. After you change the parameter, rerun the check.

---

**Note** Some subchecks are specific to configuration parameters for ERT-based systems. These parameters are not updated when you click **Modify All** unless you change the model to an ERT-based system.

---

### Capabilities and Limitations

This check does not review referenced models.

#### See Also

- hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)

## Check for equality and inequality operations on floating-point values

**Check ID:** `mathworks.misra.CompareFloatEquality`

Identify equality and inequality operations on floating-point values.

### Description

The check flags sources causing equality or inequality operations on floating-point values.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

The check does not flag blocks with equality or inequality operations on floating-point values if they are justified with a Polyspace® annotation. When you run the check, the **Blocks with justification** table lists blocks with equality or inequality operations that have a justification.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Model object has an equality or inequality operation on a floating-point value.	Consider using non-floating-point values for equality or inequality operations.

### Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Dir 1.1
- CERT C, FLP00-C
- CWE, CWE-697
- “Annotate Code and Hide Known or Acceptable Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

## Check for bitwise operations on signed integers

**Check ID:** `mathworks.misra.CompliantCGIRConstructions`

Identify Simulink blocks that contain bitwise operations on signed integers.

### Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

This check requires a Stateflow license when Stateflow is used in the model.

### Results and Recommended Actions

Condition	Recommended Action
The model has blocks that contain bitwise operations on signed integers.	Consider using unsigned integers for bitwise operations.

### Capabilities and Limitations

You can:

- The check assumes that code is generated for the whole model. When code is generated by a subsystem build or export functions, the check can product incorrect results.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance” (Simulink)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

## Check for recursive function calls

**Check ID:** `mathworks.misra.RecursionCompliance`

Identify recursive function calls in Stateflow charts.

### Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags charts that have recursive function calls.

Available with Embedded Coder and Simulink Check.

This check requires a Stateflow license.

## Results and Recommended Actions

Condition	Recommended Action
Chart has a recursive function call.	Remove recursive function call.

### See Also

- MISRA C:2012, Dir 17.2
- “Avoid Unwanted Recursion in a Chart” (Stateflow)

## Check for switch case expressions without a default case

**Check ID:** `mathworks.misra.SwitchDefault`

Identify switch case expressions that do not have a default case.

### Description

The check flags model objects that have switch case expressions without a default case.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

The check does not flag blocks without default cases if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks without default cases that have a justification.

Available with Embedded Coder and Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Model object has a switch case expression without a default case.	For Switch Case blocks, consider selecting block parameter <b>Show default case</b> to explicitly specify a default case.

### Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **See Also**

- MISRA C:2012, Rule 16.4
- ISO/IEC TS 17961: 2013, swtchdflt
- CERT C, MSC01-C
- CWE, CWE-478
- “Annotate Code and Hide Known or Acceptable Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

## **Check for blocks not recommended for C/C++ production code deployment**

**Check ID:** `mathworks.codegen.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

### **Description**

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder.

In some instances, this check flags blocks that are supported for code generation. For these blocks, you should review the footnote information that is provided in the support notes and adhere to the recommended action provided by the Model Advisor.

Following the recommendations of this check increases the likelihood of generating code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.
The Model Advisor flags blocks that are supported for code generation	Review the support notes and adhere to the recommended action provided by the Model Advisor.

## Capabilities and Limitations

You can:

- Run this check on your library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Exclude blocks and charts from this check if you have a Simulink Check license.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- na\_0027: Use of only standard library blocks
- “Blocks and Products Supported for Code Generation” (Simulink Coder)
- “Model Advisor Exclusion Overview”
- Secure Coding Standards (Embedded Coder)

## Check for missing error ports for AUTOSAR receiver interfaces

**Check ID:** `mathworks.misra.AutosarReceiverInterface`

Identify AUTOSAR receiver interface inports that do not have matching error ports.

## Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags AUTOSAR

receiver interfaces inports that are missing error ports. The following table identifies the AUTOSAR data access mode types for receiver interface ports that are flagged by the check when the corresponding error port is missing.

<b>AUTOSAR Data Access Mode Type</b>	<b>Flagged by Check?</b>
ImplicitReceive	Yes
ExplicitReceive	Yes
QueuedExplicitReceive	No
ErrorStatus	No
ModeReceive	No
IsUpdated	No
EndToEndRead	Yes
ExplicitReceiveByVal	No
otherwise	No

The check does not flag missing error ports when they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists the missing error ports that have a justification.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

<b>Condition</b>	<b>Recommended Action</b>
AUTOSAR receiver interface inport does not have a matching error port.	Add missing error port and map to the corresponding AUTOSAR receiver interface inport.
AUTOSAR receiver interface ports do not have a matching error port when data access mode is ImplicitReceive, ExplicitReceive, or EndToEndRead.	Add missing error port and map to the corresponding AUTOSAR receiver interface inport.

### Capabilities and Limitations

You can:



- Analyzes top layer/root level models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C: 2012, Directive 4.7
- “MISRA C Guidelines” (Embedded Coder)
- “Model Advisor Exclusion Overview”
- “Annotate Code and Hide Known or Acceptable Results” (Polyspace Bug Finder)
- “Configure AUTOSAR Elements and Properties” (AUTOSAR Blockset)
- “AUTOSAR Component Configuration” (AUTOSAR Blockset)

## Check for missing const qualifiers in model functions

**Check ID:** `mathworks.misra.ModelFunctionInterface`

Identify missing const qualifiers in input data pointers.

### Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications. The check flags input data pointers that do not have a const qualifier.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
A const qualifier is not defined for the input data pointer.	Consider adding a const qualifier to the input data pointer.

### See Also

- MISRA C:2012, Rule 8.13
- “MISRA C Guidelines” (Embedded Coder)

## Check integer word length

**Check ID:** `mathworks.misra.IntegerWordLengths`

Identify integer word lengths that do not comply with hardware implementation settings

### Description

The check flags integers whose word lengths exceed the number of bits permitted via the hardware implementation settings.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Model object contains integer word lengths that are not compliant with hardware implementation settings.	Update the integer so its length does not exceed the permitted number of bits. You can view the permitted number of bits in the Configuration Parameters dialog box, on the <b>Hardware Implementation &gt; Device details</b> pane.

### Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “MISRA C Guidelines” (Embedded Coder)
- “Model Advisor Exclusion Overview”

- “Secure Coding Standards” (Embedded Coder)

## Check bus object names that are used as bus element names

**Check ID:** `mathworks.misra.BusElementNames`

Identify bus object names that are used as bus element names.

### Description

Using this check increases the likelihood of generating code for embedded applications that is compliant with MISRA C:2012. The check flags instances where a Simulink.Bus object name is used as the Simulink.Bus element name.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
A bus object name is being used as a bus element name.	Change either the flagged bus object name or the bus element name so that they are not identical.

### See Also

- MISRA C:2012, Rule 5.6
- MISRA AC AGC, Rule 5.3
- “MISRA C Guidelines” (Embedded Coder)

## Secure Coding Checks for CERT C, CWE, and ISO/IEC TS 17961 Standards

In this section...
“See Also” on page 2-396
“Check configuration parameters for secure coding standards” on page 2-396
“Check for blocks not recommended for C/C++ production code deployment” on page 2-399
“Check for blocks not recommended for secure coding standards” on page 2-401
“Check usage of Assignment blocks” on page 2-402
“Check for switch case expressions without a default case” on page 2-404
“Check for bitwise operations on signed integers” on page 2-405
“Check for equality and inequality operations on floating-point values” on page 2-406
“Check integer word length” on page 2-407
“Detect Dead Logic” on page 2-408
“Detect Integer Overflow” on page 2-411
“Detect Division by Zero” on page 2-413
“Detect Out Of Bound Array Access” on page 2-414
“Detect Specified Minimum and Maximum Value Violations” on page 2-416

These checks are used to validate that code generated by Embedded Coder complies with the CERT C, CWE, and ISO/IEC TS 17961 (Embedded Coder) secure coding standards.

### See Also

- “Run Model Advisor Checks and Review Results”

### Check configuration parameters for secure coding standards

**Check ID:** `mathworks.security.CodeGenSettings`

Identify configuration parameters that might impact compliance with secure coding standards.

## Description

Following the recommendations of this check increases the likelihood of generating code that complies with CERT C, CWE, ISO/IEC TS 17961 secure coding standards.

Available with Embedded Coder and Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
<b>Diagnostics</b>	
Configuration parameter <b>Inf or NaN block output</b> is set to None or error and <b>Support non-finite numbers</b> is set to on.	When <b>Support non-finite numbers</b> is: <ul style="list-style-type: none"> <li>• on, set <b>Inf or NaN block output</b> to warning.</li> <li>• off, set <b>Inf or NaN block output</b> to warning or error.</li> </ul>
Configuration parameter <b>Inf or NaN block output</b> is set to None and <b>Support non-finite numbers</b> is set to off.	
Configuration parameter <b>Model Verification block enabling</b> is set to Use local settings or Enable All.	Set <b>Model Verification block enabling</b> to Disable All.
Configuration parameter <b>Undirected event broadcasts</b> is set to none or warning.	Set <b>Undirected event broadcasts</b> to error.
Configuration parameter <b>Wrap on overflow</b> is set to none.	Set <b>Wrap on overflow</b> to warning or error.
<b>Hardware Implementation</b>	
Configuration parameter <b>Production hardware signed integer division rounds to</b> is set to Undefined.	Set <b>Production hardware signed integer division rounds to</b> to Zero or Floor.
Configuration parameter <b>Shift right on a signed integer as arithmetic shift</b> is selected.	Clear <b>Shift right on a signed integer as arithmetic shift</b> .
<b>Simulation Target</b>	

Condition	Recommended Action
Configuration parameter <b>Compile-time recursion limit for MATLAB functions</b> is set to a value other than 0 .	Set <b>Compile-time recursion limit for MATLAB functions</b> to 0 .
Configuration parameter <b>Dynamic memory allocation in MATLAB functions</b> is selected.	Clear <b>Dynamic memory allocation in MATLAB functions</b> .
Configuration parameter <b>Enable run-time recursion for MATLAB functions</b> is selected.	Clear <b>Enable run-time recursion for MATLAB functions</b> .
<b>Code Generation</b>	
Configuration parameter <b>Code replacement library</b> is not set to None or AUTOSAR 4.0.	Set <b>Code replacement library</b> to None or AUTOSAR 4.0.
Configuration parameter <b>External mode</b> is selected.	Clear <b>External mode</b> .
Configuration parameter <b>Include comments</b> is cleared.	Select <b>Include comments</b> .
Configuration parameter <b>MAT-file logging</b> is selected.	Clear <b>MAT-file logging</b> .
For ERT-based target systems, configuration parameter <b>MATLAB user comments</b> is cleared.	Select <b>MATLAB user comments</b> .
Configuration parameter <b>Replace multiplications by powers of two with signed bitwise shifts</b> is selected.	Clear <b>Replace multiplications by powers of two with signed bitwise shifts</b> .
For ERT-based target systems, configuration parameter <b>Support continuous time</b> is selected	Clear <b>Support continuous time</b> .
For ERT-based target systems, configuration parameter <b>Support non-inlined S-functions</b> is selected	Clear <b>Support non-inlined S-functions</b> .
Configuration parameter <b>System target file</b> is set to a GRT-based target.	Set <b>System target file</b> to an ERT-based target.

Condition	Recommended Action
Configuration parameter <b>Use dynamic memory allocation for model initialization</b> is selected.	Clear <b>Use dynamic memory allocation for model initialization</b> .  <b>Note</b> Select only when <b>Code Interface Packaging</b> is set to Reusable Function.

### Action Results

Clicking **Modify All** changes the parameter values to the recommended values.

---

**Note** When you click **Modify All** for models with a GRT-based target, the Model Advisor does not update the **System target file** configuration parameter to an ERT-based system.

---

Parameter subchecks depend on the results of the parameter noted with **D** in the results table. When the result is *D-Warning*, the **Current Value** column in the results table states *Prerequisite constraint not met* for the subchecks. After you change the parameter, rerun the check.

---

**Note** Some subchecks are specific to configuration parameters for ERT-based systems. These parameters are not updated when you click **Modify All** unless you change the model to an ERT-based system.

---

### See Also

“Secure Coding Standards” (Embedded Coder)

## Check for blocks not recommended for C/C++ production code deployment

**Check ID:** `mathworks.codegen.PCGSupport`

Identify blocks not supported by code generation or not recommended for C/C++ production code deployment.

**Description**

This check partially identifies model constructs that are not recommended for C/C++ production code generation as identified in the Simulink Block Support (Simulink Coder) tables for Simulink Coder and Embedded Coder.

In some instances, this check flags blocks that are supported for code generation. For these blocks, you should review the footnote information that is provided in the support notes and adhere to the recommended action provided by the Model Advisor.

Following the recommendations of this check increases the likelihood of generating code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

**Results and Recommended Actions**

<b>Condition</b>	<b>Recommended Action</b>
The model or subsystem contains blocks that should not be used for production code deployment.	Consider replacing the blocks listed in the results. Click an element from the list of questionable items to locate condition.
The Model Advisor flags blocks that are supported for code generation	Review the support notes and adhere to the recommended action provided by the Model Advisor.

**Capabilities and Limitations**

You can:

- Run this check on your library models.
- Analyzes content of library linked blocks.
- Analyzes content in all masked subsystems.
- Exclude blocks and charts from this check if you have a Simulink Check license.

**Edit-Time Checking**

This check is supported by edit-time checking.



**See Also**

- na\_0027: Use of only standard library blocks
- “Blocks and Products Supported for Code Generation” (Simulink Coder)
- “Model Advisor Exclusion Overview”
- Secure Coding Standards (Embedded Coder)

**Check for blocks not recommended for secure coding standards**

**Check ID:** `mathworks.security.BlockSupport`

Identify blocks not recommended for compliance with secure coding standards.

**Description**

Following the recommendations of this check increases the likelihood of generating code that complies with CERT C, CWE, ISO/IEC TS 17961 secure coding standards.

Available with Embedded Coder and Simulink Check.

**Results and Recommended Actions**

Condition	Recommended Action
Lookup Table blocks using cubic spline interpolation or extrapolation methods were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• 1-D Lookup Table</li> <li>• 2-D Lookup Table</li> <li>• n-D Lookup Table</li> </ul>	Consider other interpolation and extrapolation methods for the Lookup Table blocks.
Deprecated Lookup Table blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"> <li>• Lookup Table</li> <li>• Lookup Table (2-D)</li> </ul>	Consider replacing the deprecated Lookup Table blocks.

<b>Condition</b>	<b>Recommended Action</b>
S-Function Builder blocks were found in the model or subsystem.	Consider replacing the S-Function Builder blocks with blocks recommended for production.
From Workspace blocks were found in the model or subsystem	Consider replacing the From Workspace blocks with blocks recommended for production.
String blocks were found in the model or subsystem. Specific blocks are: <ul style="list-style-type: none"><li>• Compose String</li><li>• Scan String</li><li>• String to Single</li><li>• String to Double</li><li>• To String</li></ul>	Consider replacing the String blocks with blocks recommended for production.

### **Capabilities and Limitations**

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### **Edit-Time Checking**

This check is supported by edit-time checking.

### **See Also**

- `na_0027`: Use of only standard library blocks
- “Model Advisor Exclusion Overview”
- “Secure Coding Standards” (Embedded Coder)

## **Check usage of Assignment blocks**

**Check ID:** `mathworks.misra.AssignmentBlocks`

Identify Assignment blocks that do not have block parameter **Action if any output element is not assigned** set to **Error** or **Warning**.

### Description

This check applies to the Assignment block that is available in the Simulink block library under **Simulink > Math Operations**.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
The model or subsystem might contain Assignment blocks with incomplete array initialization that do not have block parameter <b>Action if any output element is not assigned</b> set to <b>Error</b> or <b>Warning</b> .	Set block parameter <b>Action if any output element is not assigned</b> to one of the recommended values: <ul style="list-style-type: none"> <li>• <b>Error</b>, if Assignment block is not in an Iterator subsystem.</li> <li>• <b>Warning</b>, if Assignment block is in an Iterator subsystem.</li> </ul>

### Capabilities and Limitations

- Runs on library models.
- Analyzes content of library linked blocks.
- Analyzes content in masked subsystems.
- If you have a Simulink Check license, allows exclusions of blocks and charts.

### Edit-Time Checking

This check is supported by edit-time checking. However, the following check condition is not supported because edit-time checking is unable to determine whether the Assignment block is an Iterator subsystem.

Set block parameter **Action if any output element is not assigned** to one of the recommended values:

- **Error**, if Assignment block is not in an Iterator subsystem.
- **Warning**, if Assignment block is in an Iterator subsystem.

### See Also

- MISRA C:2012, Rule 9.1
- ISO/IEC TS 17961: 2013, uninitref
- CERT C, EXP33-C
- CWE, CWE-908
- “hisl\_0029: Usage of Assignment blocks” (Simulink)
- “MISRA C Guidelines” (Embedded Coder)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

## Check for switch case expressions without a default case

**Check ID:** `mathworks.misra.SwitchDefault`

Identify switch case expressions that do not have a default case.

### Description

The check flags model objects that have switch case expressions without a default case.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

The check does not flag blocks without default cases if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks without default cases that have a justification.

Available with Embedded Coder and Simulink Check.

## Results and Recommended Actions

Condition	Recommended Action
Model object has a switch case expression without a default case.	For Switch Case blocks, consider selecting block parameter <b>Show default case</b> to explicitly specify a default case.

## Capabilities and Limitations

You can:

- Run this check on your library models.
- Exclude blocks and charts from this check if you have a Simulink Check license.

## Edit-Time Checking

This check is supported by edit-time checking.

## See Also

- MISRA C:2012, Rule 16.4
- ISO/IEC TS 17961: 2013, swtchdflt
- CERT C, MSC01-C
- CWE, CWE-478
- “Annotate Code and Hide Known or Acceptable Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

## Check for bitwise operations on signed integers

**Check ID:** `mathworks.misra.CompliantCGIRConstructions`

Identify Simulink blocks that contain bitwise operations on signed integers.

## Description

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

This check requires a Stateflow license when Stateflow is used in the model.

### Results and Recommended Actions

Condition	Recommended Action
The model has blocks that contain bitwise operations on signed integers.	Consider using unsigned integers for bitwise operations.

### Capabilities and Limitations

You can:

- The check assumes that code is generated for the whole model. When code is generated by a subsystem build or export functions, the check can product incorrect results.
- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance” (Simulink)
- “MISRA C:2012 Compliance Considerations” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

## Check for equality and inequality operations on floating-point values

**Check ID:** `mathworks.misra.CompareFloatEquality`

Identify equality and inequality operations on floating-point values.

### Description

The check flags sources causing equality or inequality operations on floating-point values.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

The check does not flag blocks with equality or inequality operations on floating-point values if they are justified with a Polyspace annotation. When you run the check, the **Blocks with justification** table lists blocks with equality or inequality operations that have a justification.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Model object has an equality or inequality operation on a floating-point value.	Consider using non-floating-point values for equality or inequality operations.

### Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Dir 1.1
- CERT C, FLP00-C
- CWE, CWE-697
- “Annotate Code and Hide Known or Acceptable Results” (Polyspace Bug Finder)
- “Secure Coding Standards” (Embedded Coder)

## Check integer word length

**Check ID:** `mathworks.misra.IntegerWordLengths`

Identify integer word lengths that do not comply with hardware implementation settings

### Description

The check flags integers whose word lengths exceed the number of bits permitted via the hardware implementation settings.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.

Available with Embedded Coder and Simulink Check.

### Results and Recommended Actions

Condition	Recommended Action
Model object contains integer word lengths that are not compliant with hardware implementation settings.	Update the integer so its length does not exceed the permitted number of bits. You can view the permitted number of bits in the Configuration Parameters dialog box, on the <b>Hardware Implementation &gt; Device details</b> pane.

### Capabilities and Limitations

You can:

- Exclude blocks and charts from this check if you have a Simulink Check license.

### See Also

- MISRA C:2012, Rule 10.1
- CERT C, INT13-C
- CWE, CWE-682
- “MISRA C Guidelines” (Embedded Coder)
- “Model Advisor Exclusion Overview”
- “Secure Coding Standards” (Embedded Coder)

### Detect Dead Logic

**Check ID:** `mathworks.sldv.deadlogic`



Identify logic that stays inactive during simulation.

### Description

This check identifies portions of your model that stay inactive during simulation.

You can run a more detailed analysis that identifies both dead logic and active logic using Simulink Design Verifier™ design error detection. For more information, see “Detect Dead Logic Caused by an Incorrect Value” (Simulink Design Verifier).

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards

### Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See: <ul style="list-style-type: none"> <li>• “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)</li> <li>• “Support Limitations for Model Blocks” (Simulink Design Verifier)</li> <li>• “Support Limitations for Simulink Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for Stateflow Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)</li> </ul> Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).

Result	Recommended Action
Dead logic found in model	<p>Simulink Design Verifier proved that these decision and condition outcomes cannot occur and are dead logic in the model. Dead logic can also be a side effect of specified constraints on parameters or specified minimum and maximum constraints on input ports. In rare cases, dead logic can result from approximations performed by Simulink Design Verifier. It is possible that there are objectives that this analysis did not decide. To extend the results of this analysis, use Simulink Design Verifier design error detection to also identify active logic. From the Simulink Editor, select <b>Analysis &gt; Design Verifier &gt; Options</b>. In the <b>Design Error Detection</b> pane, select both <b>Dead logic</b> and <b>Identify active logic</b>.</p>
Dead logic not found in model	<p>Simulink Design Verifier did not find dead logic in the model. It is possible that there are objectives that this analysis did not decide. To extend the results of this analysis, use Simulink Design Verifier design error detection to also identify active logic. From the Simulink Editor, select <b>Analysis &gt; Design Verifier &gt; Options</b>. In the <b>Design Error Detection</b> pane, select both <b>Dead logic</b> and <b>Identify active logic</b>.</p>

**See Also**

- MISRA C:2012: Rule 2.1
- CERT C, MSC07-C
- CWE, CWE-561
- “Run Model Advisor Checks” (Simulink)
- “Secure Coding Standards” (Embedded Coder)

- “Detect Dead Logic Caused by an Incorrect Value” (Simulink Design Verifier)
- “Design Verifier Pane: Design Error Detection” (Simulink Design Verifier)

## **Detect Integer Overflow**

**Check ID:** `mathworks.sldv.integeroverflow`

Detects integer or fixed-point data overflow errors in your model

### **Description**

This check identifies operations that exceed the data type range for integer or fixed-point operations.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

**Results and Recommended Actions**

Result	Recommended Action
Failed, model incompatible	<p>Resolve the model incompatibility. See</p> <ul style="list-style-type: none"> <li>• “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)</li> <li>• “Support Limitations for Model Blocks” (Simulink Design Verifier)</li> <li>• “Support Limitations for Simulink Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for Stateflow Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)</li> </ul> <p>Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).</p>
Integer overflow found in model	<p>To view the conditions that cause the integer overflow, create a harness model. When you simulate the harness, the inputs replicate the error. Click <b>View test case</b> in the Model Advisor report.</p>

**See Also**

- MISRA C:2012: Directive 4.1
- ISO/IEC TS 17961: 2013, intoflow
- CERT C, INT30-C and INT32-C
- CWE, CWE-190
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)

- “Detect Integer Overflow and Division-by-Zero Errors” (Simulink Design Verifier)

## Detect Division by Zero

**Check ID:** `mathworks.sldv.divbyzero`

Detects division-by-zero errors in your model

### Description

This check identifies operations in your model that cause division-by-zero errors.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

### Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	<p>Resolve the model incompatibility. See</p> <ul style="list-style-type: none"> <li>• “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)</li> <li>• “Support Limitations for Model Blocks” (Simulink Design Verifier)</li> <li>• “Support Limitations for Simulink Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for Stateflow Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)</li> </ul> <p>Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).</p>

Result	Recommended Action
Division by zero found in model	To view the conditions that cause the division by zero, create a harness model. When you simulate the harness, the inputs replicate the error. Click <b>View test case</b> in the Model Advisor report.

### See Also

- MISRA C:2012: Directive 4.1
- ISO/IEC TS 17961: 2013, diverr
- CERT C, INT33-C and FLP03-C
- CWE, CWE-369
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)
- “Detect Integer Overflow and Division-by-Zero Errors” (Simulink Design Verifier)

## Detect Out Of Bound Array Access

**Check ID:** `mathworks.sldv.arraybounds`

Detects operations that access outside the bounds of an array index

### Description

This check detects instances of out of bound array access in Simulink Design Verifier.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C, CWE, ISO/IEC TS 17961 standards.

## Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	Resolve the model incompatibility. See <ul style="list-style-type: none"> <li>• “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)</li> <li>• “Support Limitations for Model Blocks” (Simulink Design Verifier)</li> <li>• “Support Limitations for Simulink Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for Stateflow Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)</li> </ul> Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).
Out of bound array access found in model	To view the conditions that cause the out of bound array access, create a harness model. When you simulate the harness, the inputs replicate the error. Click <b>View test case</b> in the Model Advisor report.

### See Also

- MISRA C:2012: Rule 18.1
- ISO/IEC TS 17961: 2013, invptr
- CERT C, ARR30-C
- CWE, CWE-118
- “Secure Coding Standards” (Embedded Coder)
- “Design Error Detection” (Simulink Design Verifier)

- “Detect Out of Bound Array Access Errors” (Simulink Design Verifier)

### **Detect Specified Minimum and Maximum Value Violations**

**Check ID:** `mathworks.sldv.minmax`

Detect signals which exceed specified minimum and maximum values

#### **Description**

This analysis checks the specified minimum and maximum values (the design ranges) on intermediate signals throughout the model and on the output ports. If the analysis detects that a signal exceeds the design range, the results identify where in the model the errors occurred.

Following the recommendations of this check increases the likelihood of generating MISRA C:2012 compliant code for embedded applications, as well as code that complies with the CERT C and CWE standards.



## Results and Recommended Actions

Result	Recommended Action
Failed, model incompatible	<p>Resolve the model incompatibility. See</p> <ul style="list-style-type: none"> <li>• “Supported and Unsupported Simulink Blocks in Simulink Design Verifier” (Simulink Design Verifier)</li> <li>• “Support Limitations for Model Blocks” (Simulink Design Verifier)</li> <li>• “Support Limitations for Simulink Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for Stateflow Software Features” (Simulink Design Verifier)</li> <li>• “Support Limitations for MATLAB for Code Generation” (Simulink Design Verifier)</li> </ul> <p>Also see “Handle Incompatibilities with Automatic Stubbing” (Simulink Design Verifier).</p>
Violation of minimum and/or maximum found in model	<p>To view the conditions that cause the violation, create a harness model. When you simulate the harness, the inputs replicate the error. Click <b>View test case</b> in the Model Advisor report.</p>

### See Also

- MISRA C:2012: Directive 4.1
- CERT C, API00-C
- CWE, CWE-628
- “Secure Coding Standards” (Embedded Coder)
- “Design Range Checks” (Simulink Design Verifier)

- “Check for Specified Minimum and Maximum Value Violations” (Simulink Design Verifier)

# Model Metrics

## Model Metrics

Model metrics analyze your model and help you assess your model with regard to size, architecture, readability, and compliance to standards. Simulink Check provides the metrics for these metric types:

- “Size Metrics” on page 2-419
- “Architecture Metrics” on page 2-420
- “Compliance Metrics” on page 2-421
- “Readability Metrics” on page 2-422

Using the Metrics Dashboard, you can collect and view model metrics to get an assessment of your project quality status. For more information, see “Collect and Explore Metric Data by Using the Metrics Dashboard”.

You can use the model metric API to run the model metrics programmatically and export the results to a file. For more information, see “Collect Model Metrics Programmatically”.

For your company guidelines and standards, you can also use the model metric API to create your own model metrics, compute those metrics, and export the metric data. For more information, see “Create a Custom Model Metric for Nonvirtual Block Count”.

## Size Metrics

To collect metric data on a model or subsystem, run these metrics.

Metric	Description
“Simulink block metric” on page 2-422	Calculates the number of blocks in the model.
“Subsystem metric” on page 2-424	Calculates the number of subsystems in the model.
“Library link metric” on page 2-425	Calculates the number of library-linked blocks in the model.

<b>Metric</b>	<b>Description</b>
“Effective lines of MATLAB code metric” on page 2-426	Calculates the number of effective lines of MATLAB code.
“Stateflow chart objects metric” on page 2-427	Calculates the number of Stateflow objects.
“Lines of code for Stateflow blocks metric” on page 2-429	Calculates the number of code lines for the following Stateflow blocks in the model: <ul style="list-style-type: none"><li>• States</li><li>• Transitions</li><li>• Truth tables</li></ul>
“Subsystem depth metric” on page 2-430	Calculates the subsystem depth of the model.
“Input output metric” on page 2-432	Calculates the number of inports and outports in your model.
“Explicit input output metric” on page 2-434	Calculates the number of inports and outports in your model.
“File metric” on page 2-435	Calculates the number of model and library files.
“Matlab Function metric” on page 2-436	Calculates the number of Matlab Function blocks in your model.
“Model file count” on page 2-437	Calculates the number of model files.
“Parameter metric” on page 2-438	Calculates the number of data objects that parameterize the behavior of a model.
“Stateflow chart metric” on page 2-439	Calculates the number of Stateflow charts in your model.

For more information on model metrics, see “Collect Model Metrics”.

## **Architecture Metrics**

To learn more about the architecture for a model or subsystem, run these metrics.

<b>Metric</b>	<b>Description</b>
“Cyclomatic complexity metric” on page 2-440	Calculates the cyclomatic complexity of the model.
“Clone content metric” on page 2-442	Calculates the fraction of total number of subcomponents that are clones.
“Clone detection metric” on page 2-443	Calculates the number of clones in components across the model hierarchy.
“Library content metric” on page 2-444	Calculates the fraction of total number of components that are linked library blocks.

For more information on model metrics, see “Collect Model Metrics”.

## Compliance Metrics

To determine if your model or subsystem is compliant with standards and guidelines, run one or more of these metrics.

<b>Metric</b>	<b>Description</b>
“MATLAB code analyzer warnings” on page 2-448	Determines warnings for MATLAB code blocks in your model.
“Diagnostic warnings metric” on page 2-433	Calculates the number of diagnostic warnings reported.
“Model Advisor Check Compliance for High-Integrity Systems” on page 2-449	Returns the fraction of checks the model passes from Model Advisor DO-178C/DO-331 Standards.
“Model Advisor Check Compliance for Modeling Standards for MAAB” on page 2-450	Returns the fraction of checks the model passes from Model Advisor MAAB Standard.
“Model Advisor Check Issues for High-Integrity Systems” on page 2-451	Reports the number of issues from Model Advisor DO-178C/DO-331 Standards.

Metric	Description
“Model Advisor Check Issues for MAAB Standards” on page 2-452	Reports the number of issues from Model Advisor MAAB Standard.

For more information on model metrics, see “Collect Model Metrics”.

### Readability Metrics

Run these metrics to determine readability for a model or subsystem.

Metric	Description
“Nondescriptive block name metric” on page 2-445	Determines nondescriptive Inport, Outport, and Subsystem block names.
“Data and structure layer separation metric” on page 2-446	Calculates the data and structure layer separation.

For more information on model metrics, see “Collect Model Metrics”.

### Simulink block metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.SimulinkBlockCount`

**Model Advisor Check ID:** `mathworks.metricchecks.SimulinkBlockCount`

Calculate the number of Simulink blocks in the model

#### Description

Use this metric to calculate the number of blocks in the model. The results provide the number of blocks at the model and subsystem level. This metric counts Simulink—based blocks, but does not include underlying blocks used to implement the block. This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Simulink block metric** in **By Task > Model Metrics > Count Metrics**.

- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.SimulinkBlockCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of blocks.
- **AggregatedValue:** Number of blocks for component and its subcomponents.
- **Measures:** Not applicable.

---

**Note** The results from metric analysis of **Simulink block metric** can differ from calling `sldiagnostics`. The result of the Simulink block metric:

- Includes referenced models.
  - Does not include any underlying blocks used to implement a MathWorks block that you used from the Simulink Library Browser.
  - Does not include links into MathWorks libraries, which means that MathWorks library blocks that are masked subsystems are counted as one block. The inner content of those blocks is not counted.
  - Does not include hidden content under Stateflow Charts or MATLAB Function blocks.
  - Does not include requirements blocks.
- 

## Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Subsystem metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.SubSystemCount`

**Model Advisor Check ID:** `mathworks.metricchecks.SubSystemCount`

Display number of subsystems in the model

### Description

Use this metric to calculate the number of subsystems in the model. The results provide the number of subsystems at the model and subsystem level.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Subsystem metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.SubSystemCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of subsystems.
- **AggregatedValue:** Number of subsystems for a component and its subcomponent.
- **Measures:** Not applicable.

### Capabilities and Limitations

The metric:



- Runs on library models.
- Analyzes content in masked subsystems.
- Does not count subsystems linked to MathWorks libraries.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Library link metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.LibraryLinkCount`

**Model Advisor Check ID:** `mathworks.metricchecks.LibraryLinkCount`

Display number of library links in the model

### Description

Use this metric to calculate the number of library-linked blocks in the model. The results provide the number of library-linked blocks at the model and subsystem level.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Library link metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.LibraryLinkCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of library linked blocks.
- **AggregatedValue:** Number of library linked blocks for a component and its subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- Does not count subsystems linked to MathWorks libraries.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Effective lines of MATLAB code metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.MatlabLOCCount`

**Model Advisor Check ID:** `mathworks.metricchecks.MatlabLOCCount`

Display number of effective lines of MATLAB code

### Description

Run this metric to calculate the number of effective lines of MATLAB code. Effective lines of MATLAB code are lines of executable code. Empty lines, lines that contain only comments, and lines that contain only an end statement are not considered effective lines of code. The results provide the number of effective lines of MATLAB code for each MATLAB Function block and for MATLAB functions in Stateflow charts.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Effective lines of MATLAB code metric** in **By Task > Model Metrics > Count Metrics**.

- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.MatlabLOCCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of effective lines of MATLAB code.
- **AggregatedValue:** Number of effective lines of MATLAB code for a component and its subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- Does not analyze the content of MATLAB code in external files.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Stateflow chart objects metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.StateflowChartObjectCount`

**Model Advisor Check ID:**

`mathworks.metricchecks.StateflowChartObjectCount`

Display the number of Stateflow objects in each chart

### Description

Run this metric to calculate the number of Stateflow objects. For each chart in the model, the results provide the number of the following Stateflow objects:

- Atomic subcharts
- Boxes
- Data objects
- Events
- Graphical functions
- Junctions
- Linked charts
- MATLAB functions
- Notes
- Simulink functions
- States
- Transitions
- Truth tables

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Stateflow chart objects metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.StateflowChartObjectCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of Stateflow objects.
- **AggregatedValue:** Number of Stateflow objects for a component and its subcomponents.

- Measures: Not applicable.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Lines of code for Stateflow blocks metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.StateflowLOCCount`

**Model Advisor Check ID:** `mathworks.metricchecks.StateflowLOCCount`

Display the number of effective lines of code for Stateflow blocks

### Description

Use this metric to calculate the number of effective lines of code in Stateflow. Effective lines of MATLAB code are lines of executable code. Empty lines, lines that contain only comments, and lines that contain only an end statement are not considered effective lines of code. This metric calculates the lines of code for the following Stateflow blocks in the model:

- Chart, counting the code on Transitions and inside States
- State Transition Table block
- Truth Table block

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Lines of code for Stateflow blocks metric** in **By Task > Model Metrics > Count Metrics**.

- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.StateflowLOCCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of Stateflow block code lines.
- **AggregatedValue:** Number of Stateflow block code lines for a component and its subcomponents.
- **Measures:** Vector with two entries: number of effective lines of code in MATLAB action language and number of effective lines of code in C action language.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Subsystem depth metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.SubSystemDepth`

**Model Advisor Check ID:** `mathworks.metricchecks.SubSystemDepth`

Calculates the maximum depth of all hierarchical children of a subsystem or model

## Description

Use this metric to count the maximum depth of all hierarchical children for a given subsystem or model starting from the given component, or root of analysis. The depth is the relative depth of the deepest branch. Depth traversal analysis stops when it reaches a referenced model or a library. Depth and level are restarted with 0 for each of these components.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Subsystem depth metric** in **By Task > Model Metrics > Count Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.SubSystemDepth`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: None`
- `slmetric.metric.AggregateComponentDetails: false`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** subsystem depth for each component in the hierarchy.
- **AggregatedValue:** Not applicable.
- **Measure:** level of component in the hierarchy.
- **AggregatedMeasure:** Not applicable.

## Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

## See Also

For more information on model metrics, see “Collect Model Metrics”.

## Input output metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.IOCount`

Display number of inputs and outputs in the model

### Description

Use this metric to calculate the number of inputs and outputs in the model, which include:

- Inputs: Inport blocks, Trigger ports, Enable ports, chart input data and events.
- Outputs: Outport blocks, chart output data and events.
- Implicit inputs: From block, where the matching Goto block is outside of the component.
- Implicit outputs: Goto block, where the matching From block is outside of the component.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Max`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** total interface size or sum of the elements of `Measures`.
- **AggregatedValue:** Number of inputs and outputs for a component and its subcomponents.
- **Measures:** Array consisting of number of inputs, number of outputs, number of implicit inputs, and number of implicit outputs, which are local to the component.
- **AggregatedMeasures:** Maximum number of inputs, outputs, implicit inputs, and implicit outputs for a component and subcomponents.

### Capabilities and Limitations

The metric:



- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Diagnostic warnings metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.DiagnosticWarningsCount`

Calculate the number of diagnostic warnings reported during a model update for simulation.

### Description

Use this metric to calculate the number of Simulink diagnostic warnings reported during a model update for simulation. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.DiagnosticWarningsCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of diagnostic warnings reported.
- **AggregatedValue:** Number of diagnostic warnings reported for component and its subcomponents.
- **Measure:** Not applicable.

### Capabilities and Limitations

- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Explicit input output metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.ExplicitIOCount`

Display number of inputs and outputs in the model, excluding From and Goto blocks.

### Description

Use this metric to calculate the number of inputs and outputs in the model, which include:

- Inputs: Inport blocks, Trigger ports, Enable ports, chart input data and events.
- Outputs: Outport blocks, chart output data and events.

This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.ExplicitIOCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Max`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Total interface size or sum of the elements of `Measures`.
- **AggregatedValue:** Number of inputs and outputs for a component and its subcomponents.
- **Measures:** Array consisting of number of inputs and number of outputs which are local to the component.

- **AggregatedMeasures:** Maximum number of inputs and outputs for a component and subcomponents.

### Capabilities and Limitations

The metric:

- Excludes From and Goto blocks.
- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## File metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.FileCount`

Calculates the number of model and library files used by a specific component and its subcomponents.

### Description

Use this metric to count the number of model and library files used by a specific component and its subcomponents. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.FileCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: None`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of model and library files.
- **AggregatedValue:** Not applicable.
- **Measures:** Not applicable.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Matlab Function metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.MatlabFunctionCount`

Calculates the number of Matlab Function blocks inside a component.

### Description

Use this metric to count the number of Matlab Function blocks inside a component. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.MatlabFunctionCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of Matlab Function blocks.
- **AggregatedValue:** Number of Matlab Function blocks for component and its subcomponents.

- Measures: Not applicable.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Model file count

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.ModelFileCount`

Calculate the number of model files.

### Description

Use this metric to count the number of model files. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.ModelFileCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: None`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of files reference by a component and its subcomponents.
- **AggregatedValue:** Not applicable.
- **Measures:** Not applicable.

### Capabilities and Limitations

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Parameter metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.ParameterCount`

Calculate the number of parameters.

### Description

Use this metric to calculate the amount of user-managed parameterization data inside a Simulink system. A parameter is a variable used by a Simulink block or object of a basic type (`single`, `double`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `boolean`, `logical`, `struct`, `char`, `cell`), `Simulink.Parameter`, `Simulink.Variant`, or `enum` value. The parameter can be stored in either the base workspace, the model workspace, or a data dictionary.

This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.ParameterCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode`: `Sum`
- `slmetric.metric.AggregateComponentDetails`: `true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of parameters used inside a component.

- **AggregatedValue:** Number of parameters for a component and its subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

This metric:

- Uses the `Simulink.findVars` function and inherits the limitations of this function.
- Counts the parameter instances in a component rather than unique parameters.
- Does not include parameters in masked workspaces.
- Does not include data type and signal objects.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Stateflow chart metric

**Metric Type:** Size

**Metric ID:** `mathworks.metrics.StateflowChartCount`

Calculate the number of Stateflow charts at any component level.

### Description

Use this metric to count the number of Stateflow charts at any component level. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.StateflowChartCount`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of Stateflow charts at the model level.
- **AggregatedValue:** Number of charts for component and its subcomponents.
- **Measures:** Not applicable.

**Capabilities and Limitations**

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

**See Also**

For more information on model metrics, see “Collect Model Metrics”.

**Cyclomatic complexity metric**

**Metric Type:** Architecture

**Metric ID:** `mathworks.metrics.CyclomaticComplexity`

**Model Advisor Check ID:** `mathworks.metricchecks.CyclomaticComplexity`

Display the local and aggregated cyclomatic complexity of the model

**Description**

Use this metric to calculate the cyclomatic complexity of the model. Cyclomatic complexity is a measure of the structural complexity of a model. It approximates the McCabe complexity measure for code generated from the model. The McCabe complexity measure is slightly higher on the generated code than on the model due to error checks that this analysis does not consider. To compute the cyclomatic complexity of an object (such as a block, chart, or state), Simulink Check uses this formula:

$$c = \sum_{1}^{N} (o_n - 1)$$



$N$  is the number of decision points that the object represents and  $o_n$  is the number of outcomes for the  $n$ th decision point. The tool adds 1 to the complexity number for atomic subsystems and Stateflow charts.

The results provide the local and aggregated cyclomatic complexity for the:

- Model
- Subsystems
- Charts
- MATLAB functions

Local complexity is the cyclomatic complexity for objects at their hierarchical level. Aggregated cyclomatic complexity is the cyclomatic complexity of an object and its descendants

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Cyclomatic complexity metric** in **By Task > Model Metrics > Complexity Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.CyclomaticComplexity`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Local cyclomatic complexity.
- **AggregatedValue:** Aggregated cyclomatic complexity.
- **Measures:** Not applicable.

## Capabilities and Limitations

The metric:

- Does not run on library models.

- Analyzes content in masked subsystems.
- Does not analyze inactive variants.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Does not analyze referenced models in accelerated mode.

### See Also

- “Collect Model Metrics”
- “Cyclomatic Complexity for Stateflow Charts” (Simulink Coverage)
- “Specify Coverage Options” (Simulink Coverage)

## Clone content metric

**Metric Type:** Architecture

**Check ID:** `mathworks.metrics.CloneContent`

Calculates the fraction of total number of subcomponents that are clones.

### Description

Use this metric to calculate the fraction of the total number of subcomponents that are clones. Clones must have identical block types and connections but they can have different parameter values. For more information on clone detection, see “Enable Component Reuse by Using Clone Detection”.

This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.CloneContent`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: None`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Fraction of total number of subcomponents that are clones

- **AggregatedValue:** Not applicable.
- **Measures:** Vector containing number of clones, total number of components, and clone group number.

### **Capabilities and Limitations**

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### **See Also**

For more information on model metrics, see “Collect Model Metrics”.

## **Clone detection metric**

**Metric Type:** Architecture

**Check ID:** `mathworks.metrics.CloneDetection`

Calculate the number of clones in a model.

### **Description**

Use this metric to count the number of clones in a model. Clones must have identical block types and connections but they can have different parameter values. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.CloneDetection`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: false`

### **Results**

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of clones.
- **AggregatedValue:** Number of clones for component and its subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”.

## Library content metric

**Metric Type:** Architecture

**Check ID:** `mathworks.metrics.LibraryContent`

Calculates the fraction of total number of components that are linked library blocks.

### Description

Use this metric to calculate the fraction of total number of components that are linked library blocks. This metric is available with Simulink Check. To collect data for this metric, use `getMetrics` with the metric identifier, `mathworks.metrics.LibraryContent`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: None`
- `slmetric.metric.AggregateComponentDetails: false`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Fraction of the total number of subcomponents that are linked library blocks.
- **AggregatedValue:** Not applicable.
- **Measures:** Vector containing the number of linked library blocks and total number of components

### Capabilities and Limitations

- If specified, analyzes the content of library-linked blocks or referenced models.

## See Also

For more information on model metrics, see “Collect Model Metrics”.

## Nondescriptive block name metric

**Metric Type:** Readability

**Check ID:** `mathworks.metrics.DescriptiveBlockNames`

**Model Advisor Check ID:** `mathworks.metricchecks.DescriptiveBlockNames`

Display nondescriptive Inport, Outport, and Subsystem block names

### Description

Run this metric to determine nondescriptive Inport, Outport, and Subsystem block names. Default names appended with an integer are nondescriptive block names. The results provide the nondescriptive block names at the model and subsystem levels.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Nondescriptive block name metric** in **By Task > Model Metrics > Readability Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.DescriptiveBlockNames`.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of nondescriptive Inport, Outport, and Subsystem block names.
- **AggregatedValue:** Number of nondescriptive Inport, Outport, and Subsystem block names for a component and its subcomponents.
- **Measures:** 1-D vector containing:

- Total number of Inport blocks
- Number of Inport blocks with nondescriptive names
- Total number of Outport blocks
- Number of Outport blocks with nondescriptive names
- Total number of Subsystem blocks
- Number of Subsystem blocks with nondescriptive names
- AggregatedMeasures: 1-D vector containing sum of:
  - Total number of Inport blocks
  - Number of Inport blocks with nondescriptive names
  - Total number of Outport blocks
  - Number of Outport blocks with nondescriptive names
  - Total number of Subsystem blocks
  - Number of Subsystem blocks with nondescriptive names

### Capabilities and Limitations

The metric:

- Does not run on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”

## Data and structure layer separation metric

**Metric Type:** Readability

**Metric ID:** `mathworks.metrics.LayerSeparation`

**Model Advisor Check ID:** `mathworks.metricchecks.LayerSeparation`

Display data and structure layer separation

## Description

Run this metric to calculate the data and structure layer separation. The results provide the separation at the model and subsystem level.

Run this metric to calculate the data and structure layer separation. The results provide the separation at the model and subsystem levels.

This metric is available with Simulink Check. To collect data for this metric:

- Using the Model Advisor, run the check, **Data and structure layer separation metric** in **By Task > Model Metrics > Readability Metrics**.
- Programmatically, use `getMetrics` with the metric identifier, `mathworks.metrics.LayerSeparation`.

For guidelines about blocks on model levels, see the MAAB 3.0 guideline db\_0143: Similar block types on the model levels.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of basic blocks on a structural level.
- **AggregatedValue:** Number of basic blocks on a structural level for a component and its subcomponents.
- **Measures:** Not applicable.

## Capabilities and Limitations

The metric:

- Does not run on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.

### See Also

For more information on model metrics, see “Collect Model Metrics”

## MATLAB code analyzer warnings

**Metric Type:** Compliance

**Metric ID:** `mathworks.metrics.MatlabCodeAnalyzerWarnings`

Use this metric to calculate the number of MATLAB code analyzer warnings from MATLAB code in the model. This metric is available with Simulink Check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of MATLAB code analyzer warnings
- **AggregatedValue:** Number of MATLAB code analyzer warnings aggregated for a component and subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

The metric:

- Analyzes MATLAB code in MATLAB Function blocks
- Analyzes MATLAB functions in Stateflow charts
- Runs on library models
- Analyzes content in masked subsystems
- If specified, analyzes content of library-linked blocks and referenced models
- Does not analyze external MATLAB code files



## See Also

- “Collect Model Metrics”
- “Check Code for Errors and Warnings” (MATLAB)

## Model Advisor Check Compliance for High-Integrity Systems

**Metric Type:** Compliance

**Metric ID:** `mathworks.metrics.ModelAdvisorCheckCompliance.hisl_do178`

Use this metric to calculate the fraction of Model Advisor checks that pass for the **High-Integrity Systems** subgroups. This metric is available with Simulink Check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Percentile`
- `slmetric.metric.AggregateComponentDetails: true`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Fraction of total number of checks passed in **High-Integrity Systems** subgroups.
- **AggregatedValue:** Fraction of total number of checks passed in **High-Integrity Systems** subgroups aggregated for a component and all of its subcomponents.
- **Measures:** Vector containing: number of checks passed in subgroups and number of checks in subgroups.
- **AggregatedMeasures:** Vector containing: number of checks passed in subgroups and number of checks in subgroup, for a component and all its subcomponents.

## Results Details

For this metric, instances of the `slmetric.metric.ResultDetail` `Value` property provides these results:

- A value of 0 indicates that a check did not run.
- A value of 1 indicates that a check passed.

- A value of 2 indicates a check warning.
- A value of 3 indicates a failure.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

### See Also

- “Collect Model Metrics”
- “Model Checks for DO-178C/DO-331 Standard Compliance”

## Model Advisor Check Compliance for Modeling Standards for MAAB

**Metric Type:** Compliance

**Metric ID:** `mathworks.metrics.ModelAdvisorCheckCompliance.maab`

Use this metric to calculate the fraction of Model Advisor checks that pass for the group **Modeling Standards for MAAB**. This metric is available with Simulink Check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Percentile`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Fraction of total number of checks passed in MAAB.
- **AggregatedValue:** Fraction of total number of checks passed in MAAB aggregated for a component and all of its subcomponents.

- **Measures:** Vector containing: number of checks passed in group and number of checks in group.
- **AggregatedMeasures:** Vector containing: number of checks passed in group and number of checks in group, for a component and all its subcomponents.

### Results Details

For this metric, instances of the `slmetric.metric.ResultDetail` Value property provides these results:

- A value of 0 indicates that a check did not run.
- A value of 1 indicates that a check passed.
- A value of 2 indicates a check warning.
- A value of 3 indicates a failure.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

### See Also

- “Collect Model Metrics”
- “Model Checks for MathWorks Automotive Advisory Board (MAAB) Guideline Compliance”

## Model Advisor Check Issues for High-Integrity Systems

**Metric Type:** Compliance

**Metric ID:** `mathworks.metrics.ModelAdvisorCheckIssues.hisl_do178`

Use this metric to calculate number of issues reported by the subgroups of Model Advisor checks for **High-Integrity Systems**. This metric counts each Model Advisor check that produces a warning or failure. If a check contains links to blocks, this metric counts one

issue for each linked block. Checks with links to the model are highlighted in the Simulink Editor. If a check does not contain links to blocks, this metric counts one issue. This metric is available with Simulink Check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

### Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of issues reported by the **High-Integrity Systems** checks
- **AggregatedValue:** Number of issues reported by the **High-Integrity Systems** checks aggregated for a component and all of its subcomponents.
- **Measures:** Not applicable.

### Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.

### See Also

- “Collect Model Metrics”
- “Model Checks for DO-178C/DO-331 Standard Compliance”

## Model Advisor Check Issues for MAAB Standards

**Metric Type:** Compliance

**Metric ID:** `mathworks.metrics.ModelAdvisorCheckIssues.maab`

Use this metric to calculate number of issues reported by the group of Model Advisor checks for **Modeling Standards for MAAB**. This metric counts each Model Advisor

check that produces a warning or failure. If a check contains links to blocks, this metric counts one issue for each linked block. Checks with links to the model are highlighted in the Simulink Editor. If a check does not contain links to blocks, this metric counts one issue. This metric is available with Simulink Check.

Aggregation properties for this metric are set to:

- `slmetric.metric.AggregationMode: Sum`
- `slmetric.metric.AggregateComponentDetails: true`

## Results

For this metric, instances of `slmetric.metric.Result` provide the following results:

- **Value:** Number of issues reported by the Model Advisor for MAAB checks.
- **AggregatedValue:** Number of issues reported by the Model Advisor for MAAB checks aggregated for a component and its subcomponents.
- **Measures:** Not applicable.

## Capabilities and Limitations

The metric:

- Runs on library models.
- Analyzes content in masked subsystems.
- If specified, analyzes the content of library-linked blocks or referenced models.
- Analyzes content in Stateflow objects.
- Adds check issues on the configuration set or issues with data objects to the issue count at the model root level.

## See Also

- “Collect Model Metrics”
- “Model Checks for MathWorks Automotive Advisory Board (MAAB) Guideline Compliance”
- “Run Model Advisor Checks and Review Results”

## **See Also**

### **Related Examples**

- *“Collect Model Metrics Using the Model Advisor”*
- *“Collect Model Metrics Programmatically”*
- *“Model Metric Data Aggregation”*
- *“Create a Custom Model Metric for Nonvirtual Block Count”*

# Model Transformer Tasks

---

# Model Transformer Tasks

In this section...
“Transformations” on page 3-2
“Transform the model to variant system” on page 3-3
“Eliminate Data Store Blocks” on page 3-4
“Transform table lookup into prelookup and interpolation” on page 3-5

Use the Model Transformer tool to refactor a model to implement variants, eliminate eligible data store blocks, and improve the simulation efficiency of table lookup operations. You can perform the steps in the Model Transformer all at once or one step at a time.

## Transformations

Use the Model Transformer tool to create models that contain these transformations:

- Replace qualifying modeling patterns with variant blocks.
- Replace data store blocks with blocks that make data dependency explicit.
- Replace n-D Lookup Table blocks with shared Prelookup blocks and multiple Interpolation blocks.

If you want to perform the transformations at once, for each step, specify the input parameters. Then, click the **Run Selected Checks** button. After you run each check, create new models with the transformations by clicking the **Refactor Model** buttons.

If you want to perform one transformation at a time, you can individually select the checks.

### See Also

- “Transform Model to Variant System”
- “Replace Data Store Blocks”



## Transform the model to variant system

Click the **Run This Check** button to identify system constants for use in variant transformations and blocks that qualify for transformation into Variant Subsystem or Variant Source blocks. These transformations are possible:

- If an If block connects to one or more If Action Subsystems and each If Action Subsystem has one output, replace this modeling pattern with a subsystem and a Variant Source block.
- If an If block connects to an If Action Subsystem that has no output or two or more outputs, replace this modeling pattern with a Variant Subsystem block.
- If a Switch Case block connects to one or more Switch Case Action Subsystems and each Switch Case Action Subsystem has one output, replace this modeling pattern with a subsystem and a Variant Source block.
- If a Switch Case block connects to a Switch Case Action Subsystem that has no output or two or more outputs, replace this modeling pattern with a Variant Subsystem block.
- Replace a Switch block with a Variant Source block.
- Replace a Multiport Switch block that has two or more data ports with a Variant Source block.

A system constant is the control input or is part of an arithmetic expression that forms the control input to Multiport Switch or Switch blocks and the inputs to If or Switch Case blocks. The control input must be Constant blocks and some combination of blocks that form a supported MATLAB expression. In the Constant block parameters dialog box, the **Constant value** parameters are the system constants. In the transformed model, system constants are part of condition expressions in Variant Source or Variant Subsystem blocks.

---

**Note** For some model patterns and settings, the Model Transformer cannot perform every one of the preceding transformations.

---

In the **Result** table, each modeling pattern is a hyperlink to the corresponding location in the model. If you do not want the Model Transformer to perform a transformation, clear the check box next to the qualifying pattern.

Click the **Refactor Model** button to create a model that contains the transformations. The transformed model is in the folder that has the prefix m2m plus the original model name.

#### See Also

- “Transform Model to Variant System”

## Eliminate Data Store Blocks

Click the **Run This Check** button to identify Data Store Memory, Data Store Read, and Data Store Write blocks that qualify for elimination. Click the **Refactor Model** button to create a model that replaces these blocks with either a direct signal line, a Delay block, or a Merge block. The model is in the folder that has the prefix m2m plus the original model name.

Replacing these blocks improves model readability by making data dependency explicit. The Model Transformer can replace these data stores:

- For signals that are not buses, if a Data Store Read block executes before a Data Store Write block, the tool replaces these blocks with a Delay block.
- For signals that are not buses, if a Data Store Write block executes before a Data Store Read block, the tool replaces these blocks with a direct connection.
- For bus signals, if the write to bus elements executes before the read of the bus, the tool replaces the Data Store Read and Data Store Write blocks with a direct connection and a Bus Creator block.
- For bus signals, if the write to the bus executes before the read of bus elements, the tool replaces the Data Store Read and Data Store Write blocks with a direct connection and a Bus Selector block.
- For conditionally executed subsystems, the tool replaces the Data Store Read and Data Store Write blocks with a direct connection and a Merge block.

The Model Transformer tool only eliminates local data stores that Data Store Memory blocks define. The tool does not eliminate global data stores. For the Data Store Memory block, on the **Signal Attributes** tab of the block parameters dialog box, the **Data store name must resolve to Simulink signal object** parameter must be cleared.

The **Result** table contains hyperlinks to the corresponding Data Store Memory, Data Store Read, and Data Store Write blocks. If you do not want the Model Transformer to

perform a transformation, before clicking the **Refactor Model** button, clear the check box next to the Data Store Memory block.

### See Also

- “Replace Data Store Blocks”

## Transform table lookup into prelookup and interpolation

To identify n-D Lookup Table blocks that qualify for transformation into shared Prelookup blocks and multiple Interpolation blocks, click the **Run This Check** button. The tool displays the identified blocks and their algorithm parameters along with the model, libraries, and referenced models. You can also clear the blocks for transformation.

The Model Transformer works if:

- The same source drives the Lookup Table blocks.
- The Lookup table blocks share the same breakpoint specification, values, and data types.
- The Lookup table blocks share the same algorithm parameters in the block parameters dialog box.
- The Lookup table blocks share the same data type for fractions parameters in the block parameters dialog box.

To create a model that replaces these Lookup Table blocks with a shared Prelookup and Interpolation blocks, click the **Run This Check** button.

Eliminating the redundant Prelookup blocks in the equivalent model generated improves the simulation speed for linear interpolations.

### See Also

- n-D Lookup Table
- “Improve Efficiency of Simulation by Optimizing Prelookup Operation of Lookup Table Blocks”



# Clone Detection Tasks

---

# Clone Detection Checks

Use the Clone Detector app to refactor a model by identifying clones and creating models that replace clones with links to subsystem blocks in a library.

## Exclude subsystems and referenced models from clone detection

To save time during model development, you can limit the scope of the clone detection analysis of your model. You can use the Clone Detection Exclusion Editor under **Exclude Components** to exclude Subsystem and Model Reference blocks from clone detection. To exclude a subsystem or referenced model, right-click the subsystem or referenced model and select **Identify Modeling Clones > Subsystem and its contents > Add to exclusions**.

After you specify the Subsystem or Model Reference blocks to exclude, the Clone Detector App uses the exclusion information to exclude blocks during analysis. By default, the exclusion information is stored in the model SLX file. Alternately, you can store the information in an exclusion file. To use an exclusion file, in the Clone Detection Exclusion Editor dialog box, clear **Store exclusions in model file**. The **Exclusion File** field is enabled.

The **Exclusion File** contains the exclusion file name and location associated with the model. You can use an exclusion file with several models. However, a model can have only one exclusion file.

Unless you specify a different folder, the Clone Detection Exclusion Editor saves the exclusion files in the current folder. The default name for an exclusion file is `<model_name>_exclusions.xml`.

If you create an exclusion file and save your model, you attach the exclusion file to your model. Each time that you open the model, the blocks specified in the exclusion file are excluded from the analysis.

To view exclusion information for a model, right-click a subsystem or Model Reference block and select **Identify Modeling Clones > Open Clone Detection Exclusion Editor**. For each subsystem or referenced model that you exclude from detection, in the Rationale field, you can provide a reason for why you are excluding it.

You may also choose to enable **Exclude library links** which excludes subsystems that are external library links and **Exclude inactive and commented out Regions** which

excludes modeling subsystems that are inactive during simulation such as variant subsystems.





# Simulink Check Apps

---

## Clone Detector

Enable model refactorization and subsystem reuse in models by identifying and replacing clones with library links

### Description

The **Clone Detector** is a tool that identifies and replaces clones, which are modeling patterns that have identical block types and connections. The tool identifies clones across referenced model boundaries. You can refactor your model by replacing the clones with library links, which enables you to reuse components.

The Clone Detector interface steps you through the process of identifying and refactoring clones. You can:

- Access clone reuse percentages in the model to determine the benefits of refactoring.
- Vary parameter tolerance to identify similar and exact clones.
- Identify the differences in a clone from the baseline subsystem.
- Access a log of clone detection results.
- Use a Simulink Test license to access an embedded Test Manager that allows you to verify the equivalence of the refactored model and the original model.



To identify and replace clones in the model:

- 1 Open the model `rtwdemo_abstime`.
- 2 Save the model to your working folder.
- 3 On the **Apps** tab, click **Clone Detector**.
- 4 In the Clone Detector tab, on the **View** section, you can select **Help** pane, **Properties** to show the Detected Clone Properties pane, or **Results** to show the Clone Detection Actions and Results pane.

- 5 Click **Find Clones** to identify clones.
- 6 Click the Clone Detection Action and Results pane and browse the results to view the identified clones.
- 7 Click **Replace Clones** to replace the clone groups. A backup model with the original layout is saved. Click the **Restore** button in the clone detection logs to revert to the original model.
- 8 Click **Check Equivalency** to open the Test Manager. This tests whether the refactored model is functionally equivalent to the original model.

## Open the Clone Detector App

In the **Apps** gallery, click **Clone Detector**.

## Examples

- “Custom Libraries” (Simulink)
- “Generate Reusable Code from Library Subsystems Shared Across Models” (Simulink Coder)
- “Enable Component Reuse by Using Clone Detection”

## See Also

### Topics

“Custom Libraries” (Simulink)

“Generate Reusable Code from Library Subsystems Shared Across Models” (Simulink Coder)

“Enable Component Reuse by Using Clone Detection”

**Introduced in R2019b**

