



VI-CarRealTime 20.0 Documentation

www.vi-grade.com

email: info@vi-grade.com

VI-CarRealTime 20.0 Documentation

Copyright Information

VI-grade GmbH

VI-Aircraft, VI-Animator, VI-Automotive, VI-BikeRealTime, VI-CarRealTime, VI-Driver, VI-DriveSim, VI-EventBuilder, VI-GraphSim, VI-MotionCueing, VI-Motorcycle, VI-Rail, VI-Road, VI-SuspensionGen, VI-Tire, VI-TireLimits

Copyright 2006-2020, VI-grade GmbH, Darmstadt, Germany.

This software contains confidential and proprietary information of VI-grade GmbH. All rights reserved. This code may not be copied or reproduced in any form, in part or in whole, without the explicit, prior written permission of the copyright owner. Third-party software is copyrighted and licensed from VI-grade GmbH suppliers.

This software may include libraries licensed under LGPL terms.

Trademarks

VI-Aircraft, VI-Animator, VI-Automotive, VI-BikeRealTime, VI-CarRealTime, VI-Driver, VI-DriveSim, VI-EventBuilder, VI-GraphSim, VI-MotionCueing, VI-Motorcycle, VI-Rail, VI-Road, VI-SuspensionGen, VI-Tire, VI-TireLimits are trademarks of VI-grade GmbH or of one of its subsidiaries.

Python is a registered trademark of the Python Software Foundation

All other trademarks referenced herein are property of their respective holders.

Printed: July 2020

Table of Contents

Part 1 VI-CarRealTime	21
1 Introduction	21
Software Architecture	21
2 VI-CarRealTime Environment	23
Window Elements	24
Toolbars	25
Treeview	26
Using	26
Filtering	29
Sorting	30
Property Editor	30
Edit Menu	32
Preferences	32
Registering Databases	35
Database Structure	35
Sessions	36
GUI Settings	37
Build Mode	38
Vehicle Model	
Conventions and references	40
Model Configuration	42
Properties	43
Custom Solver	44
Plugin	44
Driver Parameters	44
Speedgen Parameters	45
Sevenpostrig Parameters	47
Suspension Testrig Parameters	48
Live Animation	48
Solver Executive Control	49
ABS	52
TCS	52
Simulink	53
External Suspension	53
System Parameters	54
Trailer Parameters	54
Aerodynamics Parameters	55
Powertrain Parameters	55
Dampers Parameters	55
Anti Roll Bar Parameters	55
Body Compliance Parameters	56
Differential Parameters	56
Ride Height Maps Update Parameters	56
Statics Parameters	57
Setup Parameters	58
STD_TIR_REF Parameters	59
Circular Buffer	59
Output Files	59
Path Sensor	60
Output Channels	60
Clone Model	61

Suspension Subsystem	63
Wheel Location	66
Wheel Orientation	68
Compliance	70
Springs	74
Coil Springs	74
Air Springs	76
Dampers	79
Bumpers	82
Auxiliary Anti Roll Force	84
Anti-Roll Bar	85
Auxiliary Vertical Force	87
Suspension Setup Data	87
Adjustments	90
Installation Stiffness	92
Auxiliary Translational DOF	94
Steering Subsystem	98
Steering System	98
Kinematics	99
Compliance	101
Kingpin	102
Steering Feedback	105
Inertia and Damping	109
Auxiliary Spline Data	110
Rack-Pinion Steering	111
Mechanical Properties	112
Geometry	114
Column	117
Torsion Bar	120
Rack	122
Friction Models	123
Steering Assist	126
Solver	133
Components	134
Kinematic Data	134
Kingpin Moment Contribute	135
Force Data	137
Body Subsystem	137
Sprung Mass	138
Vehicle Graphics	140
Sensor Point	142
Adams Car CG Point	143
Aerodynamic Forces	144
Aerodynamic	146
General Parameters	146
Standard Aerodynamic	147
Front Downforce	147
Rear Downforce	148
Drag Force	148
Advanced Aerodynamic	148
Equations	151
Vertical forces	151
Longitudinal (drag) force	152
Lateral (side) forces	153
Rolling torque	153
Pitch (drag) torque	154
Body Setup Data	155
Cross Weight Adjustment	156

Contents

Ride Height Maps	157
User Sensor	158
Lap and Beacon Sensor	159
Displacement	160
Velocity	163
Acceleration	165
Rigid Part	167
Mass and Geometry	167
Solver	169
Bushings	169
Standard Bushing	171
Frequency Bushing	172
VI-MxMount Bushing	175
Skidplate Point	176
Powertrain Subsystem	177
Driveline	177
Driveline Layout	179
Engine	181
Engine Data	182
Ram Air	185
Fuel Consumption	186
Motors	187
Battery	190
Clutch/Torque Converter	192
Clutch	192
Torque Converter	194
Transmission	196
Manual	197
Automatic	198
Differentials	199
Front Differential	203
Rear Differential	203
Central Differential	204
Engine Part	204
Mass and Geometry	205
Solver	206
Bushings	206
Standard Bushing	208
Frequency Bushing	209
VI-MxMount Bushing	212
Wheels Subsystem	213
Rear Twin Wheels Subsystem	214
Brakes Subsystem	215
Drum Brakes	217
Auxiliary Subsystem	219
Rear Steering Subsystem	220
Adding external output to VI-CarRealTime	220
Cab Subsystem	222
VI-TireScrub	226
Tire Scrub Effect	226
Introduction	226
VI-TireLimits	227
Physical Dependencies and Tuning Parameters	228
Enabling Scrub Effect in VI-CarRealTime	235
Trailer Model	239
Conventions and references	239
Model Configuration	240
Properties	240

ABS	241
System Parameters	241
Static Parameters	242
STD_TIR_REF Parameters	242
Dampers Parameters	242
Hitch Ball Connection Parameters	242
Environment Parameters	243
Solver Executive Control	243
Output Channels	243
Body Subsystem	244
Vehicle Graphics	245
Aerodynamics	246
Sprung Mass	247
Payload	249
Adjustable Weight Distribution Hitch	250
Trailer Sway Control	252
Suspension Subsystem	253
Wheel Location	253
Wheel Orientation	255
Compliance	257
Springs	261
Coil Spring	261
Air Spring	263
Dampers	266
Bumpers	269
Auxiliary Anti Roll Force	271
Auxiliary Vertical Force	272
Wheels Subsystem	273
Brakes Subsystem	274
Example models and databases	274
Test Mode	275
Events	276
Adams Car	281
Driving Machine	281
Event Builder	283
Mini-Maneuvers	288
Steering	289
Other Tabs	290
Conditions tab	292
VIDriver	294
Cornering Events	294
Braking In Turn	294
Constant Radius Cornering	295
CourseEvents	296
MaxPerformance	296
Save/Restore	300
Matlab/Simulink interface	300
Callback m-functions	301
Custom logic	301
Path Compensation	302
Press Maneuvers	302
DLC - ISO 3888-1	307
DLC - CR	308
Obstacle Avoidance - ISO 3888-2	309
Slalom	309
Custom	310
Matlab/Simulink interface	311
Callback m-functions	312

Contents

FileDriven	313
Open Loop Steering Events	315
Impulse Steer	316
Ramp Steer	316
Sine Steer	317
Step Steer	318
Swept Steer	319
Stability	320
Fishhook	320
JTurn	322
SineWithDwell	323
Straight Line Events	324
Straight Acceleration	324
Straight Braking	325
VI-Driver Theory	326
VI-Driver Basics	326
Preview Time	327
Anticipatory Compensation	329
Yaw Rate Correction	331
Additional Controllers	331
StandStill Startup	331
Longitudinal Slip Controller	333
Human Driver	334
Cognitive Model	335
Actual Implementation	336
VISafety	338
Misuse	339
Curb Trip Rollover	341
VISpeedGen	344
StaticLapTime	344
VI-SpeedGen Theory	347
VI-SpeedGen	347
Speed Profile Computation	348
VI-SpeedGenEvo	350
Iterative Statics	351
VICTestRig	351
Sevenpostrig Events	351
Analytic	353
File	354
Suspension Testrig	354
Xternal	356
VI-DriveSim	356
VirtualTestDrive	357
Model Setup	358
Running simulation	359
SCANeR	360
Fingerprints	360
Run Analysis In Batch Mode	362
Static Analysis And Vehicle Setup	363
Setup Procedure	366
Straight Setup	367
Startup On Straight Line	368
Startup On Generic Path	369
Tire Limits	369
Review Mode	371
Solver Output Files	373
Custom PostProcessing Script	374
reportUtils Library	377

Contents

Investigation Mode	378
Investigations	379
Candidates	382
Factors	384
Responses	389
Using Custom Python Script	391
Global Settings	392
Run Investigation	394
Summary Report	395
Utilities	398
Database Utilities	398
Tire Testrig	400
Running Tire Testrig Realtime	402
Vehicle Wizard	404
General Parameters & Subsystem Files	405
Mass and Aerodynamics	406
Suspensions and Tires	407
Powertrain	408
Brakes	409
Auxiliary Sensors/Loads	409
K&C Wizard	413
Main Toolbar	414
Input Tabs	415
Vehicle Parameters	415
Mass and Geometry	415
Suspensions and Wheels	416
Powertrain	417
Brakes	418
Steering	418
Loadcase Channels	418
Channels Description	419
Loadcases	420
Loadcase Types	421
Kinematics Loadcases	421
Roll Loadcases	422
Compliance Loadcases	422
Steering Loadcases	428
Test Case Analysis	429
Suspension Kinematics Load Cases	430
Suspension Roll Load Cases	433
Suspension Compliance Load Cases	433
Steering Kinematics Load Cases	435
Data Management	435
Settings	436
Fitting	436
Data Fitting Methodology	437
Report File	437
Testing Conditions	438
Computation Settings	439
K&C Data Breakdown Into VI-CarRealTime Model	439
Plotting Area	441
Message Window	442
Trailer Wizard	442
General Parameters & Subsystem Files	443
Mass and Aerodynamics	444
Suspensions and Tires	446
Brakes	447
Property File Obfuscation	448

Contents

Obfuscate send_svm.xml	449
VDF Converter	450
Maneuvers	451
Longitudinal controllers	451
Openloop signals	451
Machine controller	452
End conditions block	453
CarSim converter	453
CarMaker converter	454
3 Model Interface	456
Input Channels	457
Driver Demands	458
Subsystem Activation Flags	459
Brake System General	462
ABS	462
TCS	462
Friction	462
Brake System	462
Hook	464
Rear Steering Actuator Disp	464
Steering	464
Engine	465
GearBox	465
Clutch	466
Differential Central	466
Differential Front	466
Differential Rear	467
Generic Engine to Clutch	467
Generic Engine to Gearbox	468
Generic Engine to Central Differential	468
Generic Engine to Front Differential	469
Generic Engine to Rear Differential	469
Generic Engine to Front Right Wheel	470
Generic Engine to Front Left Wheel	471
Generic Engine to Rear Right Wheel	471
Generic Engine to Rear Left Wheel	472
Antirollbar	472
Aerodynamic	473
Irregularity	473
Main Damper	474
Main Spring	474
Main Bumpstop	475
Main Reboundstop	475
Auxiliary Vertical	476
Main Inerter	476
Third Element	477
Mono Element	477
Testrig Loadcase	479
Tire Force	481
Tire Moment	482
Wheel Driving	482
Ext Body	483
Ext Ground	483
Ext Tracking	483
External Suspension	484
External Suspension Main Elements User Input	490
External Suspension Scaling Factors	490
Road External	491

Contents

Steering Assist	491
Output Channels	492
Aerodynamic	493
Animator Widget	495
Body Part Acceleration	495
Body Part Displacement	495
Body Part Velocity	496
Brake	496
Bumpstop	497
Reboundstop	498
Cab Bushing/Beam	499
Cab	500
Chassis Acceleration	501
Chassis Displacement	502
Chassis Velocity	502
Chassis Compliance	503
Clutch	503
Compliance Extra Outputs	504
Condition Sensor	504
Cross Weight	505
Damper	505
Differential Speeds	506
Differential Torques	506
Differential Central	507
Differential Front	507
Differential Rear	507
Differential User Output	508
Driver Demand	510
Driver Monitor	511
Engine	512
Motor	512
Motor to Central Differential	513
Motor to Clutch	513
Motor to Front Differential	514
Motor to Rear Differential	515
Motor to Gearbox	515
Motor to Front Left Wheel	516
Motor to Front Right Wheel	516
Motor to Rear Left Wheel	517
Motor to Rear Right Wheel	518
Fuel Consumption	518
Gyro	519
Global Pad	519
Global Upright	520
Hook	520
Inerter	521
Kingpin Contributes	521
Spring	522
Rear Steering System	524
Ride Height	524
Road	525
Sensor	525
Seven Postrig	526
Skidplate	527
Steering	527
Suspension	530
Suspension Testrig	531
System Threshold	531

Contents

System Mass	532
Tire	532
Torque Converter	537
Transmission	537
Upright Accelerometer	538
Vehicle	538
Vehicle States	540
Wheel	542
Wheel Angles	544
Graphics Part and Force	544
Matlab	546
Matlab/Simulink Interface	546
Introduction	547
Setup	547
Interface overview	550
VI-CarRealTime Solver Plugin Export from Simulink	553
Simulink Model Configuration	554
Simulink Model I/O (Obsolete)	555
Simulink Model Parameters	557
MaxPerformance Simulink Interface	557
PressManeuvers Simulink Interface	558
Matlab API Toolkit	558
VI-CarRealTime Utilities	559
Modeling Utilities	559
Example	571
Database Utilities	573
Example	575
Analysis Utilities	576
Example	576
API Examples	578
VI-SuspensionGen Utilities	580
Modeling Utilities	581
Database Utilities	584
Analysis Utilities	584
ResReader Tool	584
Functional Mock-up Interface (FMI)	586
FMI Master Subsystem	586
FMI Master Configuration File	588
VIGRADE_HEADER Block	588
UNITS Block	588
FMI_MASTER_PARAMETERS Block	589
FMU Block	589
Customization	591
Custom Events	591
Setting Up The Environment	592
Event Class Methods	592
VI-Driver File Prototypes	595
Event GUI	596
Event Layout	596
Custom Register	597
Event Editor	597
Editor Testing	598
Custom Code	600
Interfacing Custom Tire Model	605
Interfacing Custom Aero Forces	606
VI-CarRealTime Solver API	606
External Models	607
About External Brake Model	607

Contents

About External Powertrain Model	608
About External Suspension Model	609
4 Tutorials	610
VI-CarRealTime Interface	612
Getting Started	612
Exporting Vehicle From Adams Car	620
Running Analyses	627
Using MaxPerformance	638
Using MF-Tyre/MF-Swift tires	645
Fuel Consumption	652
Model Set-up	653
Run Analysis	656
Review Results	658
Using Vehicle Wizard	660
Using K&C Wizard	662
Running a Vehicle-Trailer analysis	665
Using CarMaker converter	668
Simulating a DRS System	670
Model Set-up	671
Run MaxPerformance Analyses	672
Review Results	674
Running Investigation	676
Run MaxPerformance with Base Model	676
Set up Ramp Steer event	679
Set up the Investigation	680
Run the Investigation	684
Analyze Investigation Results	685
Run MaxPerformance with Optimized Model	687
Compare the Results	688
Using 7Post Rig	689
Model Set-up	690
Run Events	691
Manage SPC File	694
Run 7Post Rig Event	695
Review Results	697
MATLAB/Simulink	697
Co-Simulation	698
Implementing ABS in Simulink	699
Replacing Standard Components	710
Replacing Driveline Components	718
Front/Rear Differential	719
Central Differential	720
Gearbox	721
Clutch	722
Engine	722
Assessing Vehicle Active Systems	723
Skyhook Control System	734
Using MaxPerformance in Simulink	743
Create Event in VI-CarRealTime	743
Instrument Simulink Model	744
Create and Customize Callbacks	745
Run MaxPerformance	745
Using PressManeuvers in Simulink	746
Create Event in VI-CarRealTime	746
Instrument Simulink Model	748
Create and Customize Callbacks	749
Run PressManeuvers	749
Using External Road	750

Contents

Implementing an Electric Vehicle	754
Introduction of Simulink model	754
Starting from kml path	757
Creating send file	759
Running analysis on Simulink	761
Implementing an Hybrid Vehicle	764
Introduction of Simulink model	764
Starting from kml path	767
Creating send file	769
Running analysis on Simulink	772
Connecting External Steering Model	774
VI-CarRealTime Solver Plugin Export from Simulink	778
Model Configuration	779
Model I/O (Obsolete)	781
Model Parameters	783
Model Build/Export	784
Run Investigation with a Solver Plugin	785
Build/Export Simulink Model	785
Add Auxiliary Subsystem in VI-CarRealTime	786
Set Up the Event	788
Set Up the Investigation	789
Run the Investigation	793
Review Results	793
API Toolkit	795
Manage a VI-CarRealTime Model	796
Manage a VI-SuspensionGen Model	801
Suspension Optimization	805
Run Event in VI-CarRealTime	806
Fit Toe Curves	808
Part 1	809
Utility Function	813
Part 2	814
Part 3	816
Using FMI Master	817
FMI: External Driveline	818
FMI: External Suspension	825
VI-CarRealTime Customization	834
Creating Solver Plugin	834
Building Custom Library	835
Running Event With Custom Solver (Standalone)	835
Running Event with Custom Solver (Matlab)	839
Creating Custom Tire Library	843
Building Custom Tire Library	843
Running Event With Custom Tire Model	845
Creating Custom Aerodynamics Force Library	846
Building Custom Aerodynamics Force Library	846
Running Event With Custom Aero Forces Model	847
Creating Custom Event	848
Build Up Environment	848
Add New Event	849
Run Custom Event	859
Review Results	861
Part 2 VI-CarRealTime Adams Interface	863
1 Suspension Analysis Setup	864
Suspension Kinematics	865
Suspension Compliance	867
Auxiliary Roll Stiffness	869

Contents

Dynamic X-DOF Identification Loadcase	870
Steering Kinematics	871
Solver	872
2 Create Suspension Characteristic Files	872
Parameters	873
Sequence of Analyses	875
3 Create Vehicle Model	880
Parameter Extraction	882
Creating Multiple Body Sensors	886
4 Create Static Loadcase File	888
5 Load VI-Driver plugin	889
6 Run Automatic Model Validation	889
Export	891
Components	892
Suspension	892
Longitudinal Dynamics	893
Lateral Dynamics	894
Combined Dynamics	895
Vertical Dynamics	895
Part 3 VI-CarRealTime HIL Overlays	897
1 VI-CarRealTime SCALEXIO Overlay	897
Introduction	897
Installation	897
Licensing	899
Virtual File System	900
Limitations	902
Tutorial	902
ABS Control	902
2 VI-CarRealTime NI-PXI Overlay	919
Introduction	919
Installation	920
Licensing	921
Limitations	923
Tutorials	923
ABS Control	924
Setting up the working directory	924
Generating the maneuver file	925
Uploading to the target	929
Setting up the Simulink Model	930
Building the application	935
Monitoring the application with VeriStand	936
FMI with the NI-Veristand Interface	940
Setting up the working directory	940
Running the FMI experiment locally	942
Preparing the import of the NI-Veristand Interface	943
Uploading to the target	944
Running the FMI experiment on the target with Veristand	945
3 VI-CarRealTime Simulink Real-Time Overlay	949
Introduction	949
Installation	949
Limitations	951
Getting Started	951
Set Up Simulink Model	951
Build Simulink Real-Time Application	954

Contents

Tutorial	955
ABS Control	955
4 VI-CarRealTime ETAS Overlay	964
Introduction	964
Installation	964
Licensing	966
Limitations	967
Tutorial	967
ABS Control	967
Setting up the working directory	967
Generating the maneuver file	970
Creating the LABCAR-OPERATOR project	974
Running the experiment	979
Configuring the OnPreBuild event	981
Part 4 Appendix A: VI-Tire	983
1 Overview	983
2 Tire Axis System	983
3 Tire Slip Definition	985
4 Effective Radius	985
5 Selecting The Tire Model	986
6 Tire Models Dispatching	986
7 Internal models	987
PAC2002	987
MF_05	987
PACMC	987
Legacy Tire Models	987
8 Cosin FTire interface	988
9 Siemens MF-Tyre/MF-Swift interface	988
10 Michelin Tametire interface	990
11 Fraunhofer CDTire	990
12 User Defined Tire Model	990
Tire User Interface	990
Tire Utility Functions	994
Tire-Road Interface	996
Tire-Road General Interface	998
Tire Output Map	1000
User Tire Data Save And Restore	1003
Testing procedure for Save/Restore with VI-CarRealTime	1004
13 Tire/Road Compatibility	1005
14 Bibliography	1005
Part 5 Appendix B: File Format	1006
1 AER	1006
AER - Standard Format	1006
AER - Advanced Format v1.0	1008
AER - Advanced Format v2.0	1013
Converting from v1.0 to v2.0	1022
2 CFG	1031
3 DRD	1032
4 PMF	1033

5 PWR	1034
6 REP	1040
7 SPC	1042
Example 1	1045
Example 2	1046
Example 3	1046
Example 5	1047
8 VTT	1048
VIGRADE_HEADER Block	1048
UNITS_Block	1048
MODEL Block	1049
SYSTEM Block	1050
ANALYSIS Block	1051
ANALYSIS Block Common	1051
ANALYSIS Block Longitudinal Slip	1053
ANALYSIS Block Lateral Slip	1053
ANALYSIS Block Combined	1053
ANALYSIS Block CornStiff	1053
ANALYSIS Block Free Input	1053
ANALYSIS Block Six Dof	1054
ANALYSIS Block Z Locked	1055
FUNCTION	1055
STEP	1056
SINE	1056
RAMP	1057
USER	1057
VTT Examples	1058
Longitudinal Slip	1059
Lateral Slip	1060
Combined	1061
Cornering Stiffness	1062
Free Input	1063
Six Dof	1065
Z Locked	1066
9 RDF	1067
Common Blocks	1067
MDI_HEADER Block	1067
UNITS Block	1068
MODEL Block	1068
GLOBAL Block	1070
WIDTH Block	1072
WIDTH_TABLE Block	1072
FRICTION Block	1072
Optional Blocks	1074
PLANT_PATH Block	1074
VERTICAL_PATH Block	1075
BANK_ANGLE_PATH Block	1075
MEASURED_CENTERLINE Block	1075
IRREGULARITIES Block	1076
Theory	1078
Sample	1079
LEFT_PSD_TABLE Block	1079
RIGHT_PSD_TABLE Block	1080
PSD_COHERENCY_TABLE Block	1080
PSD_PHASE_TABLE Block	1081
MEASURED_IRREGULARITIES	1081

Contents

LEFT_KERB_GEOMETRY Block	1081
RIGHT_KERB_GEOMETRY Block	1082
FRICITION_TABLE Block	1082
CENTERLINE_PROPERTIES Block	1083
SECTION_PROPERTIES Block	1084
SECTION Block	1085
NODES Block	1086
ELEMENTS Block	1087
MATERIALS Block	1088
TRACK_PATH Block	1088
ROAD_DATA block	1089
OPEN_CRG block	1089
RDF Examples	1089
Analytic Sample	1090
Measured Sample	1091
Extrusion Sample	1092
Mesh Sample	1095
Flat Sample	1095
External Sample	1096
Composite Sample	1097
OpenCRG Sample	1098
Discontinued ARM Road Model	1098
Deprecated SURFMESH Road Model	1099
10 SDF	1099
MODEL Block	1099
CONTROLLER Block	1100
VEHICLE Block	1101
General parameters:	1101
Vehicle mode parameters:	1102
Motorcycle mode parameters:	1103
MANEUVER Block	1103
Maneuver sub block	1103
MACHINE sub block	1104
PATH_CONTROL	1104
STRAIGHT	1104
SKIDPAD	1104
PATH	1105
LATERAL_ACC	1105
SPEED_CONTROL	1105
MAINTAIN	1105
MAP	1105
ACCMAP, ACCMAP2	1105
LATACCMAP	1106
END_CONDITION Sub Block	1106
PATH Block	1106
DDC_FILE	1107
DRD_FILE	1107
MEASURED	1107
MAP Block	1107
11 VDF	1108
VIGRADE_HEADER Block	1108
UNITS Block	1108
DEBUG_PARAMETERS Block	1109
GLOBAL_SETTINGS Block	1109
CONTROLLER_STANDARD Block	1110
STEERING_STANDARD Block	1111
THROTTLE_STANDARD Block	1112
BRAKING_STANDARD Block	1112

Contents

GEAR_STANDARD Block	1112
GEAR_EDS Block	1114
GEAR_SCHEDULED_CONTROLLER Block	1116
HUMAN_CONTROLLER Block	1119
CLUTCH_STANDARD Block	1120
TIRE_BASIC_CONTROLLER Block	1121
TIRE_LONSLIP_STANDARD Block	1122
STARTUP Block	1123
MANEUVER_LIST Block	1124
MANEUVER Block	1124
Control Channels	1125
MACHINE	1126
TIRE	1126
OPTIONS	1126
GEAR_SCHEDULED	1127
HUMAN	1127
END_CONDITION	1128
MAP Block	1128
DRD MAP Block	1129
PATH Block	1130
DRD Type	1130
NUMERIC_DATA Type	1131
SHAPE STRAIGHT Type	1131
SHAPE SKIDPAD Type	1131
END_CONDITION Block	1132
OPEN_SIGNALS Block	1134
STEP Block	1135
SINE Block	1135
COSINE Block	1136
SWEEP Block	1136
CONSTANT Block	1137
RAMP Block	1137
IMPULSE Block	1138
DCD Block	1138
USER_Block	1139
VDF Example 1	1140
VDF Example 4	1144
12 TIR	1147
Common Blocks	1147
MDI_HEADER block	1147
UNITS block	1148
MODEL block	1149
DIMENSION block	1150
VERTICAL block	1151
LONG_SLIP_RANGE block	1151
SLIP_ANGLE_RANGE block	1152
INCLINATION_ANGLE_RANGE block	1152
VERTICAL_FORCE_RANGE block	1152
Optional Blocks	1153
SCALING_COEFFICIENTS block	1153
LONGITUDINAL_COEFFICIENTS block	1156
LATERAL_COEFFICIENTS block	1158
ALIGNING_COEFFICIENTS block	1162
ROLLING_COEFFICIENTS block	1165
OVERTURNING_COEFFICIENTS block	1166
SECTION_PROFILE_TABLE block	1166
SPEED_RADIUS_LOAD_DATA block	1167
13 UBF File Format	1167

Contents

Elastomer Model	1168
Hydromount Model	1168
Property File	1169
Part 6 Appendix C: Advanced Topics	1175
1 Environment Variables In VI-CarRealTime.....	1175
2 Using FTire At Best In VI-CarRealTime.....	1176
3 Building Custom Logics	1178
Behavior tree	1178
Introduction	1178
Building blocks	1179
Nodes	1179
Sequence	1180
Selector	1180
Parallel	1181
Leaves	1181
Decorators	1181
Python bindings	1182
Introduction	1182
Behavior tree Python module reference	1183
Utils	1187
Examples	1189
Using tirelimits	1189
Pushing lateral performance	1189
Pushing performances	1190
Keep the vehicle in a given GG range	1193
Maintaining accelerations inside a range.	1193
Part 7 Installation Guide	1197
1 VI-CarRealTime 20.0 installation on Windows.....	1197
How To Get The Software	1197
Running the Installer	1198
Standard Installation	1199
Components	1199
Installation Folder	1200
Installing	1200
Network Installation	1202
Installation Folder	1202
Installing	1203
Batch Installation	1204
Uninstalling	1204
2 Setting up the License	1205
3 Troubleshooting	1206
4 License Toolkit	1208
Part 8 Release Notes	1210
1 What's New	1210
VI-CarRealTime 20.0	1210
What's New in VI-Road	1214
2 Licenses	1215
3 3rd Party Compatibility	1217
4 System Requirements	1218
5 Updating models	1219

Contents

Updating to version 20	1219
Updating to version 19	1219
Updating to version 18	1219
6 Changed Behaviour	1219
Version 20	1219
Version 19	1221
Version 18	1221
7 Known Issues	1222
8 Revision History	1223
Release 20.0	1223
Release 19.2	1227
Release 19.1	1227
Release 19.0	1230
Release 18.2	1232
Release 18.1	1233
Release 18.0	1234
Release 17.3	1237
Release 17.2	1237
Release 17.1	1238
Release 17.0	1241
Release 16.2	1243
Release 16.1	1244
Release 16.0	1244

1 VI-CarRealTime

1.1 Introduction

VI-CarRealTime is a virtual modeling and simulation environment targeted to a simplified 4 wheels vehicle model. Its functionalities include the ability to assemble the vehicle system by collecting its fundamental subsystems, specifying dynamic maneuver schedules, launching standalone or Matlab-Simulink embedded simulations, post-processing the obtained results.

The environment based on underlying solver consists of:

- Symbolically derived parameterized equations of motion;
- Pacejka tire model;
- Sophisticated, virtual driver model.

Such a complete model+environment (tires, driver) is exportable using C and C++ code generation, for subsequent usage for SIL, HIL or vehicle dynamic simulators on Windows or Linux computer platforms.

An intuitive graphical user-interface supports the model definition, model testing, and results review. The global system architecture is fixed (but modifiable upon request), although the system and subsystem parameters are modifiable, as well as all the property files, which characterize component's behaviour.

All system data could either come from experimental tests performed in a lab, or from a virtual test performed within Adams Car. The user can track over 900 output variables during the simulation, being able to pass in over 100 inputs optionally (by writing user subroutines), and can plot and graphically animate the simulation results.

1.1.1 Software Architecture

VI-CarRealTime software is organized into several components; each of them is used for a specific set of tasks.

- [VI-CarRealTime Environment](#)
- [Adams Car plugin](#)
- [Accessories](#)

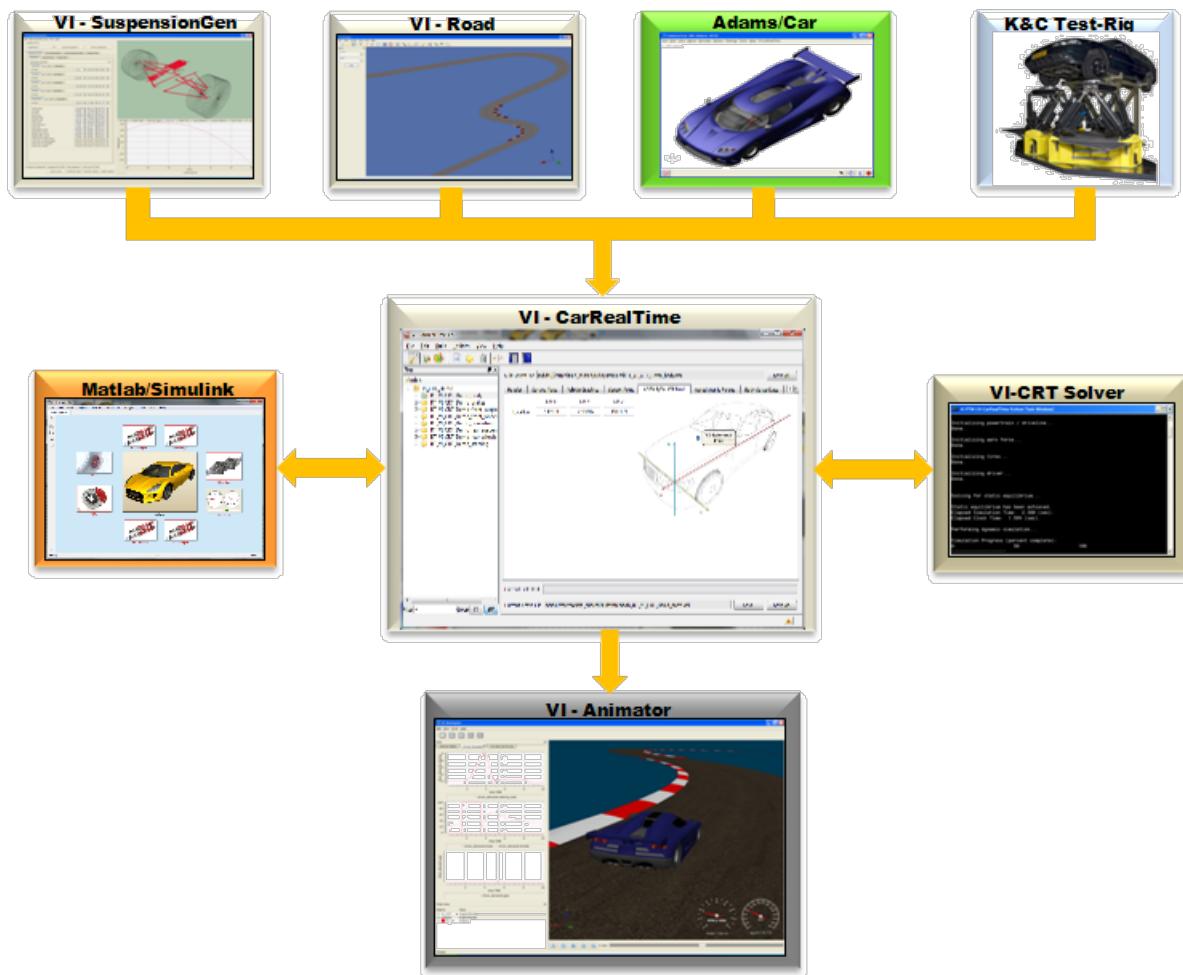
The framework is used to manage models and simulations and to invoke accessories when needed (plots, road editing, post-processing etc.). Vehicle models can be created by the user through direct input of vehicle data (suspension curves, mass and inertia properties, property files etc.) or, when an Adams Car model of the vehicle is available using the Adams Car plugin.

The Adams Car plugin is capable of extracting all vehicle data needed for VI-CarRealTime model and to organize them in a database.

Accessories include:

- **VI-Road**
a tool for generating roads and driver paths.
- **VI-Animator**
post-processing tool for plots and animations (as an alternative Adams/PPT can be used).
- **VI-SuspensionGen**
suspension curves generation utility.
- **VI-TireLimits**
a tool for evaluating the force envelope given an input tire property file.

The image below shows a flow chart of VI-CarRealTime process flow:



The vehicle model can be generated using Adams Car plugin, VI-SuspensionGen or directly by the user; the VI-CarRealTime framework is used to visualize and edit model data and it communicates with solver and

VI-CarRealTime

MATLAB/Simulink through a socket or files. Using the framework it is also possible to start accessory applications for road managing and post-processing.

1.2 VI-CarRealTime Environment

VI-CarRealTime environment is divided into four work modes:



- [Build](#)

The Build mode allows you to edit model data and change system configuration. You can also work on multiple models at once. The Build mode is the default for starting VI-CarRealTime.



- [Test](#)

The Test mode allows you to run your model.



- [Review](#)

The Review mode allows you to visualize analysis results using either VI-Animator or Adams/PostProcessor. You can postprocess the output of standard VI-CarRealTime events. Postprocessing has two formats: animation and plots. A majority of standard VI-CarRealTime events have either an animation, a plot, or both.



- [Investigation Mode](#)

The Investigation mode allows you to run multiple simulations based on one or more events, while modifying a set of model parameters. You can define measurements as well to check on each simulation.

The VI-CarRealTime window has a consistent design in each mode. These elements include:

- Menubar
- Toolbar
- Treeview
- Property Editor

The toolbars in VI-CarRealTime change according to the work mode. Below is the basic toolbar that is available in all work modes.

The following figures show the toolbars that are available in each work mode:

- **VI-CarRealTime Basic Toolbar:**



- **VI-CarRealTime Build Mode Toolbar:**



- **VI-CarRealTime Test Mode Toolbar:**



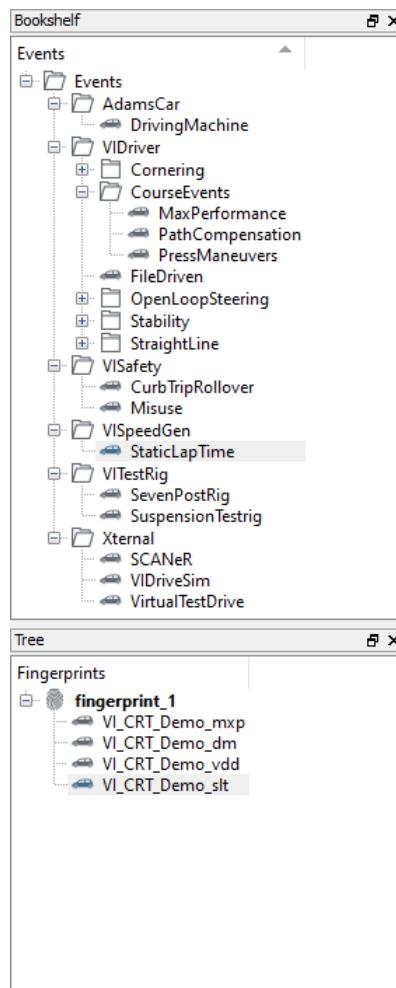
- VI-CarRealTime Review Mode Toolbar:



- VI-CarRealTime Investigation Mode Toolbar:



The VI-CarRealTime treeview changes according to the work mode. The treeview has different features in each work mode. Below are the two basic sections of the treeview:



1.2.1 Window Elements

The VI-CarRealTime window includes the following elements:

- **Menu bar**

contains pull down menus for File, Edit, Build, Test, Review, Utilities, View and Help. Menu content changes depending on the work mode and installed accessories.

- **Toolbar**

allows the user to change work modes and it contains commonly used tools to work with the models. The tools in the toolbar are arranged in the order used during the process of creating and testing the model. Depending on the work mode VI-CarRealTime enables or disables them or adds additional tools.

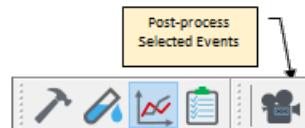
VI-CarRealTime

• Treeview

is a hierarchical listing of objects in the current session. It appears along the left side of the VI-CarRealTime window. The tree is especially useful when creating/modifying a model or an event.

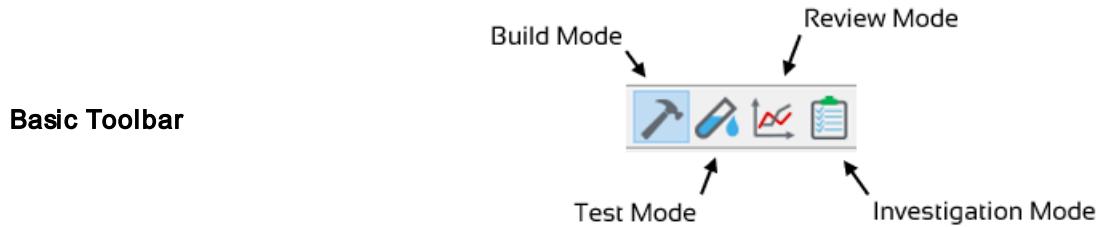
• Property Editor

is the area of the window that displays field for modifying the object selected from the Treeview.

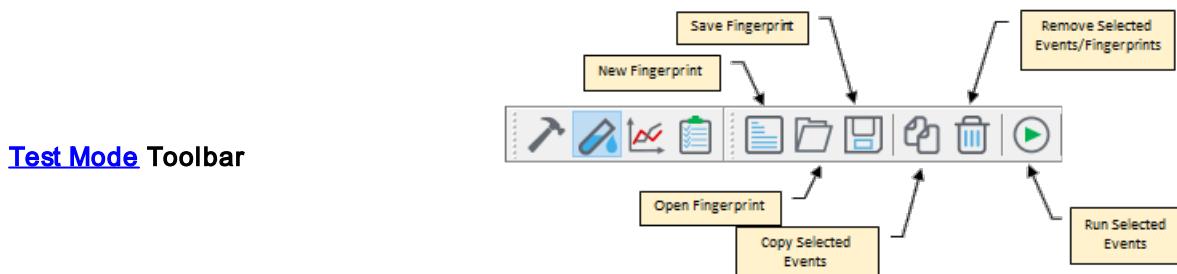
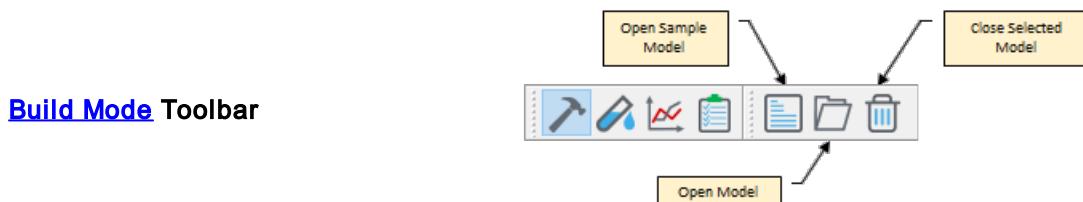
**Toolbars**

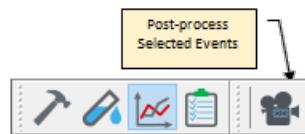
The toolbars in VI-CarRealTime change depending on the work mode.

The **Basic Toolbar** is active in all work modes and is suited for mode selection.

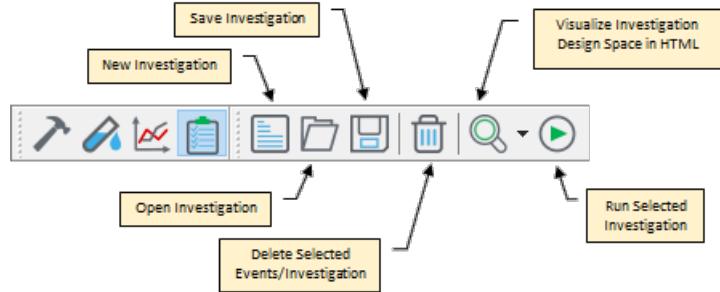


Accordingly to the selected work mode (Build, Test or Review) the toolbar contains specific button functionalities. The images below show the aim of each button in a specific context.



Review Mode Toolbar

Post-process Selected Events

Investigation Mode Toolbar

Delete Selected Events/Investigation

Open Investigation

Visualize Investigation Design Space in HTML

Run Selected Investigation

Save Investigation

New Investigation

Treeview

The VI-CarRealTime treeview changes according to the work mode. The treeview has different features in each work mode.

[Learn how to use the treeview in each work mode.](#)

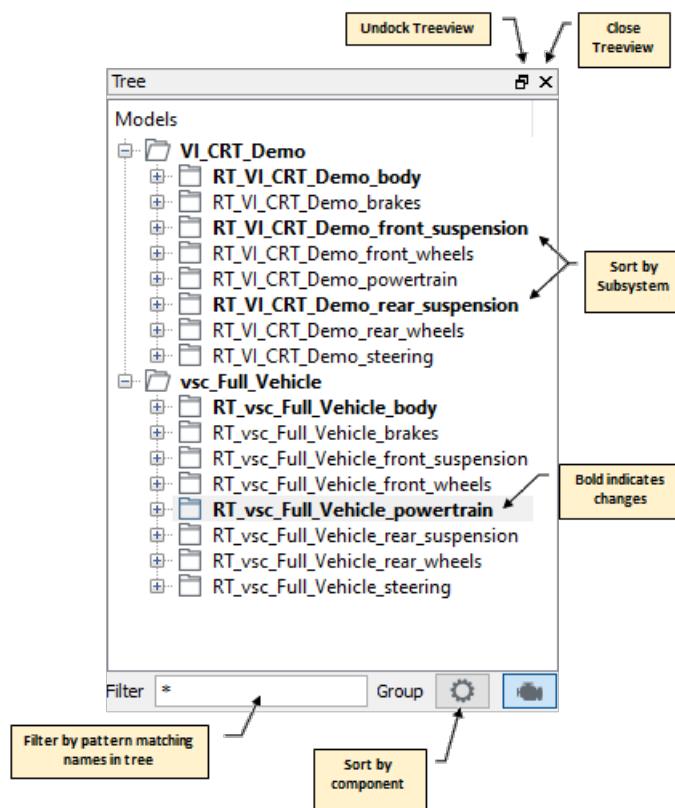
Using

Using the Treeview

Select a work mode to learn how to use its treeview:

- [Build](#)
- [Test](#)
- [Review](#)
- [Investigation](#)

Build Mode:



To use the Build mode treeview:

- Select a folder to display all the attributes (subsystems, components) in the property editor.
- Expand a folder to select a specific attribute to be displayed in the property editor.
- Use the text box to filter the treeview.
- Use the sorting tools to sort the treeview.

Test Mode:

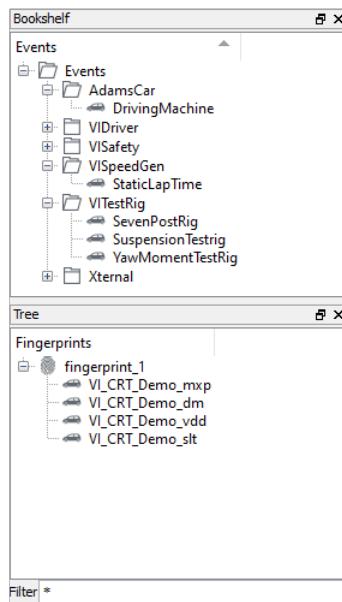
The treeview in Test mode is divided into two sections:

- **Event bookshelf**

organizes all available events in a tree structure. By double clicking on an event class it creates a new event in current fingerprint. Right mouse button opens popup men for creating the event in an existing or new fingerprint and to use a specific model loaded in session.

- **Fingerprint tree:**

- displays the fingerprints and all events in fingerprints
- allows unlimited number of events in a fingerprint
- allows to manipulate multiple fingerprints
- display changed data in treeview with bold text



To use the Test mode treeview:

- Double-click an event in the event bookshelf to add an event to a new or existing fingerprint.
- From the fingerprint tree, select the fingerprint to view the first four input parameters and attributes for every event from the fingerprint in a table.
- From the fingerprint tree, select an event you want to edit.

Review Mode:

The Review mode treeview is divided into two sections:

- Postprocessing bookshelf
- Fingerprint tree

To use Review mode treeview:

- To create a new custom postprocessor, double-click an event class in the postprocessing bookshelf.
- Double-click a fingerprint to postprocess.

Investigation Mode:

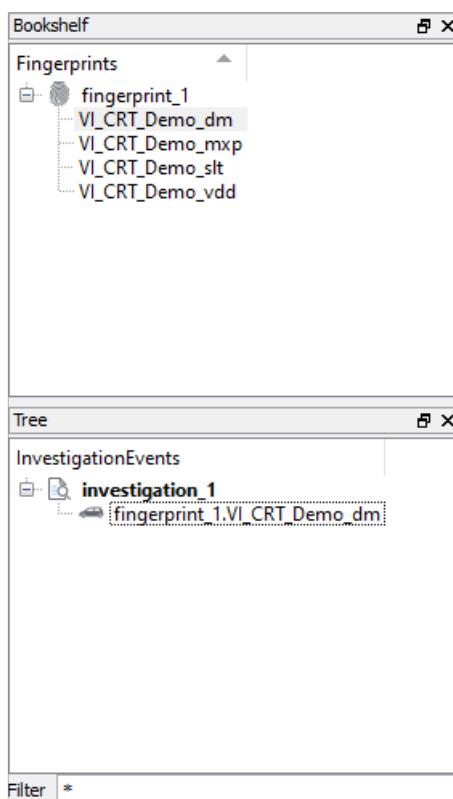
The treeview in Investigation mode is divided into two sections:

- **Fingerprint bookshelf**

organizes all available events in a tree structure. Right-click an event in the event bookshelf to add an event to a new or existing Investigation.

- **Investigation Events Tree:**

displays the investigations and the events. Click an event in the investigation tree bookshelf to edit an investigation.



Filtering

Filtering the Treeview

You can filter the treeview in all work modes. The filter in the VI-CarRealTime treeview performs some basic pattern matching using a UNIX shell-style wildcard string.

To filter the treeview:

- Enter the character pattern you want to match in the Filter field.

Wildcards

* - matches an arbitrary sequence of characters.

? - matches one character.

[chars] - matches any of chars.

Examples:

Matches anything starting with f and ending with d such as:

f*
fd
frd
fled
fried
flyed

Matches anything starting with f and having d as the third character such as:

f?d
fad
fdb

f1d
f2d

Matches anything starting with f followed by an x, y, or z followed by d, such as:

f[xyz]d fxd
 fyd
 fzd

Sorting

Note that you can only sort data in the **Build Mode**.

The Build Mode treeview has two different hierarchical data representations available:

- **Sort by Subsystem**
Arranges the data in the tree in the following way: Model -> **Subsystems** -> Components.
- **Sort by Component**
Arranges the data in the tree in the following way: Model -> **Components** -> Subsystems.

Both sorting methods give access to the same data.

To sort the treeview:

- Select the "Sort by subsystem"  button to view the subsystems.
- Select the "Sort tree by component type"  button to view all the components in the subsystem.

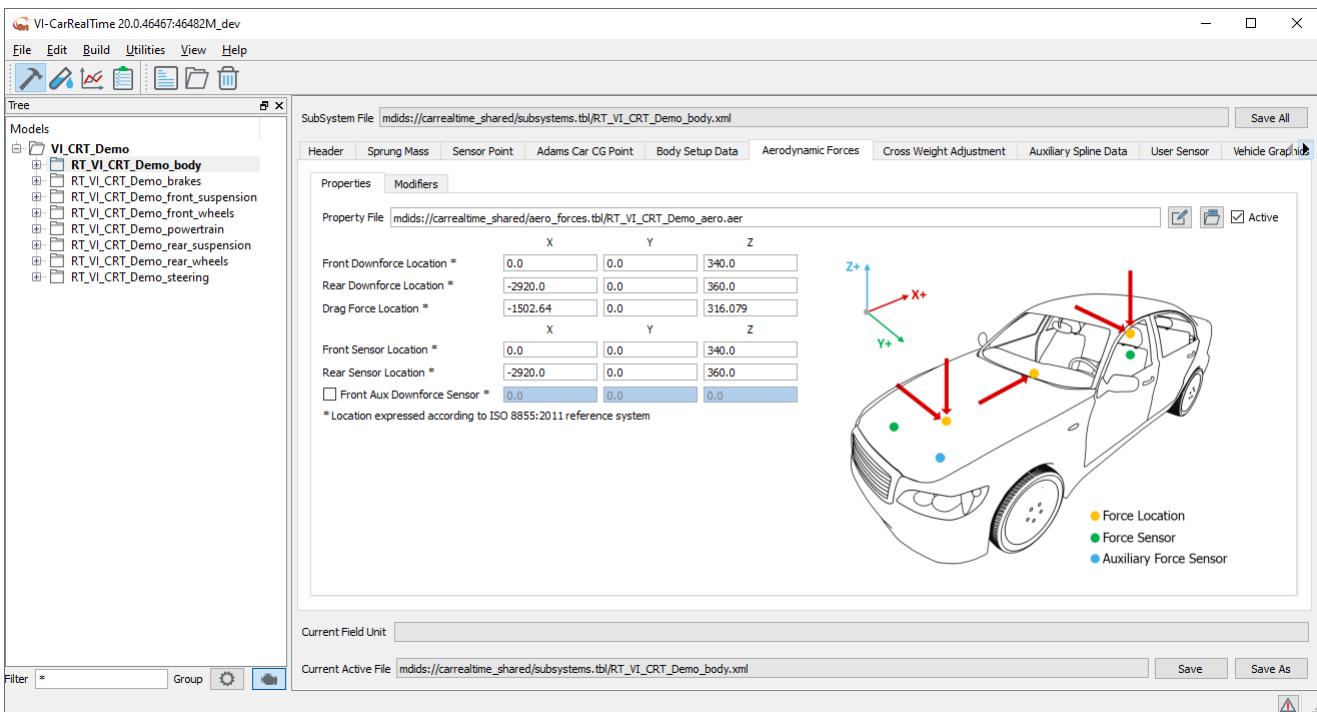
Property Editor

The property editor in VI-CarRealTime changes its appearance based on the current work mode and your selection in the tree view. It allows you to modify objects you selected in the tree view.

The next sections explain the property editor in each mode:

- [Build](#)
- [Test](#)
- [Review](#)
- [Investigation](#)

VI-CarRealTime

**Build Mode:**

The property editor in Build mode:

- Displays the subsystem file you are working on.
- Shows what component you are editing in a tab.
- Displays the file name of the component when you place your cursor over it.
- Allows you to manipulate data in editable table cells.
- Allows you to add new elements to your model.
- Allows you to save the modifications to the model or save it as a new file.

Test Mode:

The property editor in Test mode allows you to:

- Select a single event to edit.
- View current field units feedback.
- Select optional road data file.
- View input parameters for any given number of events from the fingerprint tree.
- Select a fingerprint to view the first four input parameters and attributes for every event from the fingerprint in a table. This table allows you to quickly view and modify the parameter settings prior to building and running your simulations.
- View input parameters and attributes for the events you selected in the table.

Review Mode:

The property editor in Review mode:

- Is a dynamically created table based on your selection in the treeview.
- Creates a table containing an entry for each selected event.
- Each event will have a checkbox for enabling animations.

Investigation Mode:

The property editor in Investigation mode allows you to:

- Visualize the parameters available in model. They are shown as candidates and can be promoted to factors.

- Select factors and modify investigation variations.
- Create and modify responses to evaluate investigation results.
- Modify output folder and select different output maps.

Note: The toggle settings you have selected will be retained when you switch modes. They are saved and retained when you exit your VI-CarRealTime session. When you re-open VI-CarRealTime, the postprocessor toggles will be set to the setting you saved for a given session file.

1.2.2 Edit Menu

The Edit menu in the main VI-CarRealTime menu bar defines the following options:

- [Preferences](#)
- [GUI Settings](#)

Preferences

The preferences are used to define the work environment specific to the machine you are using. In particular, the preferences are XML files that define the *local* settings of the VI-CarRealTime session.

Before running VI-CarRealTime, you must either set up the preferences or load an existing preferences file.

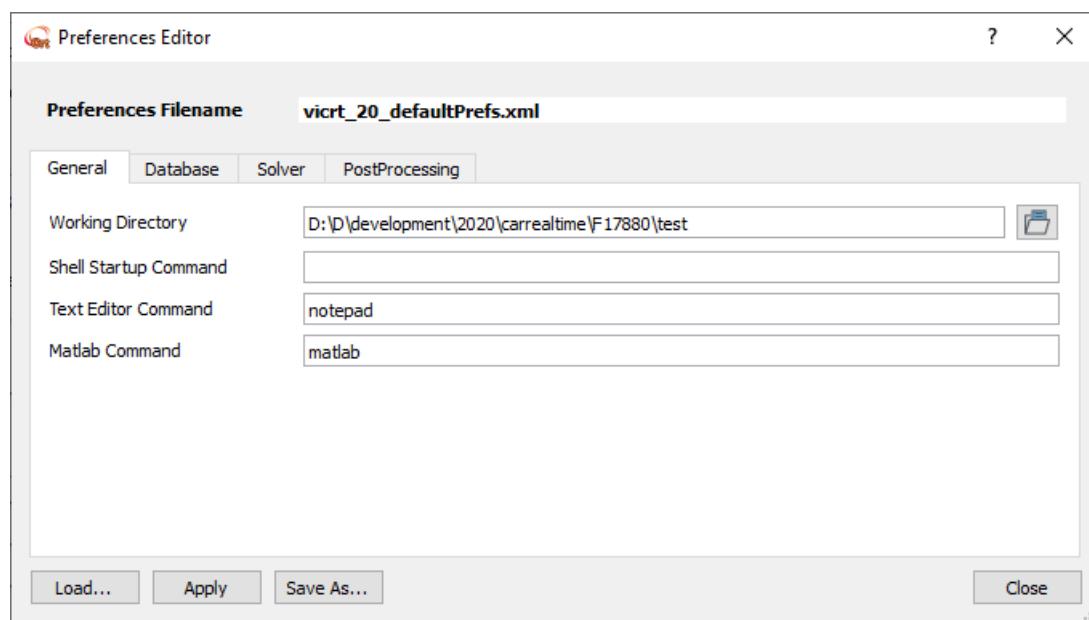
Note: incorrect preference settings can prevent VI-CarRealTime from starting up.

When opening VI-CarRealTime GUI, a file named **vicrt_20_defaultPrefs.xml** is searched for in the *working directory* and machine *HOME* directory.

If the preferences xml file is not found it is created using default values.

To set preferences, from the **Edit** menu, select **Preferences**.

The Preferences window appears:



The Preferences Editor is composed by 4 panels:

- [General](#)
- [Database](#)
- [Solver](#)

- [PostProcessing](#)

General

The General panel defines the following fields:

- **Working Directory**

defines the working directory where all VI-CarRealTime output files will be saved. The user is allowed to modify the directory according to his needs.

- **Shell Startup Command**

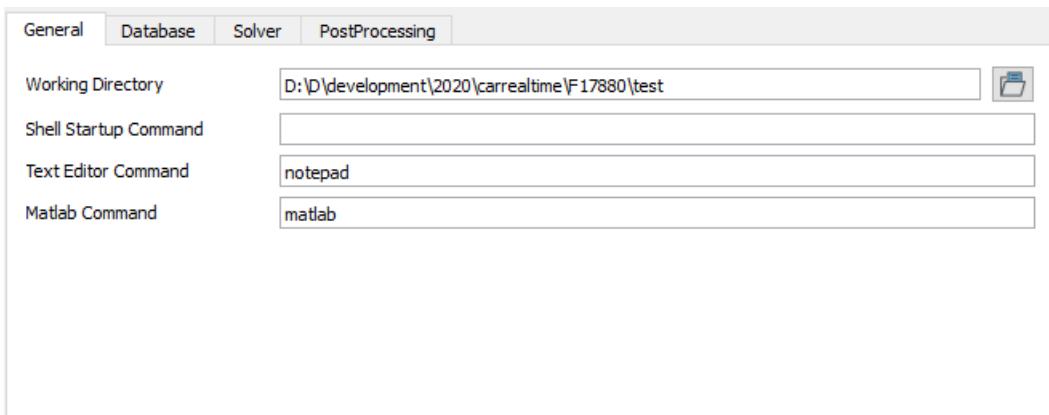
it allows the user to specify the full path name of the executable to be used by VI-CarRealTime to open shell command (i.e. cmd.exe for Windows).

- **Text Editor Command**

it allows the user to specify the full path name of the executable to be used by VI-CarRealTime to open text files.

- **Matlab Command**

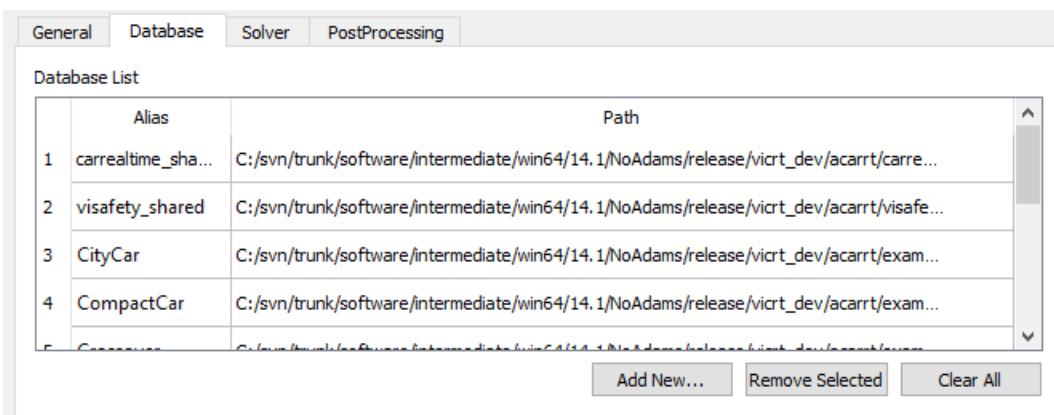
it allows the user to specify the full path name of the MATLAB version to be used in VI-CarRealTime.



Database

The database panel lists the databases registered in session. The user has the possibility to add or remove databases according to his needs.

Please refer to [Registering Databases](#) topic for all the details.



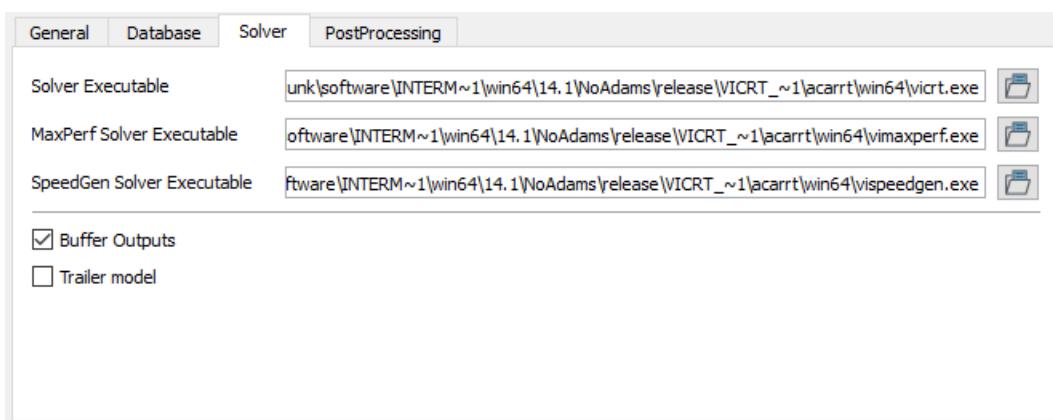
Solver

The Solver panel references the paths of the solver executable used by VI-CarRealTime:

- **Solver Executable**
it defines the full path of the solver executable to run the [events](#).
- **MaxPerf Solver Executable**
it defines the full path of the solver executable to run the [MaxPerformance](#) event.
- **SpeedGen Solver Executable**
it defines the full path of the solver executable to run the [StaticLapTime](#) event.

Moreover it allows the user to set the following options:

- **Buffer Outputs**
when the flag is active, the simulation result files in the working directory will be written after the simulation has been completed.
- **Trailer model**
when the flag is active, it enables the trailer functionality in [Test Mode](#).

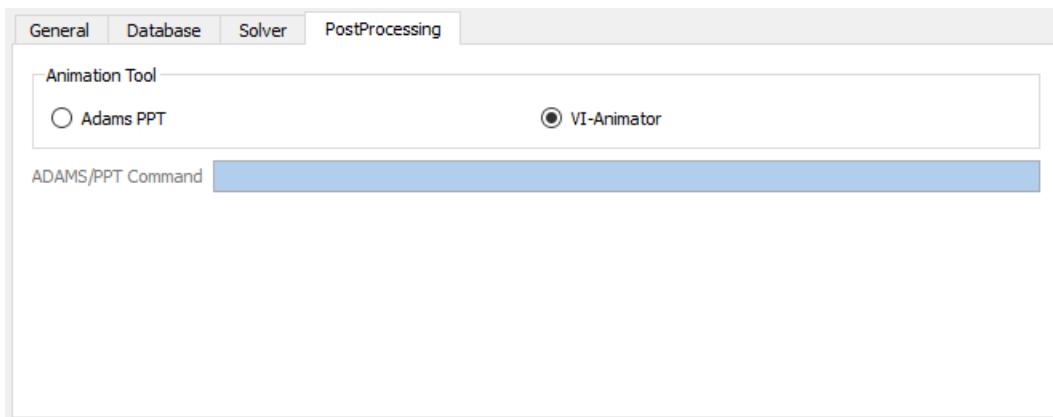


PostProcessing

The PostProcessing panel allows the user to choose the post processing tool.

Available choices are:

- **VI-Animator**
default VI-CarRealTime post-processing tool provided with VI-CarRealTime installer.
- **Adams PPT**
Adams Post Processor.

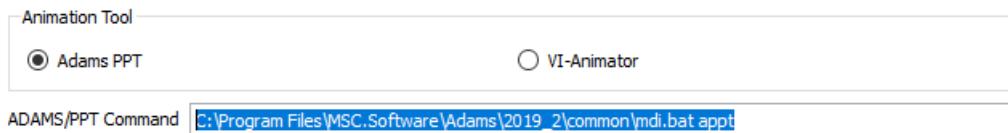


When Adams/PPT radio box is selected, the Adams/PPT Command field is enabled and the user must enter the command for starting the Adams Postprocessor installed in his machine.

For example, the following command allows to start up Adams Postprocessor 2019.2 x64:

C:\MSC\~1.SOF\ADAMS_~1\2019_2\common\mdi.bat appt

As a general rule to start Adams Postprocessor, it is sufficient to call the **mdi.bat** file stored in the **common** folder of Adams installation directory followed by the string **appt**.



Registering Databases

VI-CarRealTime data is organized inside a database structure (that can be shared with Adams Car databases).

A database is a folder structure in which model property files are organized into subfolders, each of them containing a specific property file type (systems, subsystems, components, events, etc.)

Databases need to be registered among VI-CarRealTime preferences in order to have file paths correctly resolved from database **alias**.

How to add a new database

In order to add a new database to VI-CarRealTime session you can open the **Preferences Editor** and do the following:

- Click the **Add New..** button in the database registration area of the editor
- Browse for the directory location of the database
- Click the **OK** button.

The selected database will be added to the current session and VI-CarRealTime will use its Alias to resolve the database and retrieve all the files inside the database folders.

Databases are automatically registered in VI-CarRealTime whenever a model belonging to an unregistered database is loaded into the session. Nevertheless the registration will not be saved into a preference file (it can be done clicking the **Apply** button in the preferences editor) so it will remain valid only during the current session unless preferences are saved.

Note: A database alias is automatically derived by the application from the folder base name of the database (please be sure to respect this rule if you plan to import VI-CarRealTime from Adams Car).

Database List		
	Alias	Path
1	carrealtime_shared	C:/svn/trunk/software/intermediate/win64/14.1/NoAdams/release/vicrt_dev/acarr...
2	visafety_shared	C:/svn/trunk/software/intermediate/win64/14.1/NoAdams/release/vicrt_dev/acarr...
3	CityCar	C:/svn/trunk/software/intermediate/win64/14.1/NoAdams/release/vicrt_dev/acarr...
4	CompactCar	C:/svn/trunk/software/intermediate/win64/14.1/NoAdams/release/vicrt_dev/acarr...

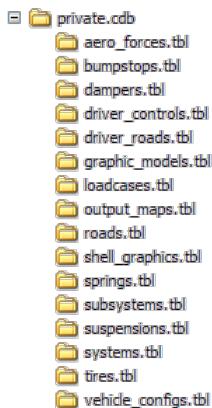
Database Structure

Each database consists of one directory (*.cdb) and several subdirectories (*.tbl), called tables. Each subdirectory contains files for specific types of components such as springs and dampers, or files for performing tests such as loadcases.

VI-CarRealTime divides a database into the following table elements by default:

- Models and topological information (systems, subsystems);
- Analysis information (such as analysis scripts, loadcases);
- Driver files (such as driver inputs and roads);
- Tires and roads;
- Property files (such as springs, dampers, and remaining tables).

An example of the default structure of a database for VI-CarRealTime is shown in the figure below.



Sessions

A session in VI-CarRealTime encompasses the entire vehicle modeling environment. From one interface you can modify vehicle data, build events, test events, and review results. The entire environment, including user-specified settings, can be saved in a session file.

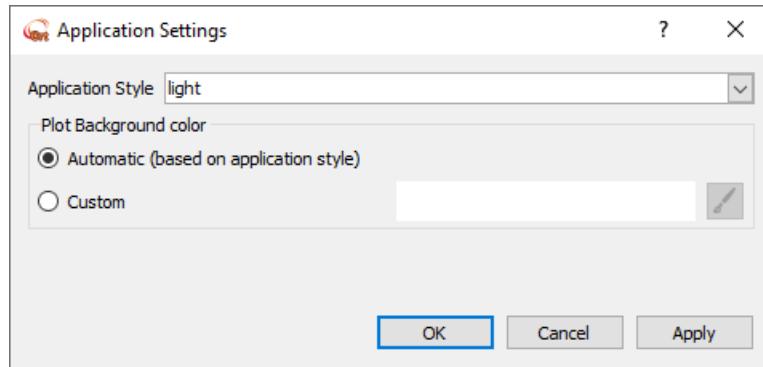
To work with sessions:

From the File menu, select one of the options explained in the list below to indicate how you would like to use VI-CarRealTime.

- **New Session:** Starts a new session of VI-CarRealTime that does not have any vehicles loaded.
 - If you start a new session, VI-CarRealTime
 - prompts you to save your changes.
 - deletes the models, fingerprints, and any simulations after you make your selection.
- **Open Session:** Opens a saved session. VI-CarRealTime
 - prompts you for the saved session file.
 - reads session file.
 - sets the current working directory to the directory specified in the Preferences Panel.
 - adds the Preferences Registered Databases to the session database.
 - loads any custom events, models, fingerprints, and/or review settings.
- **Save Session:** Saves the current session file.
 - **Save Session As:** Saves the current session file to a unique name.
 - **Exit Session:** Closes the session you are working on. VI-CarRealTime
 - prompts you to save any changes you may have made to your current session.
 - updates your preferences with used databases.
 - writes `vicrt_20_defaultPrefs.xml`.
 - writes `vicrt_20_defaultSession.xml`.

GUI Settings

The GUI Settings menu allows the user to set specific visualization options for the main VI-CarRealTime GUI and for the plots in the Curve Editor.



The panel defines the following fields:

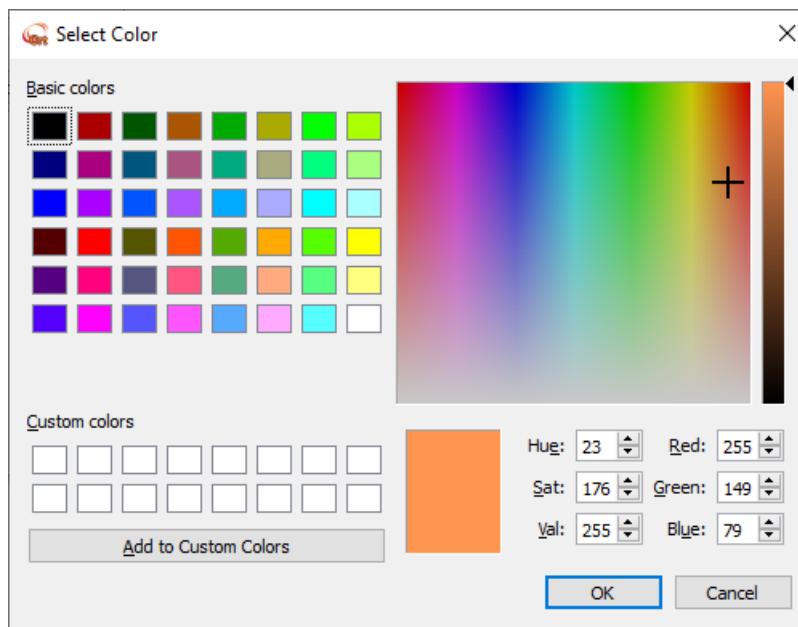
- **Application Style**

select the theme to be used for VI-CarRealTime. Available choices are [light](#) and [dark](#).

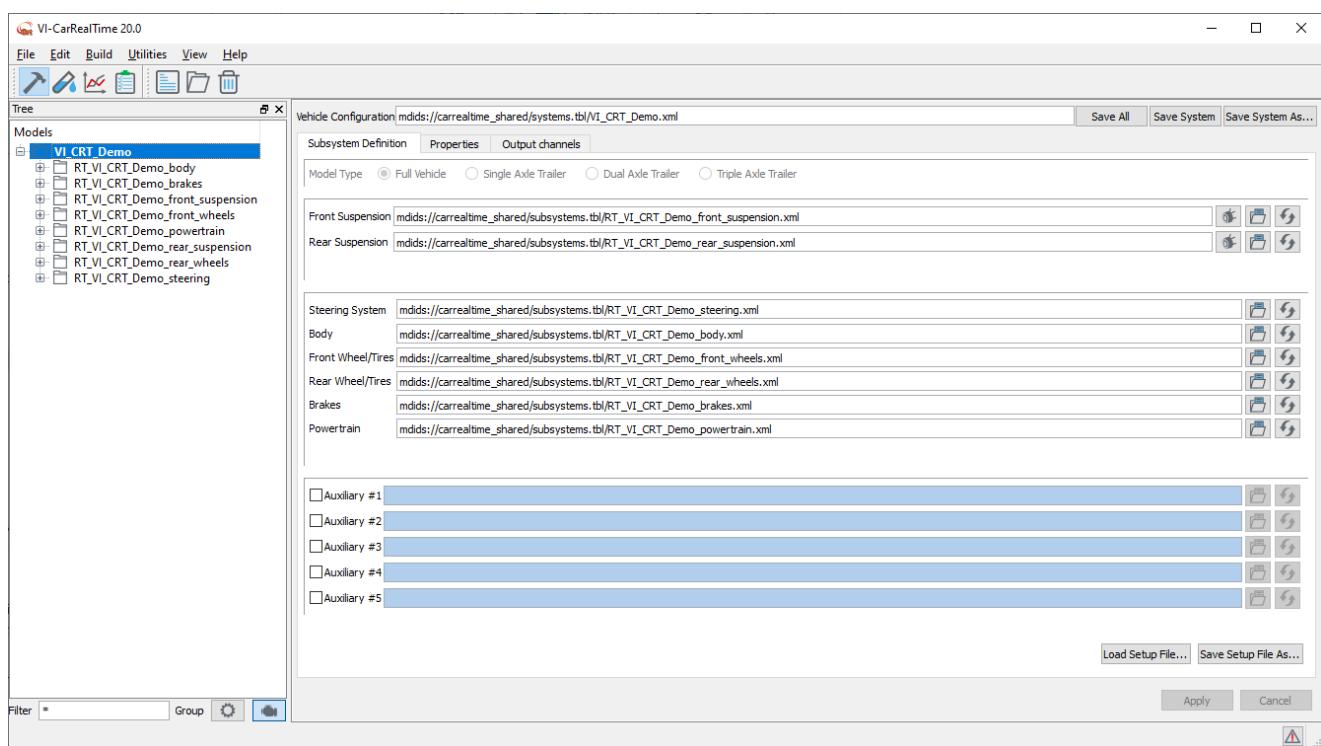
- **Plot Background color**

allows the user to set the background color used in the plots in the Curve Editor tool. Available choices are:

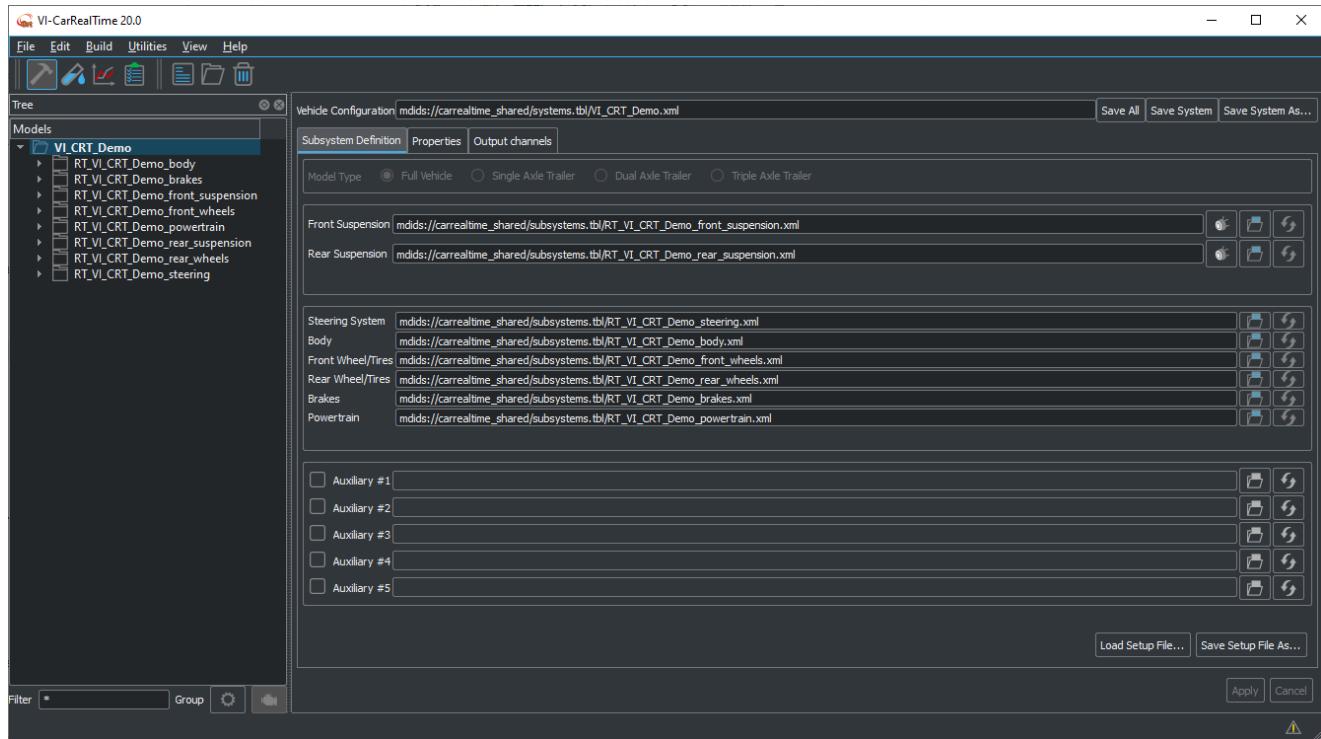
- **Automatic:** the background color is automatically chosen based on selected Application Style
- **Custom:** the user can choose the desired color using the color palette



Light Mode



Dark Mode



1.2.3 Build Mode

VI-CarRealTime Build mode lets you edit model data and change system configuration.

The Build button in menu toolbar allows to:

- Load model:
- Load Example Full Vehicle Model

VI-CarRealTime

- Register example Databases

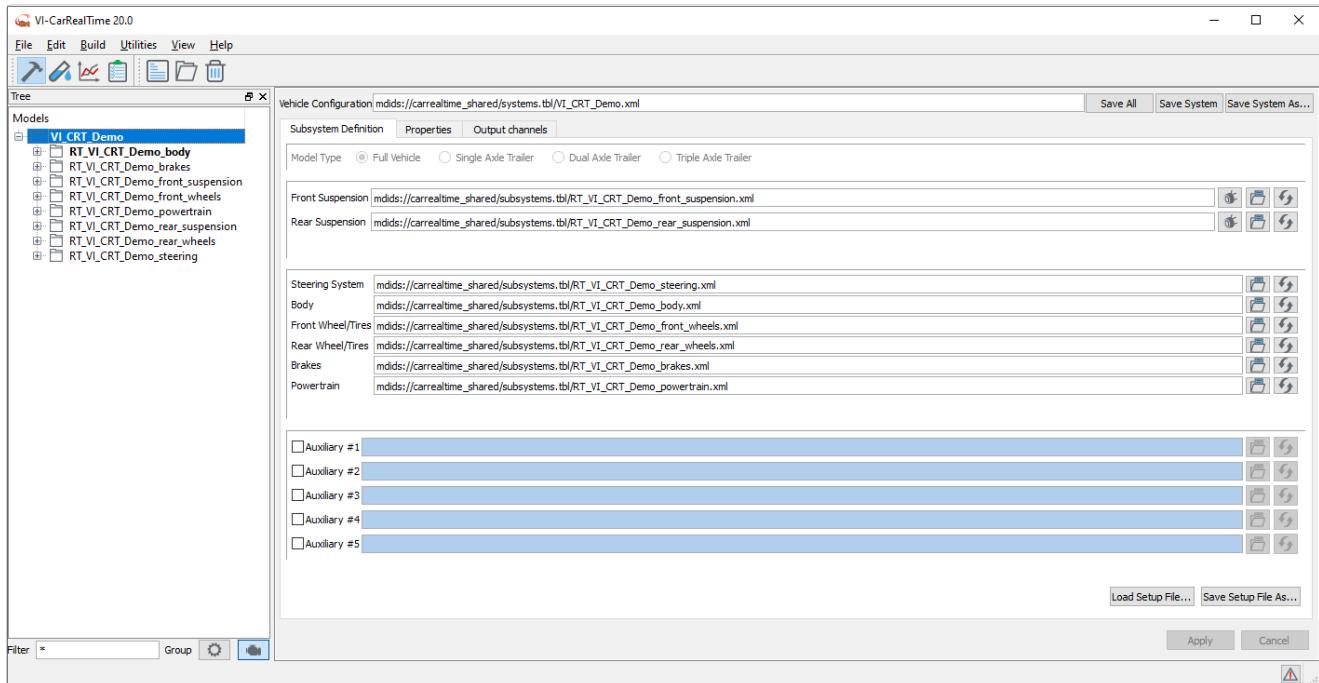
Build mode has a tree view of the model on the left that is always visible. It can be grouped either by subsystem



or by component type through the buttons:
[Learn more on how to use the VI-CarRealTime treeview.](#)

User editable parameters for the selected subsystem (or component) are shown in the frame on the right of the property editor.

By selecting the assembly name in the tree view the list of vehicle subsystems (or components) will be shown.



The user can change the vehicle composition by opening different subsystem files using [model configuration GUI](#).

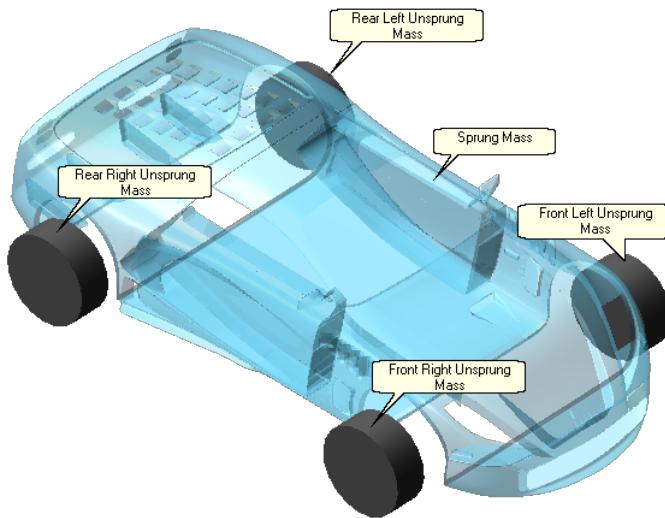
The system editor also includes a tab to access [vehicle properties and parameters](#).

Vehicle Model

VI-CarRealTime contains a simplified model of a four-wheeled vehicle. The vehicle is a reduced degree-of-freedom (DOF) model. It includes only **14 degrees of freedom** distributed as follows.

The model includes **5 rigid parts**:

- **vehicle chassis** (sprung mass)
- **4 wheel parts** (unsprung masses)



The vehicle chassis has **6 DOFs** while wheel parts have **2 DOFs** each (one for describing the motion w.r.t. the [vehicle body](#) and the other [wheel](#) spin). The model features the possibility of splitting the vehicle chassis into two parts connected by releasing one or more relative DOFs of the two halves. The relative DOFs are restrained through user defined linear stiffness.

The VI-CarRealTime [suspension](#) does not have linkages or bushings, and the [steering](#) system does not have parts for the steering wheel or rack.

Suspension and steering system properties (kinematic, compliance and component data) are described by lookup tables using a conceptual approach.

For each suspension the possibility of considering an [additional stiffness](#) in series to the main spring is supplied (further 4 DOFs when the feature is activated for all the suspensions).

Other vehicle subsystems ([brakes](#) and [powertrain](#)) are described using differential and algebraic equations, so no extra part is present in the model.

Body chassis torsional [compliances](#) may also be included (up to 6 DOFs) so the total vehicle DOFs increases to 20 when all compliances DOFs are active.

For each suspension it is possible to enable an additional [longitudinal degree of freedom](#) (further 4 DOFs when the feature is activated for all the wheels).

The Vehicle model's intent is to accurately predict overall vehicle behaviour for cornering, braking, and acceleration-performance studies for four-wheeled vehicles with independent-front and independent-rear suspensions. The simplified model is described in terms of commands and functions that use an internal development environment working as a symbolic manipulator tailored for deriving multibody equations and a code generator. Given the model description, it outputs C code for a simulation program for the particular model that is described. Parameters for the model (for example, wheelbase, spring stiffness, etc.) can be changed at run time and are passed in by input files.

The simplified vehicle **runs faster than real time**, depending on the computer platform. This makes the model potentially useful for HIL simulations and driving simulators. Because of the fast simulation times, the model is also useful as a plant for controller design, and for doing optimization in which a high number of simulation runs are required.

Conventions and references

Coordinate Systems

The reference frames and coordinate systems used in VI-CarRealTime are shown in the figure below. These coordinate systems are consistent with the following standards:

- ISO 8855, Road vehicles - Vehicle dynamics and road-holding ability - Vocabulary. 1991
- SAE Recommended Practice J670f, Vehicle Dynamics Terminology.

Global Reference Frame

The root body of the tree representing the rigid-body system is the global frame. The body is a Newtonian or inertial frame, which means that it does not accelerate in translation and does not rotate.

The global inertial frame of reference, N (N denotes Newtonian), for VI-CarRealTime models has unit vectors n_x , n_y , and n_z , and origin point N_0 as shown in the picture below.

Note: in design condition, Global Reference Frame coincides with Vehicle Reference System.

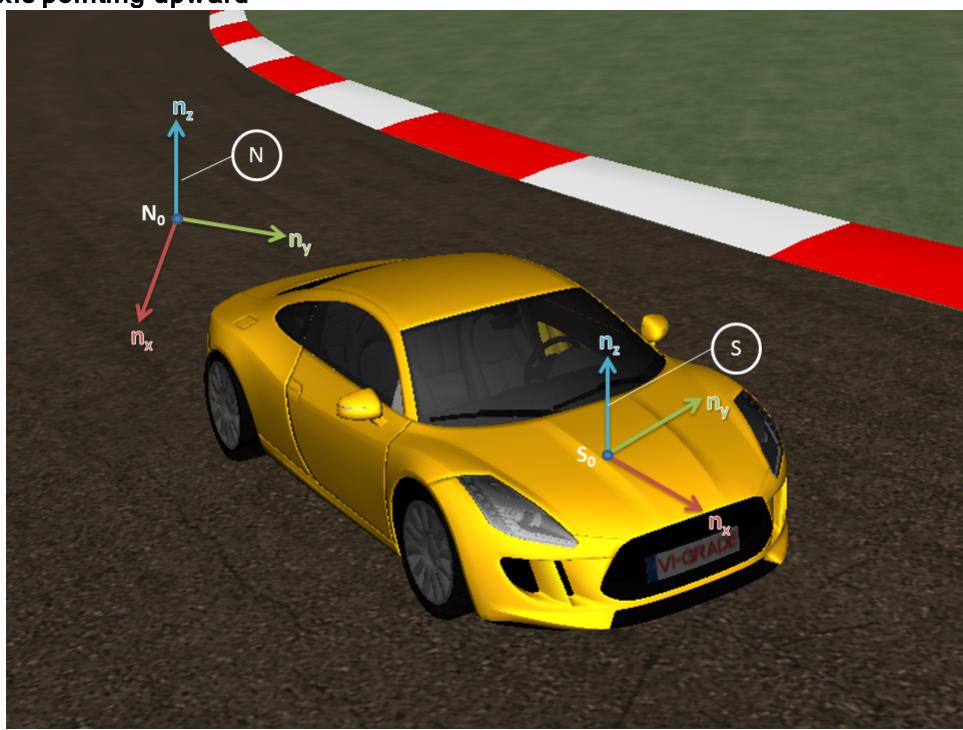
Vehicle Reference System

The vehicle model is positioned with respect to the global frame.

The origin of the Vehicle Reference System S_0 is located at Z=0 of Global Reference Frame and at half front vehicle track.

The triad is oriented, at design time, as the Global Reference Frame with:

- X+ axis pointing forward in the direction of motion
- Y+ axis pointing leftward
- Z+ axis pointing upward



Wheel Location Reference System (used in suspension definition)

It is defined by a triad positioned at the wheel center (left and right, at design time); the orientation is the same as VI-CarRealTime global reference system with X+ forward, Z+ vertical up, Y+ left.

The unsprung masses heights at design represent the [suspension_jounce](#) 0 condition for the suspension kinematic curves and for the elastic elements motion ratios: a wrong unsprung mass height with respect to the sprung mass generates a different chassis position after the static.

Driver Location Reference System

It is defined by a triad positioned on the rear axle midpoint; the orientation is the same of the vehicle reference system.

Road Reference System

It is defined by a triad having the Z+ axis oriented as the road normal vector, the X+ axis oriented as the vehicle forward direction and the Y+ axis accordingly.

Gyro Reference System

The gyro reference system (gyro marker) is located on the body (by default it is located coincident to [sensor point](#)) and is oriented with the Z axis coincident with global Z during the simulation with the X axis along the vehicle movement direction, so it follows the vehicle displacement, follows the yaw but not the pitch of the vehicle.

Suspension Reference System

It is defined by a triad positioned on the hub center (same as wheel center); the orientation is defined by camber/toe/caster angles.

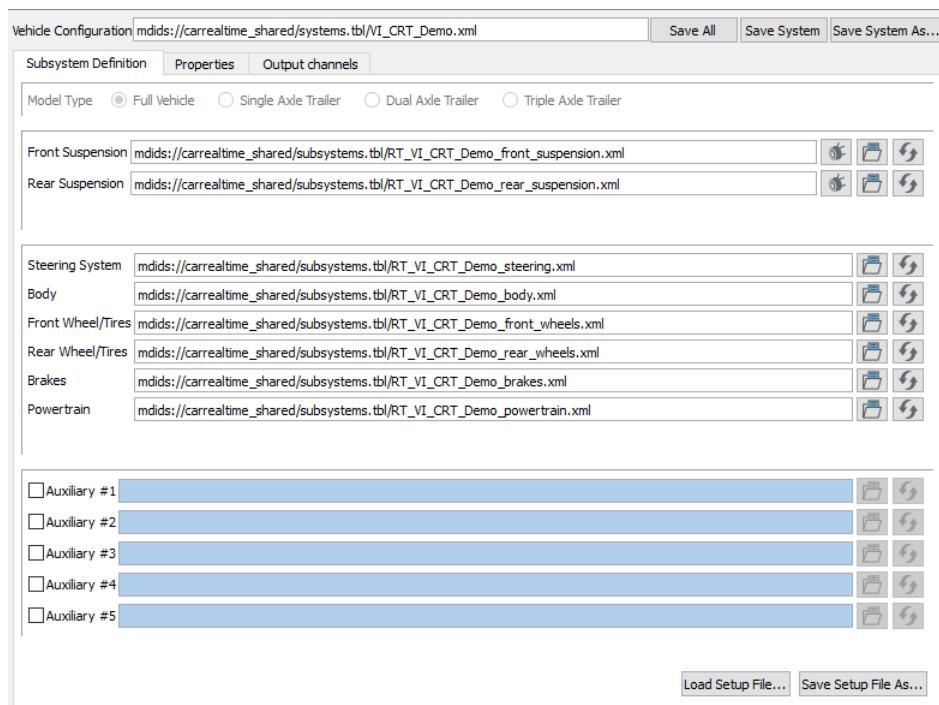
Model Configuration

The definition of vehicle model in VI-CarRealTime is realized using the system files, having xml format and stored in vehicle database under system.tbl table.

The system file contains the following set of information:

- Units
- List of vehicle subsystem files:
 - [Front Suspension](#)
 - [Rear Suspension](#)
 - [Steering](#)
 - [Body](#)
 - [Powertrain](#)
 - [Front Wheels and Tires](#)
 - [Rear Wheels and Tires](#)
 - [Brakes](#)
 - [Auxiliary Subsystem](#)

The property editor can be used to select appropriate property file for each of the subsystems.



It is possible to exchange subsystems among different vehicle models by selecting them in the system property editor. There is no need to save the system file since the changes done to vehicle composition (using the **Apply** push button) will be kept in memory during the complete VI-CarRealTime session.

Subsystem files can be loaded into model using the  button, reloaded from file using the  button.

Suspension subsystems can be created/edited using VI-SuspensionGen through  button. When a VI-SuspensionGen (.sgs) file having the same name of the selected VI-CarRealTime subsystem file is present in the database under suspensions.tbl directory, clicking the  button will automatically load the suspension file into VI-SuspensionGen.

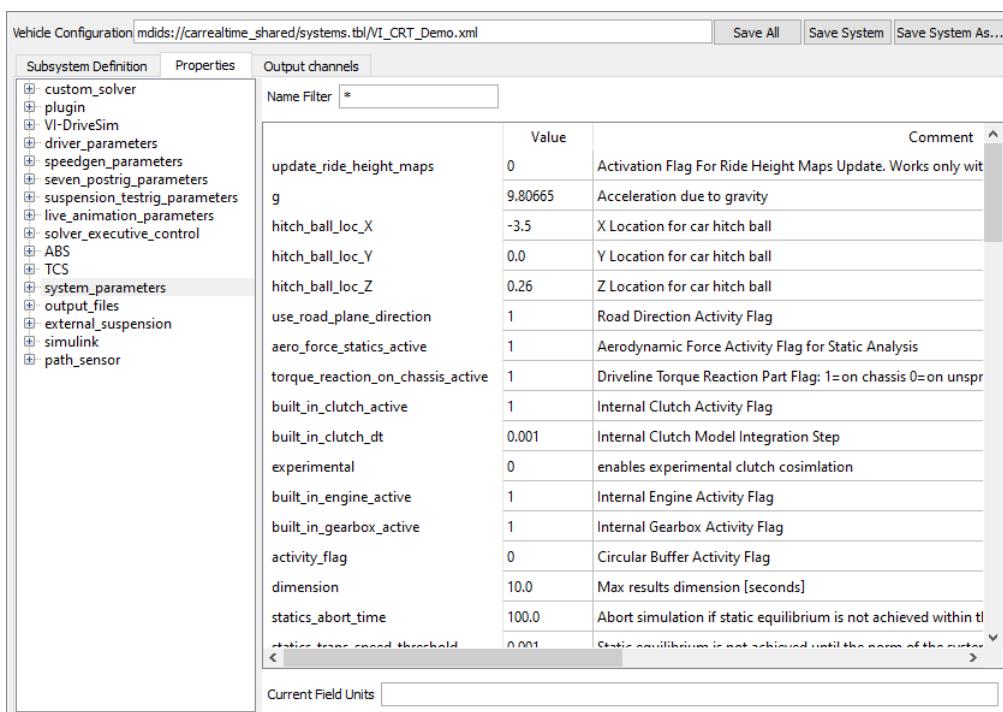
The following topics will be covered in this chapter:

- [Vehicle properties](#)
- [Output channels](#)
- [Clone Model](#)

Properties

Vehicle system files contain a set of information beyond the list of the subsystems.

The auxiliary information stored in system files can be edited using property editor.



The treeview for system properties include the following sections:

- [Custom Solver](#)
- [Plugin](#)
- [Driver Parameters](#)
- [VI-SpeedGen Parameters](#)
- [SevenPostrig Parameters](#)
- [Suspension Testrig Parameters](#)

- [Live Animation Parameters](#)
- [Solver Executive Control](#)
- [ABS](#)
- [TCS](#)
- [System Parameters](#)
- [Output Files](#)
- [Simulink](#)
- [External Suspension](#)
- [Path_Sensor](#)

The parameters included in this section are used to setup the name and activity of a custom solver library to be used for VI-CarRealTime simulation.

Custom Solver

- **active**
specifies if a user library have to be used.
- **library**
specifies the name of the custom solver library to be used (when active is set to 1).

The parameters included in this section are used to setup the name and activity of a plugin solver library to be used for VI-CarRealTime simulation.

Please, refer to [customizing](#) section of the help for more details on how to use plugin solver library.

Plugin

- **active**
specifies if a user library have to be used.
- **library**
specifies the name of the custom solver library to be used (when active is set to 1).

Driver parameters tree include all information about driver strategy and global vehicle information needed by driver algorithms used in MaxPerformance and StaticLapTime events. Driving machine events get similar information from the specific event file (sdf, xml), except for preview time and anticipatory compensation that are read here.

Below is a description of the parameters:

- **preview_time** (used in DrivingMachine, VIDriver, MaxPerformance events)
The preview time parameter define how "far" the controller looks ahead of the vehicle for planning the control action. The effect of increasing the preview time is that the controller will produce smoother control actions on the steering but on the other side the path tracking accuracy will be reduced.
- **anticipatory_compensation** (used in DrivingMachine, VIDriver, MaxPerformance events)
In order to include the delay between the imposed control action and the vehicle response, a time constant could be specified. The controller will anticipate its own actions accordingly. Typical values range from 0 to 0.15 seconds.
- **max_perf_max_itx** (used in MaxPerformance events)
Maximum number of iterations. The event is stopped when the number of iterations overpasses such value, even if the input path (DRD) has not been totally covered.
- **max_perf_save_restore_mode** (used in MaxPerformance events)

VI-CarRealTime

It allows to switch on (1) or off (0) the save/restore of the states of the model. Set the variable to 0 can avoid errors when simulating with a model including a plug-in dll whose code doesn't implement save/restore of its states.

A detailed description of the following parameters can be found in the [VDF file](#) documentation.

- **DT1** used in VIDriver, MaxPerformance events.
- **DT2** used in VIDriver, MaxPerformance events.
- **gear_shifting_time** used in VIDriver, MaxPerformance events.
- **clutch_raise_time** used in MaxPerformance events.
- **clutch_fall_time** used in VIDriver, MaxPerformance events.
- **throttle_raise_time** used in VIDriver, MaxPerformance events.
- **throttle_fall_time** used in VIDriver, MaxPerformance events.

Speedgen parameters tree includes advanced settings for VI-SpeedGen static solver.

Advanced parameters affecting the "standard" speed profile.

- **use_design_configuration**

this flag manages the masses geometry configuration used to initialize the static vehicle model.
If 0 speedgen vehicle model cog represents the after static/setup configuration, if 1 the design one (legacy mode).
Allowed values: 0 - 1

- **lazy_gear_shift**

enable/disable the gear shifting logic based on up-shift and down-shift rpms. If enabled the speed profile is calculated shifting the gears using the engine rpms; if disabled on each path point the vehicle uses the gear providing the maximum value of power (measured at wheels).
Allowed values: 'TRUE' - 'FALSE'

- **pwr_scaling_factor**

scaling factor of the powertrain map.

In the **defaults** subtree:

- **longVelMin**

minimum longitudinal velocity used as lower limit for the pure lateral algorithm [m/sec].

- **longVelMax**

maximum longitudinal velocity used as upper limit for the pure lateral algorithm [m/sec].

- **longVelTol**

longitudinal velocity tolerance used for the pure lateral algorithm convergence [m/sec].

- **longAccMin**

minimum longitudinal acceleration used as lower limit for the longitudinal acceleration/deceleration algorithm [m/sec²].

- **longAccMax**

maximum longitudinal acceleration used as upper limit for the longitudinal acceleration/deceleration algorithm [m/sec²].

- **longAccTol**

longitudinal acceleration tolerance used for the acceleration/deceleration algorithm convergence [m/sec²].

Advanced parameters affecting the "post-processing" speed profile.

- **speed_postprocessing**

enable/disable the speed postprocessing mode. During the postprocessing, VI-SpeedGen modifies the calculated speed profile scaling the high speed brake actions (see the 'SCALE_HIGH_SPEED_BRAKING' parameter), joining the apex points closer than a given threshold (see the 'APEX_JOIN_DISTANCE' parameter) and then applying a smoothing (tunable with the SMOOTH_SAMPLES parameter).

Allowed values: 'FALSE' - 'TRUE'

- **apex_join_distance**

threshold distance used when joining the apex points. If two consecutive apex points are closer than the given threshold, those points are joined using constant acceleration.

- **smooth_samples**

number of samples used by the smoothing algorithm. It uses a moving average with a window of $2n+1$ samples (where n is the value of smooth samples). If the value is 0 or negative, the smoothing is disabled.

Scale high speed braking parameters.

- **scale_high_speed_braking**

when enabled, the speed profile is calculated scaling the braking force using the 'BRAKE_SCALING_MAP' map. The scaling is applied only when the distance between the braking point and the apex point is smaller than 'HIGH_SPEED_BRAKING_MAX_DELTAS', and the difference of speed between the braking point and the apex point is smaller than 'HIGH_SPEED_BRAKING_MAX_DELTA_V'. Those two parameters are needed to avoid scaling the brake force when it's needed to going hard on brakes.

Allowed values: 'TRUE' - 'FALSE'

- **high_speed_braking_max_delta_s**

maximum distance between the braking point and the following apex point needed to enable the brake pedal scaling function of speed.

- **high_speed_braking_max_delta_v**

maximum speed difference between the braking point and the following apex point needed to enable the brake pedal scaling function of speed

In the **high_speed_braking_parameters** subtree the parameters to define the brake scaling vs. speed map are collected. The map is built as a ramp in the **speed_min - speed_max** range with the limit values that are **brake_min** (brake scaling at **speed_min**) and **brake_max** (brake scaling at **speed_max**).

Note: to enable the brake scaling both **scale_high_speed_braking** and **speed_postprocessing** must be activated.

The following parameters are used by VI-SpeedGenEvo (when [Use Suspension Data](#) flag is checked):

- **solverTolerance**

suspension jounce tolerance. The vehicle equilibrium is achieved when the difference between the estimated jounce and the computed jounce is below the tolerance.

- **suspKinematicsPreEventIntegrationStep**

integration time step of the suspension testrig analysis used to characterize the vehicle ground stiffness.

- **suspKinematicsMapsNPoints**

number of points in static model suspension kinematic maps.

- **frontSuspKinWheelTravelMax**

maximum wheel travel of the front suspension used for parallel and opposite travel analysis.

- **frontSuspKinWheelTravelMin**
minimum wheel travel of the front suspension used for parallel and opposite travel analysis.
- **rearSuspKinWheelTravelMax**
maximum wheel travel of the rear suspension used for parallel and opposite travel analysis.
- **rearSuspKinWheelTravelMin**
minimum wheel travel of the rear suspension used for parallel and opposite travel analysis.

Sevenpostrig parameters tree include settings for SevenPostRig analysis.

Below is a description of the parameters:

- **body_stake**
The body stake is represented by three PID controllers. The stake is used in order to block the vehicle chassis along the xy plane and around the yaw dof.
 - **x_translational_kp**
Proportional gain for chassis x-translation stiffness lock PID.
 - **y_translational_kp**
Proportional gain for chassis y-translation stiffness lock PID.
 - **z_rotational_kp**
Proportional gain for chassis z-rotation stiffness lock PID.
- **activity_flag**
Set the activity of all the body stake controllers during the sevenpostrig simulation: 0=inactive, 1=active.
- **actuator_gains**
The vertical displacement of each wheel is controlled by a PID controller.
 - **hub_actuators_kp**
Proportional gain for the hub vertical displacement tracking PID.
 - **hub_actuators_ki**
Integral gain for the hub vertical displacement tracking PID.
 - **hub_actuators_kd**
Derivative gain for the hub vertical displacement tracking PID.
- **front_aeroforce_actuator_loc_*(X,Y,Z)**
Location of front aerodynamic force actuator.
- **rear_left_aeroforce_actuator_loc_*(X,Y,Z)**
Location of rear left aerodynamic force actuator.
- **rear_right_aeroforce_actuator_loc_*(X,Y,Z)**
Location of rear right aerodynamic force actuator.
- **(front_left,front_right,rear_left,rear_right)*_spring_length**
Spring design length. It is used in simulation with spring excitation target when the input signal is the spring length rather than spring deflection.
- **controller_gain**
Controller reaction value while tracking the reference (hub acceleration or spring length) signals. It is used only when the simulation mode is in Hub Acceleration Tracking or Spring Length Tracking mode (see [SPC file format](#)). When a simulation is submitted, the sevenpostrig calculates its internal reference profile. The tracking routines work using as reference the hub position time history, which is reconstructed from the original signal read from the .spc file.
- **statistics**
Activation flag for additional statistic outputs. If active a set of standard ride metrics will be added to the end of simulation report file.
- **actuator_max_jounce**
Upper displacement limit for testrig actuators.

- **actuator_min_jounce**
Lower displacement limit for testrig actuators.
- **unlock_wheels**
Enables wheel rotational degree of freedom (available only in contact patch actuation mode).
- **psd_output**
Activation flag for Psd data output. If active, at the end of simulation an automatic PSD calculation will be done for the following channels:
 - Individual hub vertical accelerations;
 - Individual tyre contact patch load;
 - Body heave at front axle centerline, CG and rear axle centerline;
 - Body pitch angle;
 The data will be written in a CSV file named <output_prefix>_psd.atl .
- **psd_segment_point**
Numbers of points for time sequence used for PSD calculation;
- **psd_overlap**
Time sequence overlapping percentage (%);
- **psd_fft_points**
Number of points used by Fast Fourier Transform algorithm (it has to be an exact power of 2).
- **psd_sampling_freq**
Sampling frequency for PSD calculation.

Suspension Testrig Parameters tree include settings belonging to the suspension testrig simulation:

Actuator Gains:

- **hub_actuators_kp**
Proportional gain for the hub displacement tracking PID.
- **hub_actuators_ki**
Integral gain for the hub displacement tracking PID.
- **hub_actuators_kd**
Derivative gain for the hub displacement tracking PID.

Use pid constant part:

if 1 previous PID value will be summed up to runtime pid value - useful to increase the convergence in case of small unsprung masses-

Static threshold :

testrig step is not achieved until the norms of the system speeds & acceleration are below this value.

Setup:

- **active**
if 1, a preliminary static analysis will be performed in order to define the suspension configurations after setup. The setup configuration will be used as starting point for the suspension testrig analysis.
- **gravity_for_testrig_setup**
it defines the value of the gravity field (in m/s²). Set the value to 0 allows to run suspension testrig analyses without gravity.

Live animation parameters tree include settings relative to connection between VI-CarRealTime and VI-Animator

Below is a description of the parameters:

- **multicast_server_flag**

VI-CarRealTime

The parameter specifies the server type will be created by VI-CarRealTime. If 1, the signals for live animation will be sent through a multicast server. The main difference is that using the multicast mode, multiple clients can connect to the same server and it is possible to connect/disconnect clients multiple time but on the other side more network traffic is generated. In addition multiple server sending data to the same multicast address can interfere each other. For all these reason the default setting for this parameter is 0.

When the multicast option is not active, the VI-CarRealTime Solver process will wait up to 60 seconds for the client (VI-Animator) connection before starting the dynamic computation.

- **udp_port**

The parameter represent the udp port number used for server connection. If 0, the port number will be automatically detect at simulation run.

- **tcp_port**

This is the tcp port number used to communicate with VI-Animator .

- **reuse_animator_flag**

Setting this flag equal to 1 user can reuse a previous instance of VI_Animator for next analysis runs. This option is effective only in multicast mode. Please note that in order to reuse a previous opened VI-Animator instance user must not push the stop button on VI-Animator interface.

- **max_solver_waiting_time**

maximum time that VI-CarRealTime waits for a VI-Animator connection.

Solver Execute Control parameters tree include settings belonging to the solver calculation:

Multithread subtree:

- **tire_calc_active_flag**

enables multithread calculation for tires. Currently this option affects only F-Tire model and requires a specific F-Tire license.

Extended log subtree:

- **active verbose mode**

enables more detailed runtime log information.

- **active runtime log**

enables runtime writing of simulation log file.

Compliance Maps subtree:

- **use polinomial flag**

if enabled compliance splines will be replaced by a 3D polinomial surface.

- **polinomial order**

order of compliance surface polinomial.

- **use spline extrapolation**

enables spline extrapolation (allowed values 0/1).

Kinematic Maps subtree:

- **use polinomial flag**

if enabled kinematic splines will be replaced by a 3D/2D polinomial surface/curve.

- **polinomial order**

order of kinematic polynomial surface/curve.

- **use spline extrapolation**

enables spline extrapolation (allowed values 0/1).

Kinematic subtree:

- **steering acceleration dependency**

if enabled, front suspension second derivative of base, track, toe, side view angle and camber will depend by second derivative of steering actuator (default value 0->inactive).

Compliance subtree:

- **cross compliance jounce bias**

specifies the jounce used as second independent variable on cross compliance splines (0: side jounce - 1: opposite jounce).

- **compliance velocity computation flag**

activity flag for compliance deformation velocity computation.

The standard compliance data describes the equilibrium deformation generated by a static load so no information about the deformation velocity are available: this flag enables a first order model to reconstruct the compliance deformation velocity starting from a numerical differentiation of compliance displacements.

- **compliance velocity cutoff filter**

cut off frequency of first order filter used to smooth the discrete compliance velocity.

- **compliance extra output activity**

if 1, compliance states additional outputs will be added to standard result set.

- **compliance map linear extrapolation flag**

if 1, compliance maps linear extrapolation will be possible (default = flat outside the map).

- **compliance map translational cut off threshold**

threshold value for translational compliance map activation.

- **compliance map rotational cut off threshold**

threshold value for rotational compliance map activation.

Runtime Road Plane:

- **flying car computation mode**

It allows the user to choose how the runtime road plane is computed when car's contact patch lose contact with the ground (flying car). The following choices are available:

- *standard computation (value = 0)*

VI-CarRealTime solver computes a valid road plane when at least 3 tires are in contact or when 2 tires, belonging to the same suspension, are in contact. In the latter case the solver uses the 2 contact patches location and 1 road normal direction to compute the plane.

- *estimated contact points (value = 1)*

road plane is estimated by using the theoretical contact patches of the first 3 tires.

- *Maintain heights for aero/skidplate (value = 2)*

the heights of the first 3 contact patches right before the flying begins is used to compute the plane.

Static Result subtree:

- **active dump**

if 1 solver will perform only the static analysis and the static results will be saved.

Spline subtree:

- **method**
spline interpolation method (options: DeBoor, Akima).

Driveline subtree:

- **automatic_initial_gear_selection**
if 1 and vehicle model uses automatic transmission, the option enables the automatic initial gear selection depending on engine limits and shifting parameters.
- **enable_extra_parts**
when the flag is set to 1, the rigid parts in powertrain subsystem are treated as extra parts by VI-CarRealTime solver, such allowing to connect rigid bodies directly to the main engine part (remove the constraint that a rod part must be attached to the chassis).
- **enable_extra_output**
when the flag is set to 1, additional [differential outputs](#) are enabled.

External Input:

- **computation mode**
it allows the user to define how to deal with external input values. To be used with Runge-Kutta midstep integrator. Available options are:
 0 = standard computation . No actions from the solver, inputs are used as they are.
 1 = backward interpolation . The mean value between actual integration step and previous integration step is used for each external input.
 2 = forward extrapolation . Current integration step and previous integration step input values are used to extrapolate the input value to be used at mid step.
- **use_damper_force_extrapolation**
it allows the user to define how to deal with external input damper force extrapolation. Available options are:
 0 = standard computation . No actions from the solver, input damper force is used as it is.
 1 = force extrapolation . The actual input damper force value is obtained extrapolating linearly the points damper force vs. damper velocity at previous integration step and current integration step.

Kingpin Moment Computation:

- **extra output activity**
flag which enables the creation and the computation of [additional outputs](#) for each single contribute of the kingpin moment (left and right).
- **inertial contribute activity**
flag which switches on [1] off [0] the inertia contributes in kingpin moment computation.
- **steering_axis_compliance_correction_activity**
flag which switches on [1] off [0] the compliance contributes on steering axis computation. If active, compliance splines will be automatically computed using the [instant axis approach](#) described on steering subsystem. Moreover, when the option is active, solver neglects existing steering axis kinematic splines and recomputes them using the same method.

Vehicle:

- **simplified_model**

activity flag which enables the usage of a simplified vehicle model. The flag can be modified in order to use simplified versions of the VI-CarRealTime model to run specific events.

In particular the following options are available:

- *simplified_model = 0* , uses the VI-CarRealTime model as it is;
- *simplified_model = 1* , uses the maximum simplified version of the model;
- *simplified_model = 2* , uses the kinematic version of the model.

The following tables summarizes the features that are activated/deactivated for each model flavor:

	Simplified Model (simplified_model = 1)	Kinematic Model (simplified_model = 2)	Full Model (simplified_model = 0)
Body and Aerodynamic	YES	YES	YES
Brakes	YES	YES	YES
Driveline	YES	YES	YES
Tires	YES	YES	YES
Suspension Kinematic	NO	YES	YES
Steering Kinematic	NO	YES	YES
Suspension Compliance	NO	NO	YES*
Advanced Steering	NO	NO	YES*
Body Compliance	NO	NO	YES*
Additional Degrees of Freedom	NO	NO	YES*
Engine Part with bushings	NO	NO	YES*
Body on Frame with bushings	NO	NO	YES*
Skidplate	NO	YES*	YES*
External Plugin Library	YES	YES*	YES*

(*) the feature must be active in the VI-CarRealTime model you're using.

The parameters included in this section are used to setup the control systems embedded in VI-CarRealTime model.

Embedded ABS system (ABS)

- **ABS_control_threshold**
longitudinal slip threshold for ABS engagement.
- **ABS_control_gain**
ABS controller proportional gain.
- **Built_in_ABS_active**
ABS controller activity flag (1: active; 0: inactive).
- **low_speed_threshold**
threshold vehicle speed below which the ABS system is switched off.

The parameters included in this section are used to setup the control systems embedded in VI-CarRealTime model.

Embedded Traction Control System (TCS)

- **TCS_control_threshold**
longitudinal slip threshold for TCS engagement
- **TCS_control_gain**
TCS controller proportional gain
- **Built_in_TCS_active**
TCS controller activity flag (1: active; 0: inactive)

The parameters embedded in this section are used to enable/disable flags for further models in Similink.

Simulink flags:

- **aerodynamics**

- **active_fx_body_driver_feedforward**

This parameter represents the drag force feedforward activity flag w.r.t. the *chassis* orientation frame. [0 disabled/ 1 enabled]

- **active_fx_tracking_driver_feedforward**

This parameter represents the drag force feedforward activity flag w.r.t. the gyro. [0 disabled/ 1 enabled]

- **additional_input**

- **powertrain_extra_input**

This parameter enables/disables a flag for external (Simulink) additional powertrain inputs. [0 disabled/ 1 enabled]

- **wheel_orientation_extra_input**

This parameter enables/disables a flag for external (Simulink) wheel orientation control. [0 disabled/ 1 enabled]

- **engine_bushing_extra_input**

This parameter enables/disables a flag for external (Simulink) engine bushing forces and torques. [0 disabled/ 1 enabled]

In particular with this flag you enable the flag for adding additional forces and torques (modeled in Simulink) to the bushings which support the engine block; these external forces and torques components will be added to the elastic forces and torques of the [bushings panel](#) embedded in [powertrain subsystem](#). Note that this panel is optional, so it's not available for all the powertrain subsystems.

The parameters embedded in this section are used enable/disable the activity of auxiliary set of input to be used to interface the vehicle model with external suspension model (entire suspension or single component replacement is allowed). The description of the full list of auxiliary input can be found in [Input Channels In VI-CarRealTime](#)

External Suspension flags:

Front

- **user_input_activity**

This flag toggles the activity of auxiliary input for front suspension subsystem.

- **user_spring_input**

This flag toggles the activity of auxiliary input for front main springs.

- **user_damper_input**

This flag toggles the activity of auxiliary input for front main dampers.

- **user_bumpstop_input**

This flag toggles the activity of auxiliary input for front main bumpstop.

- **user_reboundstop_input**

This flag toggles the activity of auxiliary input for front main reboundstop.

- **user_arb_input**

This flag toggles the activity of auxiliary input for front antiroll bar.

- **user_kinematic_scaling_input**

This flag toggles the activity of auxiliary input for front suspension kinematic scaling factors.

- **`user_compliance_scaling_input`**

This flag toggles the activity of auxiliary input for front suspension compliance scaling factors.

Rear

- **`user_input_activity`**

This flag toggles the activity of auxiliary input for rear suspension subsystem.

- **`user_spring_input`**

This flag toggles the activity of auxiliary input for rear main springs.

- **`user_damper_input`**

This flag toggles the activity of auxiliary input for rear main dampers.

- **`user_bumpstop_input`**

This flag toggles the activity of auxiliary input for rear main bumpstop.

- **`user_reboundstop_input`**

This flag toggles the activity of auxiliary input for rear main reboundstop.

- **`user_arb_input`**

This flag toggles the activity of auxiliary input for rear antiroll bar.

- **`user_kinematic_scaling_input`**

This flag toggles the activity of auxiliary input for rear suspension kinematic scaling factors.

- **`user_compliance_scaling_input`**

This flag toggles the activity of auxiliary input for rear suspension compliance scaling factors.

Steering

- **`user_input_activity`**

This flag toggles the activity of auxiliary input for steering subsystem.

The system parameters section contains parameters mainly related to model setup, components, and general system settings.

- [**Powertrain**](#)
- [**Statics**](#)
- [**Body Compliance**](#)
- [**Setup**](#)
- [**Differential**](#)
- [**Anti Roll Bar**](#)
- [**Std_tir_ref**](#)
- [**Dampers**](#)
- [**Trailer**](#)
- [**Update Ride Height Maps**](#)
- [**Circular Buffer**](#)
- [**General Settings**](#)

g: gravity field value (m/s^2)

Trailer parameters are:

- **`hitch_ball_loc[X, Y, Z]`**

this is the location of the hitch ball w.r.t the vehicle reference system.

Aerodynamics parameters are:

- **use_road_plane_direction**

Flag used to define the reference system for aerodynamic forces. When it is set to 1 aero forces are applied w.r.t. road plane; when it is set to 0 aero forces are applied w.r.t. the chassis reference frame.

- **aero_forceStatics_active**

Flag used to keep aerodynamic forces active during static equilibrium. When it is set to 1 aero forces are active in statics.

Powertrain tree includes the subtrees and parameters related to clutch, engine and gearbox. The activity flags can be used to enable/disable the single components of powertrain (the case could be of having a clutch model in Matlab/Simulink and keep using internal engine and gearbox)

Clutch

- **builtin_clutch_active**

flag used to switch the activity of internal clutch model. (1: active; 0: inactive)

- **builtin_clutch_dt**

when clutch modeling is enabled an internal solver algorithm is used to solve clutch states. the parameter is used to set the internal clutch model integration step.

- **experimental**

flag used to enable the experimental clutch cosimulation. (1: active; 0: inactive)

Such experimental feature consists in evaluating the *analytic* Jacobian matrix of the equations of the clutch model, whereas the standard mode computes the *numeric* Jacobian matrix.

Engine

- **builtin_engine_active**

flag used to switch the activity of internal engine model. (1: active; 0: inactive)

Gearbox

- **builtin_gearbox_active**

flag used to switch the activity of internal gearbox model. (1: active; 0: inactive)

Driveline

- **torque_reaction_on_chassis_active**

flag used to select the reaction part on which the driveline torque acts. (1: chassis; 0: unsprung mass)

Dampers parameters are:

- **damper_lp_filter**

value used to set the activity of the filter acting on the main dampers. When this value is set to 0 the filter is inactive. For any other positive value the cutoff frequency of the filter is given by: $1/(output_step * damper_lp_filter)$.

The Anti Roll Bar parameters are:

- **[front/rear]_aux_roll_damping**

proportional damping used in auxiliary anti roll forces. Default is 5% (0.05).

- **setup_active**

flag that specifies if the anti roll forces have to be deactivated during setup. When it is set to 0, anti roll forces are active in setup. When set to 1, forces are deactivated during setup and at the end of setup procedure the delta jounce and average jounce in input to the anti roll force map are offset to zero.

VI-CarRealTime chassis model can be split into two parts in order to consider body deformation effects happening during analysis. In general the two body halves can have up to six relative degrees of freedom. The user can select any combination he likes and, except in case of selecting the only torsional DOF, when the other DOFs are kinematically locked, in all other combination the locking of unused DOFs is done by PID driven forces and torques controlling halves relative movement. The body compliance tree contains the PID gains parameters.

- **rotational_[kp,kd,ki]**
Gain for chassis rotational stiffness lock PID.
- **translational_[kp,kd,ki]**
Gain for chassis translational stiffness lock PID.

The differential parameters are:

- **locked_differential_[kp, ki, kd]**
locked differential PID controller gains, used to keep zero angular velocity difference for wheels belonging to same axle.
- **LSD_omega_threshold**
limited slip differential threshold for changing differential torque sign.
- **user_differential_lp_filter**
Activity flag for low pass filter applied to user differential model in order to avoid solver instability due to high stiffness. The fundamental cutoff frequency is 100hz, user can change this using the **user_differential_lp_filter** value. The cut off frequency applied at the differential model will be 100/user_differential_lp_filter Hz. If user_differential_lp_filter has zero value no filter is applied.

Ride Height Maps Update parameters are:

- **update ride height maps**
Activation Flag For Ride Height Maps Update; works only with DrivingMachine events. The parameter has children to set the defaults used during StaticLapTime and MaxPerformance events automatic RH maps update.
The automatic ride height maps update is computed through a coast down event split into two mini-maneuvers. The first one is intended for settling the aero forces and it uses the two smoothing time parameters below to ramp up the aero forces to their nominal value; the first mini-maneuver end condition is set by the aero_settling time. During the second mini-maneuver the neutral gear is engaged and the vehicle velocity is decreased through aero drag. The measured RH are then used to populate RH maps.
- **v_init**
Initial velocity for automatic ride height maps computation (MaxPerformance and StaticLapTime events). [m/s]
- **v_end**
Velocity end condition for automatic ride height maps computation (MaxPerformance and StaticLapTime events). [m/s]
- **dt**
Integration step size for automatic ride height maps computation (MaxPerformance and StaticLapTime events). [s]
- **dtout**
Output step size for automatic ride height maps computation (MaxPerformance and StaticLapTime events). [s]
- **aero_drag_smoothing_time**
Smoothing time for Aerodynamics Drag Force (MaxPerformance and StaticLapTime events). [s]
- **aero_downforce_smoothing_time**
Smoothing time for Aerodynamics Down Forces (MaxPerformance and StaticLapTime events). [s]
- **aero_settling_time**

Settling Time for Aerodynamics Forces (MaxPerformance and StaticLapTime events). [s]

- **experimental_mode**

Flag to enable Ride Height maps description based on load (set flag value to 1) instead of longitudinal velocity (set flag value to 0). When the flag is 0 a dynamic simulation using [v_init](#) as initial speed with open clutch on a straight track is run: the simulation runs until the vehicle velocity, due to aero drag, reaches [v_end](#). When the flag is 1, starting from vehicle static equilibrium conditions, front and rear axle load is swept from [minimum load percentage](#) to [maximum load percentage](#) values.

- **minimum_load_percentage**

Percentage of axle load with respect to static condition to be used as minimum for ride height independent variable.

- **maximum_load_percentage**

Percentage of axle load with respect to static condition to be used as maximum for ride height independent variable.

- **loadcase_duration**

Time required to increase the axle load. [s]

The Statics Parameters are:

- **statics_abort_time**

Abort simulation if static equilibrium is not achieved within this time.

- **statics_trans_speed_threshold**

Static equilibrium is not achieved until the norm of the system translational speeds is below this value.

- **statics_trans_accel_threshold**

Static equilibrium is not achieved until the norm of the system translational accelerations is below this value.

- **statics_angular_speed_threshold**

Static equilibrium is not achieved until the norm of the system angular speeds is below this value.

- **statics_angular_accel_threshold**

Static equilibrium is not achieved until the norm of the system angular accelerations is below this value.

- **statics_use_brakes**

Flag used to activate brakes activity during statics.

- **statics_use_linear_damper**

Flag used to switch to linear damper in statics. Useful to increase convergence rate of the statics.

- **statics_linear_damper_rate_[front/rear]**

damper rate used if statics_use_linear_damper flag is set to 1.

- **statics_use_linear_spring**

Flag used to switch to linear springs in statics.

- **statics_linear_spring_rate_[front/rear]**

spring rate used if statics_use_linear_spring flag is set to 1.

- **statics_brake_damping**

allows the user to define a rotational damping value to be applied in wheel rotational degree of freedom during static analysis.

- **statics_integration_time_step**

integration time step to be used for static/setup analysis only (non positive value to inherit from dynamic event).

Note: When using MATLAB/Simulink to replace built in springs or damper components, internal property files have to be replaced with zero force characteristics ones (i.e. `vi_null_spring.xml` , `vi_null_damper.xml` from carrealtime_shared database). Since external inputs coming from Simulink are not applied during statics it may cause convergence problems in case of springs. Enabling the `statics_use_linear_spring` flag can solve the issue.

The Setup Parameters tree contains all system level information relative to the vehicle setup.

Global Setup parameters:

- **mode**
flag for ride height and cross weight setup method (1: using pushrod; 2: using spring preload). When this flag is set to 2 (preload) pin height adjustment is not performed, even if enabled in the model suspension tab.
- **tolerance**
tolerance allowed for setup targets; warning issued for relative errors greater than tolerance [0.0 - 1.0].
- **statistics**
boolean flag (0=False, 1=True) used to enable setup statistics in simulation log file.
- **orientation_reference**
value used to select the desired reference for the wheels angle targets.
1 -> target angles are measured with respect to sprung mass (chassis)
2 -> target angles are measured with respect to ground.
- **setup on flat road**
boolean flag (0=False, 1=True) to enabling road flat model during setup analysis.

Ride_height:

The PID controllers used for the vehicle setup have a different meaning according to the setup mode specified in the "mode" parameter. When adjusting vehicle ride height with pushrod (mode 1) the PID controller changes the pushrod length (meters). When adjusting the vehicle using spring preload (mode 2) the PID controller changes the spring forces (Newton)

- **ride_height_preload_[kp,kd,ki]**:
PID gains for preload (mode 2) ride height setup.
- **ride_height_standard_[kp,kd,ki]**:
PID gains for pushrod (mode 1) ride height setup.

Note: Suggested values for the PID gains are:

mode 1 (standard)	standard_kp=1	standard_kd=0	standard_ki= 10
mode 2 (preload)	preload_kp=1.0E5	preload_kd=1.0E4	preload_ki=1.0E6

A tuning of the parameters should be done depending on the model.

Pin_height:

- **pin_height_[kp,kd,ki]**
controller gains for pin height adjustment. These setup is only done when using pushrod method (mode 1).

Note: Suggested values for the PID gains are: kp =1e5, kd=1e4, ki=1e6. A tuning of the parameters should be done depending on the model.

Cross weight:

- **ratio_[kp,ki]**
controller gains for ratio method (target: force ratio).
- **force_[kp,ki]**
controller gains for front and rear force methods (target: force difference).
- **preload_RH_CW_gain**

controller gain multiplying force_kp and force_ki used when RH mode is set to 2 (preload).

Note: For both ratio and force method the same consideration made above for the ride height PID gains apply. Suggested values are:

mode 1 (pushrod)	ratio_kp=0.1	ratio_ki=1.0
mode 2 (preload)	force_kp=1e-5	force_ki =1e-4
		preload_RH_CW_gain =1.0E6

Setup_wheels:

- **setup_wheels_active**
flag to activate the rigid tires during the setup phases.
- **setup_wheels_normal_stiffness**
stiffness value used for rigid tire.
- **front/rear_setup_wheels_radius**
value of setup wheels radius.

STD_TIR_REF parameters are:

- **std_tire_ref_*(X,Y,Z,PSI,THETA,PHI)**
tire/road reference marker absolute position and orientation.

The parameters included in this section are used to setup the activity and the dimension of an optional circular buffer to be used by VI-CarRealTime simulation to store simulation results. This can be useful when running very long simulations, especially when running external events (e.g. VIDriveSim), in which the duration of the run is usually unknown.

Activating the circular buffer option will cause VI-CarRealTime to save only the latest section of the simulation results. Length of the buffer, in seconds, is specified by the user.

Circular_buffer

- **activity_flag**
specifies whether circular buffer should be enabled or not.
- **dimension**
specifies the size of the buffer, in seconds.

The parameter tree in the current section contains options to define the format of simulation output files. By setting the parameter value to 1 it activates the corresponding output file type.

Available formats are:

- **XML**
- **PI Toolbox**
- **Tabular**
- **CSV**
- **ATLAS**
- **Matlab**
- **Req**
- **Nam_Overwrite** (Applies only if Request format is active. Specifies the overwrite strategy for the nam file: If 1, nam is always overwritten, if 0 is overwritten in files only mode, not in interactive mode).
- **SevenPost**

This tree contains the options to generate a .spc file (see [SPC file format](#)) from a full vehicle analysis.
Below is a description of parameters:

- **SPC**

This is the flag to activate the writing of spc from a full vehicle analysis.

- **AeroMode**

This parameter sets the activity of front/rear actuators in the sevenpost event (0 = 'DATA' , 1 = 'MODEL').

- **OperativeMode**

This parameter sets the which kind of target the sevenpostrig simulation will follow. (0 = hub acceleration; 1 = spring length).

- **CompensationFlag**

This activity flag enables the acceleration compensation. If it is set to 1 in addition to the aerodynamic forces, the downforce actuators could be used to introduce the load transfer effects produced by longitudinal and lateral acceleration.

The parameters included in this section are used to setup the path sensor tolerance be used for VI-CarRealTime simulation.

Path Sensor:

tolerances

- **end of path**

specifies the tolerance used for end of path checking (meter).

- **beacon**

specifies the tolerance used for beacon checking (meter).

Output Channels

Output channels can be completely customized in VI-CarRealTime. Each system file contains the reference to an output mapping file (.xml). The file contains the list of all the outputs created by the solver.

When the vehicle model is modified (for example adding more sensors or introducing engine part), also the output channels update accordingly. The Output Channels Map file is not updated automatically, but it is possible to synchronize it to the model using the Update Map button. This procedure overwrite the selected files adding the missing channels (or removing obsolete ones) and maintains the settings of the existing valid channels.

	Active	Result Set	Component	Scale
aero_forces_aero_balance	yes	aero_forces	aero_balance	1.0
aero_forces_center_downforce	yes	aero_forces	center_downforce	1.0
aero_forces_center_sideforce	yes	aero_forces	center_sideforce	1.0
aero_forces_drag_arm	yes	aero_forces	drag_arm	1.0
aero_forces_drag_force	yes	aero_forces	drag_force	1.0
aero_forces_drag_force_front	yes	aero_forces	drag_force_front	1.0
aero_forces_drag_force_rear	yes	aero_forces	drag_force_rear	1.0
aero_forces_drag_torque	yes	aero_forces	drag_torque	1.0
aero_forces_front_downforce	yes	aero_forces	front_downforce	1.0
aero_forces_front_sideforce	yes	aero_forces	front_sideforce	1.0
aero_forces_height_auxiliary	yes	aero_forces	height_auxiliary	1.0
aero_forces_height_front	yes	aero_forces	height_front	1.0
aero_forces_height_rear	yes	aero_forces	height_rear	1.0
aero_forces_lateral_velocity	yes	aero_forces	lateral_velocity	1.0
aero_forces_longitudinal_velocity	yes	aero_forces	longitudinal_velocity	1.0
aero_forces_pitch	yes	aero_forces	pitch	1.0
aero_forces_rear_downforce	yes	aero_forces	rear_downforce	1.0
aero_forces_rear_sideforce	yes	aero_forces	rear_sideforce	1.0
aero_forces_roll	yes	aero_forces	roll	1.0
aero_forces_roll_torque	yes	aero_forces	roll_torque	1.0
aero_forces_side_slip	yes	aero_forces	side_slip	1.0

For each of the available channels it is possible to customize:

- **Activity**
- **Result Set Name**
- **Component Name**
- **Scale Factor**
- **Offset**
- **Live Animation** (setting this flag to "yes" will make the selected channel available during live animation).

The Output Channels File is stored into the database in output_maps.tbl table.

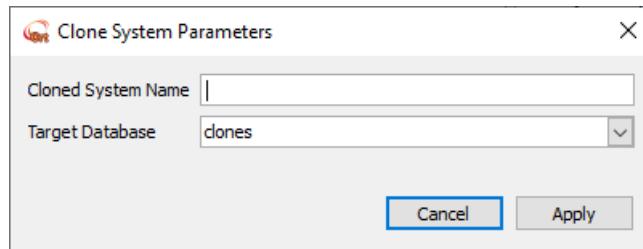
NOTE: Scale Factor and Offset do not affect output data when using the Matlab/Simulink Interface.

Clone Model

When a model is opened in [Build Mode](#), it is possible to clone it by right-clicking on the system in the tree view and selecting *Clone Model*.

The user will be asked to enter:

- **Cloned System Name**
name assigned to the cloned system. The cloned subsystems will use such name as name prefix.
- **Target Database**
specifies the database used in the cloned system. All the databases registered in the session are available; furthermore a database called *clones* is provided as option by the clone tool. Such database does not exist in the disk, but only in VI-CarRealTime session and it is automatically saved in the working directory when the user saves the cloned system.



The cloned model will be created in VI-CarRealTime session.

It is mandatory to specify a target database, whose [alias](#) will be used to identify the relations among system and subsystems and among subsystem and the property files; among the databases available as target databases, the tool provides a generic database having alias=*clones*.

The cloned system can be used in VI-CarRealTime session as a standard system, so it is possible to modify its parameters in [Build Mode](#) and run events in [Test Mode](#).

Note: the difference with respect to a standard system is that such model only exists in VI-CarRealTime session: no file has been saved (yet) in the disk.

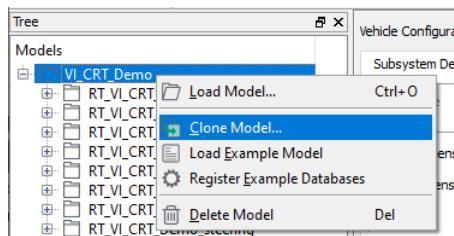
It is possible anyway to save the cloned model by using [Save All](#) as per a standard model. Only now the whole model (system, subsystems, property files) will be saved in the target database.

Due to the concept itself of cloning, not all the property files of the cloned system will be saved in the target database: only the XML property files of the elastic elements will be saved in the target database, whilst the other property files still reference the file in the original model that has been used to clone the system. For all the details look at the example below.

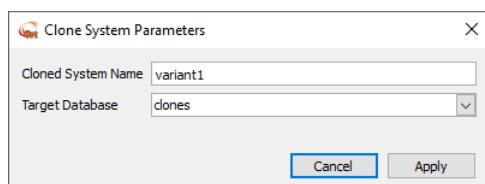
Note: cloning a system and saving it in the target database is not equivalent to [publish](#) the system, since the publish copies all the files in the target database.

Here an example:

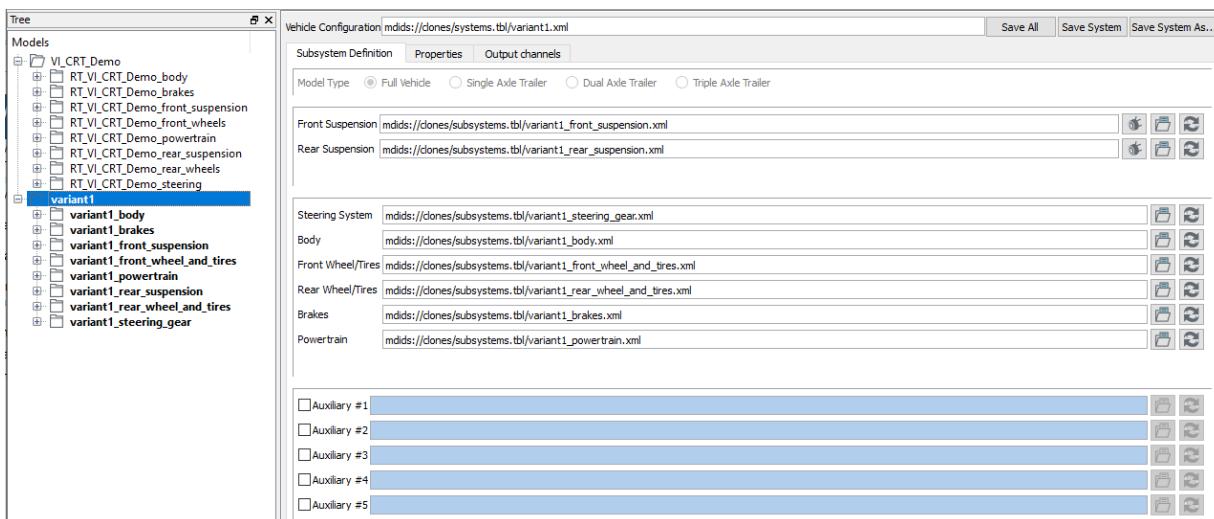
- open a model in session
- right-click on the system and select *Clone Model...*



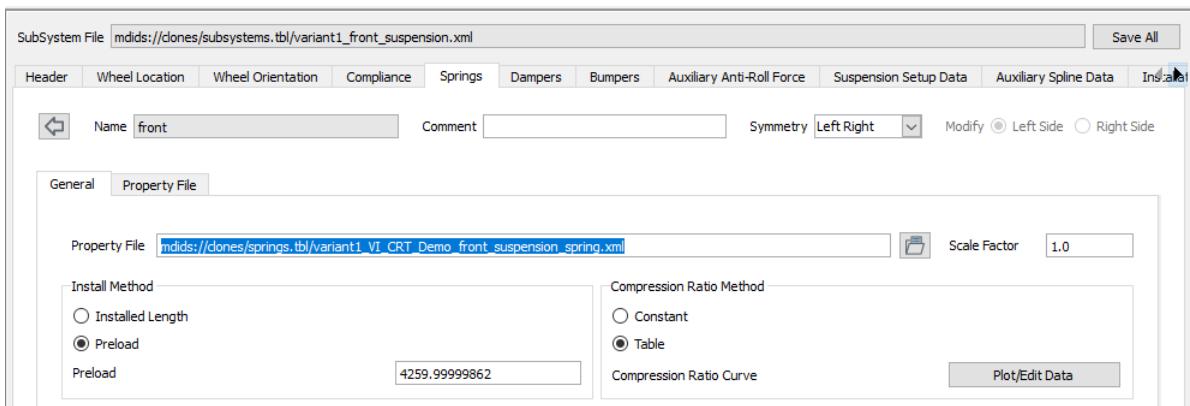
- set a name for the cloned system and use default "clones" as target database:



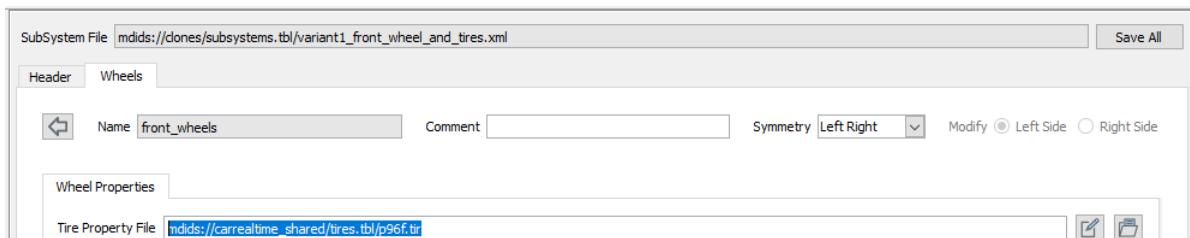
- the cloned system generated will have the system and all the subsystems **bold**, because no file has been yet saved in the target database:



- note that all the subsystems are referenced in the system using the `clones` alias.
- The same happens for the springs/dampers/bumpers property files...



- ...but not for the following property files:
 - [Output map](#) XML in the system
 - [AER](#) file in body subsystem
 - [TIR](#) file in front and rear wheel subsystems
 - [PWR](#) file in powertrain subsystem
 - all graphic files (XGR, OBJ, CMD, IVE) in body subsystem



Suspension Subsystem

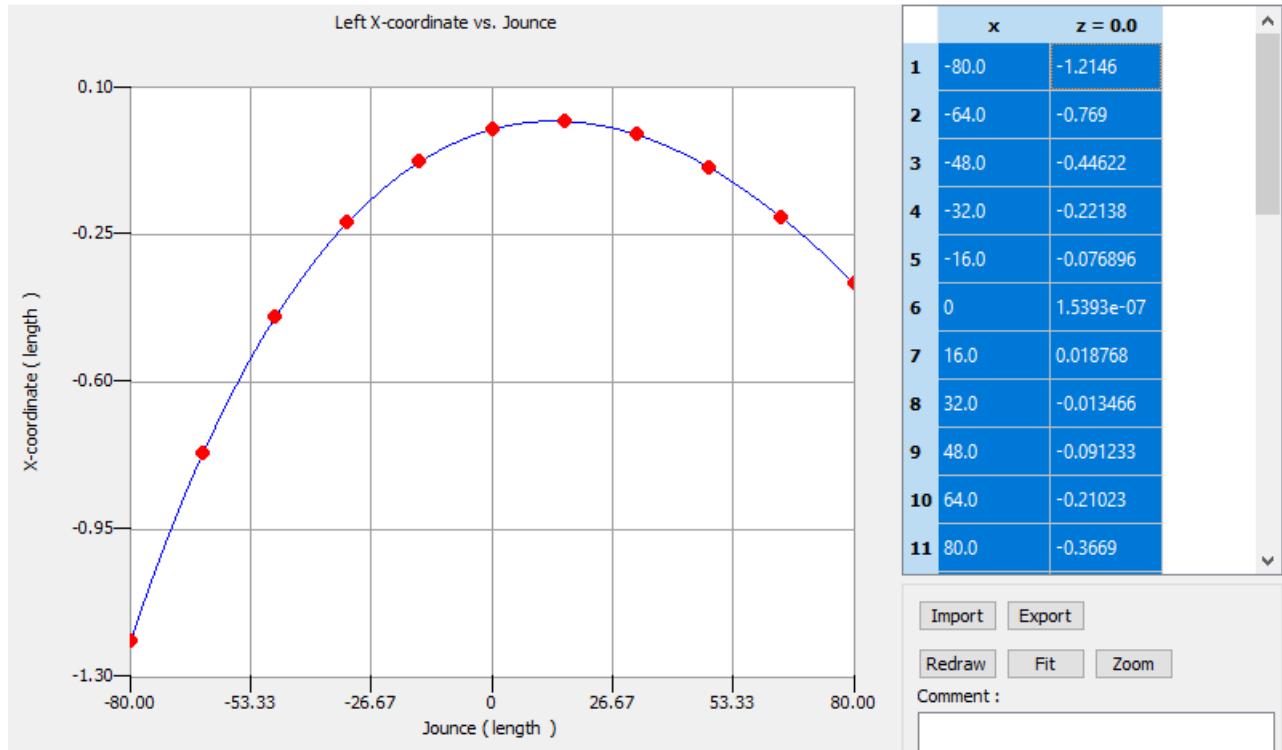
In VI-CarRealTime suspensions subsystems are described using a conceptual approach.

No physical part or linkage is present in the model; the movement of the wheel is related to vertical jounce (independent variable in equations of motion) using a special constraint in order to define the position and orientation (5 DOF) by lookup tables.

For what concerns the kinematic definition, VI-CarRealTime supports two different flavors of suspensions:

- **Independent**

suspension kinematic curves are mapped as a function of the vertical jounce of the same wheel (**2D splines**).



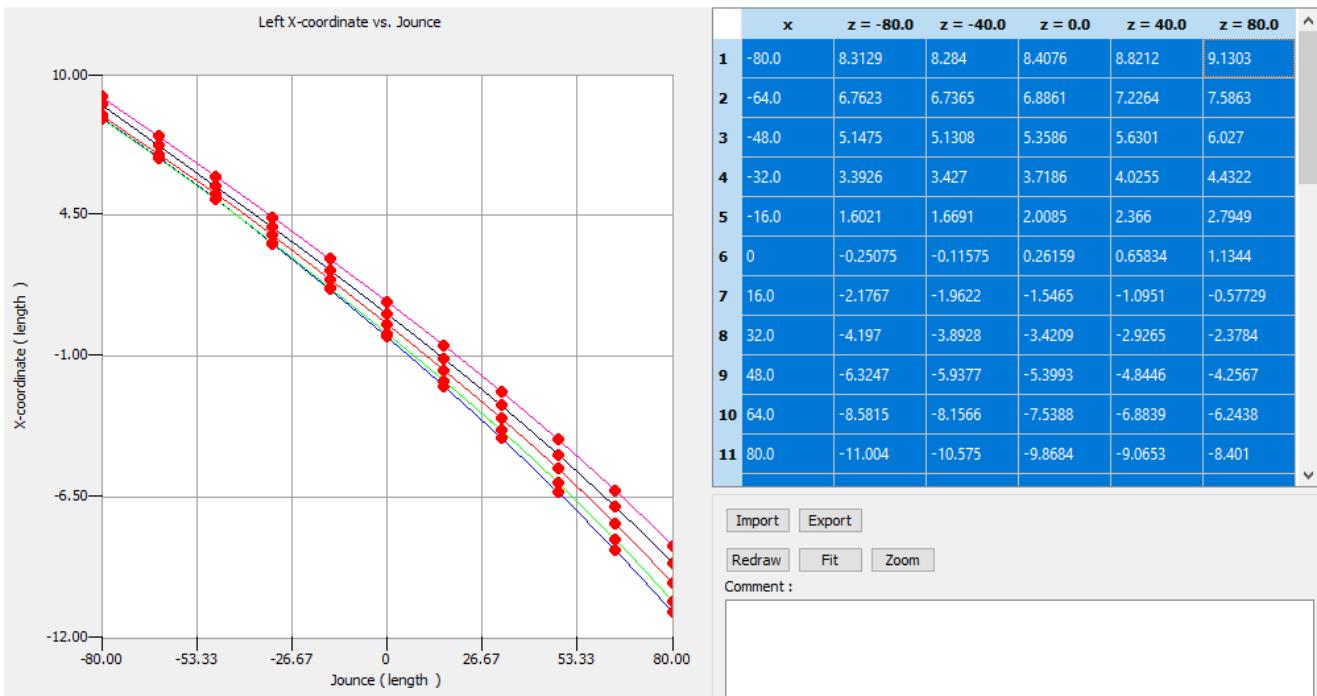
- **Dependent**

suspension kinematic curves are mapped as a function of the vertical jounce of both, left and right, wheels (**3D splines**).

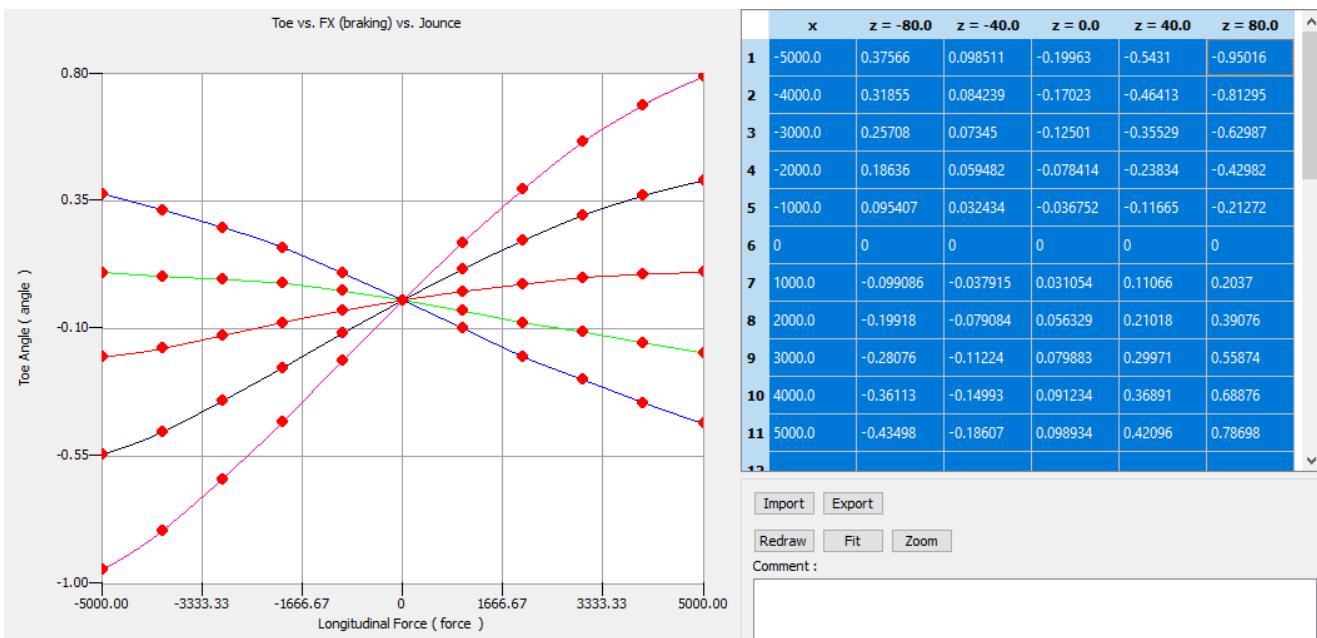
In particular:

- the first independent variable (x) is the jounce of the same wheel
- the second independent variable (z) is the jounce of the paired wheel

VI-CarRealTime

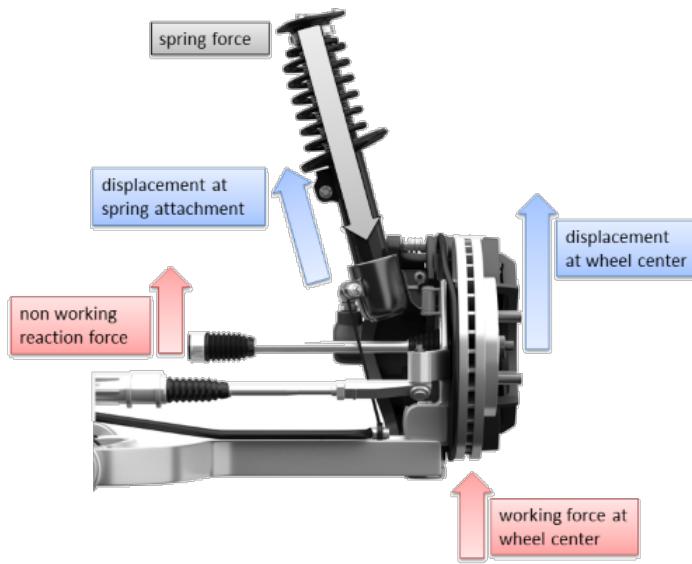


The compliance effect is also taken into account mapping the location (base and track variation) and orientation (toe, camber and side-view angle variation) of the unsprung mass as function of wheel jounce and external load (3D splines).



The effect of suspension components (springs, dampers, bumpers etc.) is projected onto wheels and computed by:

- applying motion ratio from wheel to suspension component travel;
- use component lookup tables to get element forces;
- use motion ratio to compute equivalent force at wheel center.



The following panels are available in the Suspension Subsystem:

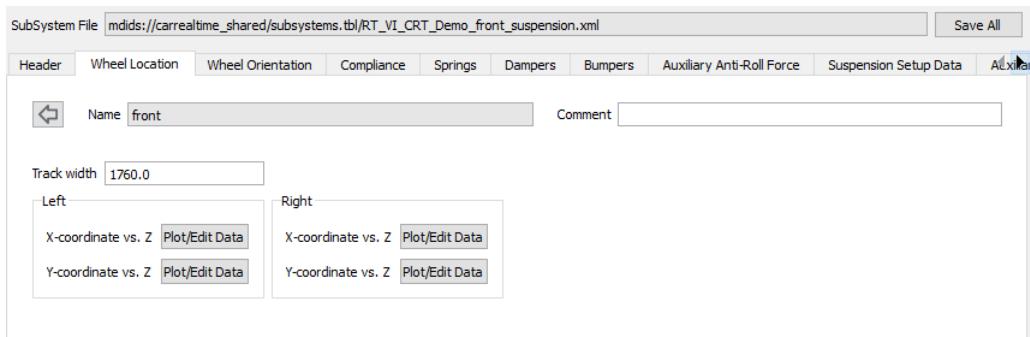
- [Wheel location](#)
- [Wheel Orientation](#)
- [Compliance](#)
- [Springs](#)
- [Dampers](#)
- [Bumpers](#)
- [Auxiliary Anti Roll Force](#)
- [Anti-Roll Bar](#)
- [Auxiliary Vertical Force](#)
- [Suspension Setup Data](#)
- [Installation Stiffness](#)
- [Auxiliary Translational DOF](#)

Wheel Location

Wheel location is described using the following data:

- **Track width**
the distance among left/right wheel centers at design time (Y direction of global reference system).
- **Wheel center X-coordinate vs Z (Left/Right)**
wheel base variation in wheel location reference system as a function of wheel jounce.
- **Wheel center Y-coordinate vs Z(Left/Right)**
wheel track variation in wheel location reference system as a function of wheel jounce.

VI-CarRealTime



The track and base variation are described by spline data editable via the **Curve Editor**. Curve data can be imported or exported from/to external ascii file using the specific push buttons.

Each curve has wheel jounce as an independent variable (X).

For [dependent suspensions](#) (**only rear**) an extra independent variable (Z) inputs the paired wheel jounce.

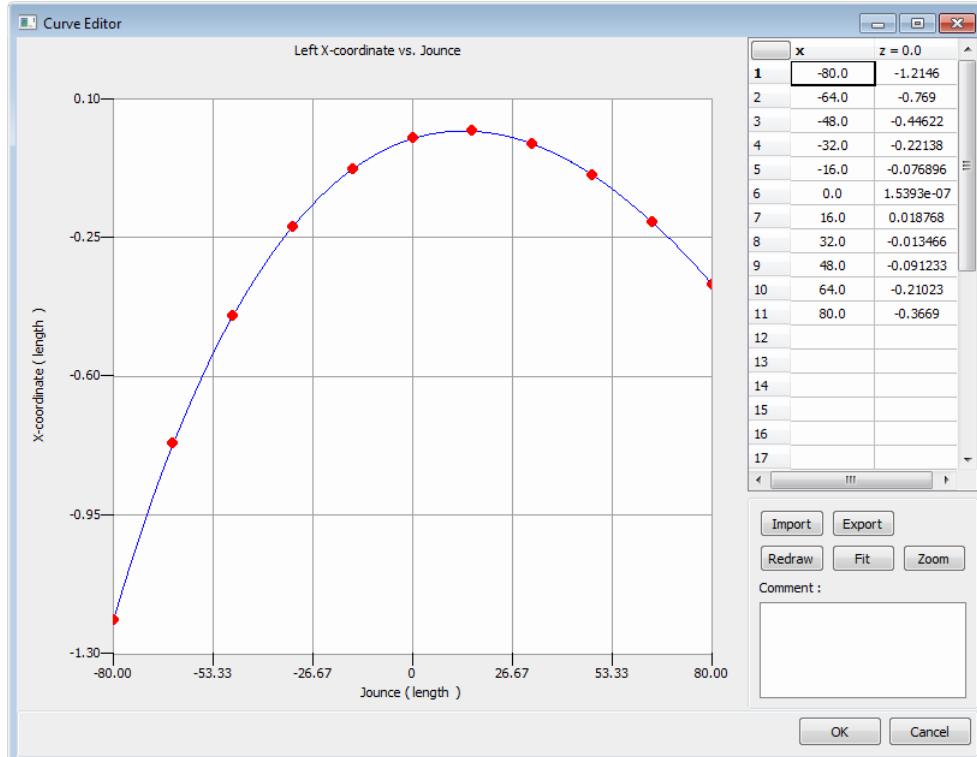
The **Wheel Location Reference System** is defined by a triad positioned at wheel center (left and right, at design time); the orientation is the same of VI-CarRealTime global reference system with X+ forward, Z+ vertical up, Y+ left.

For **front suspensions** the track and base variation are set as depending on steering wheel angle and jounce. A specific set of curve data is present in the [steering subsystem](#) file.

In that case the curve data in the Wheel Location editor:

- are not used by VI-CarRealTime solver if the front suspension curves are defined as [independent](#);
- are automatically summed up by VI-CarRealTime solver if the front suspension curves are defined as [dependent](#).

Note: the same concept also applied for [Wheel Orientation](#) panel.



Wheel Orientation

Wheel orientation in VI-CarRealTime is described using the following data:

- **Side View Angle**

is the [variation](#) of the wheel carrier side-view rotation angle with respect to the caster angle at design time. It is positive for a clockwise rotation, as seen from the left side of the vehicle.

For front suspensions, this spline is used only when it is mapped as a 3D spline [$sva = f(jounce_L, jounce_R)$], otherwise the [Side View Angle vs Input Steer vs Jounce](#) spline is used.

- **Toe Angle**

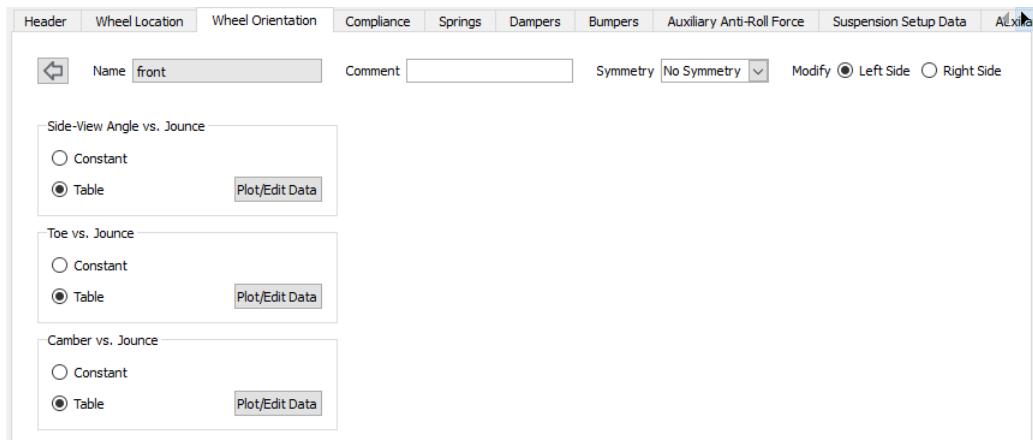
is the angle between the longitudinal axis of the vehicle and the line of intersection of the wheel plane and the vehicle's XY plane. It is positive if the wheel front is rotated in towards the vehicle body.

For front suspensions, this spline is used only when the [Use Steer input and Toe vs jounce Tables](#) radio button is checked in [Kinematic](#) panel of the steering subsystem.

- **Camber Angle**

is the angle the wheel plane has with respect to the vehicle's vertical axis. It is positive when the top of the wheel leans outward from the vehicle body.

For front suspensions, this spline is used only when it is mapped as a 3D spline [$camber = f(jounce_L, jounce_R)$], otherwise the [Camber Angle vs Input Steer vs Jounce](#) spline is used.

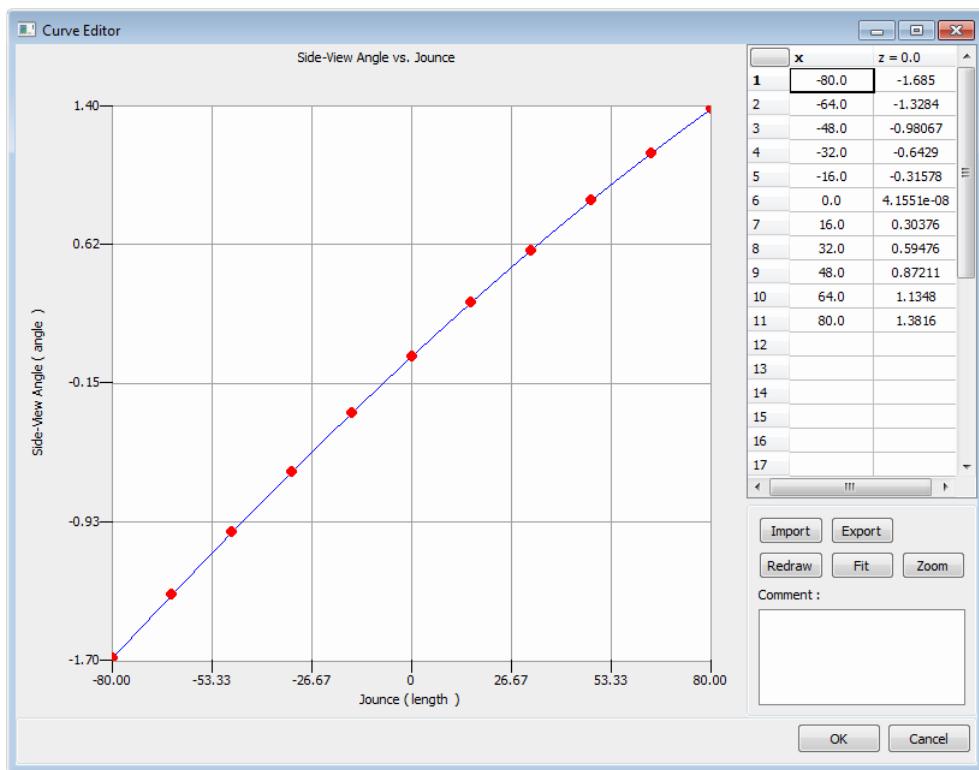


For each of the properties above the user can specify the dependency on suspension jounce as **constant rate** or **table data**:

- **Constant** value is set by the **Property Editor**.
- **Table** data can be input/edited using the **Curve Editor**. Each curve has wheel jounce as independent variable (X). For dependent suspensions (**only rear**) an extra independent variable (Z) input the paired wheel jounce.

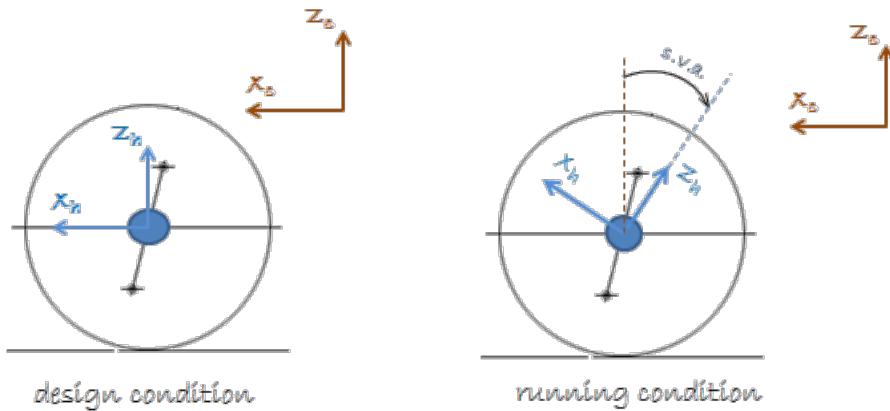
Symmetry

Wheel orientation includes symmetry option; the user can select it through the option menu in the Property Editor. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.



Note: The inclination angle, a measurement available in full-vehicle analyses, is the angle the wheel plane makes with respect to the road surface. The inclination angle is used for tire calculations.

The figure below illustrates the meaning of the side view angle (s.v.a.) spline.



The s.v.a. vs. Jounce spline measures the variation of the wheel carrier side-view rotation angle with respect to the caster angle at design time. In design condition the reference system of the Hub (blue part in figure) has the same orientation as the chassis reference (z_h is parallel to z_c), so the side view angle is 0; with respect to this design configuration, the side view angle changes as a function of the suspension jounce and of the steering wheel angle (**Side View Angle vs Jounce** and **Side View Angle vs Input Steer vs Jounce** splines).

More details on the **Side View Angle** can be found in the *Output Description* paragraph of the VI-SuspensionGen documentation.

The derivative of the **Side View Angle vs Jounce** curve is used to evaluate the anti-dive and anti-squat behaviour of the suspension.

Such a spline is not used by VI-CarRealTime solver for suspension kinematic computation; in fact the caster angle does not affect the kinematic behaviour of a suspension, it changes only the steering axis geometry (kingpin) and this phenomenon is characterized by the splines defined in the [kingpin panel](#) of the [steering subsystem](#).

Compliance

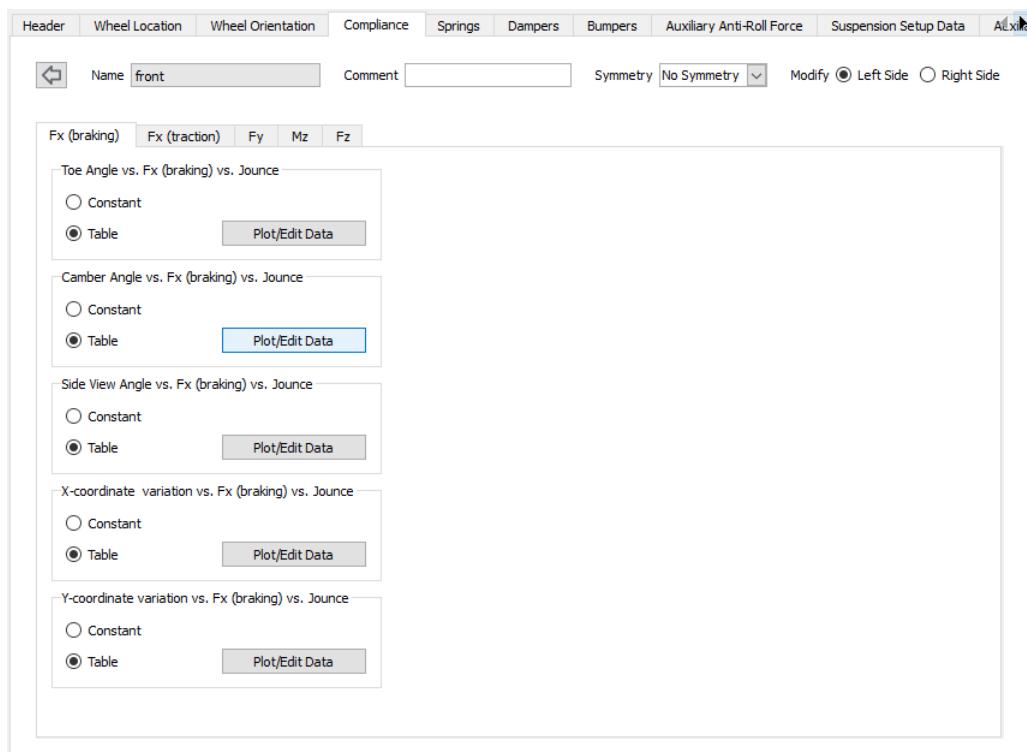
In VI-CarRealTime the wheel orientation is implemented using the effect superposition principle.

Given that the kinematic position and orientation being defined by the dependency on suspension jounce (left and right for dependent rear suspensions) and eventually steering wheel angle (for front suspensions), the effect of external loads on a given suspension property (track or base variation, toe, camber, side view angle) is computed as the sum of the effect given by all active loads taken singularly.

$$\varepsilon = \varepsilon_k + \sum_{i=1}^n \varepsilon_f$$

where:

- ε is the total value of a suspension property (toe angle, side view angle etc.).
- ε_k is the value of the suspension property given by kinematic dependencies (jounce, steering wheel angle).
- ε_f is the value of the suspension property **variation** given by a specific load (F_x , F_y , T_z etc.).



VI-CarRealTime vehicle model includes the dependency of the following data on external loads:

- **Toe Angle**
- **Camber Angle**
- **Side View Angle**
- **Wheel Center Y-coordinate Variation**

- **Wheel Center X-coordinate Variation**

The following external loads are considered as inputs:

- **F_x (braking)** - [ISO-W](#)
- **F_x (traction)** - [ISO-C](#)
- **F_y** - [ISO-W](#)
- **M_z** - [ISO-W](#)
- **F_z** - [ISO-W](#)

Note: for each wheel the longitudinal compliance switch between *F_x (traction)* spline and *F_x (braking)* spline is controlled by the braking moment; *F_x (traction)* spline are used until a braking moment is applied on the wheel.

The table below summarizes the compliance dependencies for a single corner.

Entity	Dependencies				
Toe Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Camber Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Side View Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Y-coordinate Variation	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
X-coordinate Variation	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z

Cross compliances

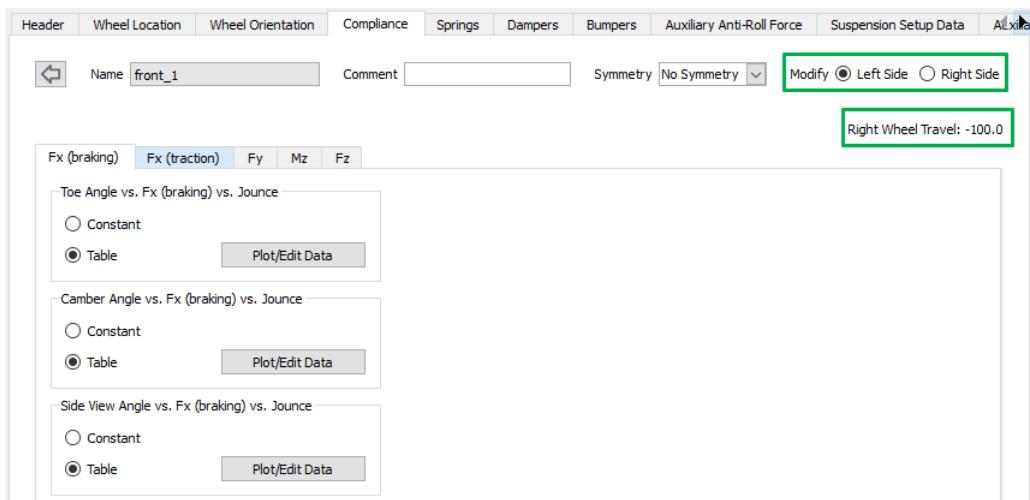
A VI-CarRealTime vehicle model also includes the dependency of the compliance of a wheel given by a force applied on the paired wheel. An equivalent set of data is input for direct and cross compliance (with the exclusion of FZ cross dependency).

For each of the properties above the user can specify the dependency on suspension jounce as **constant rate** or **table data**:

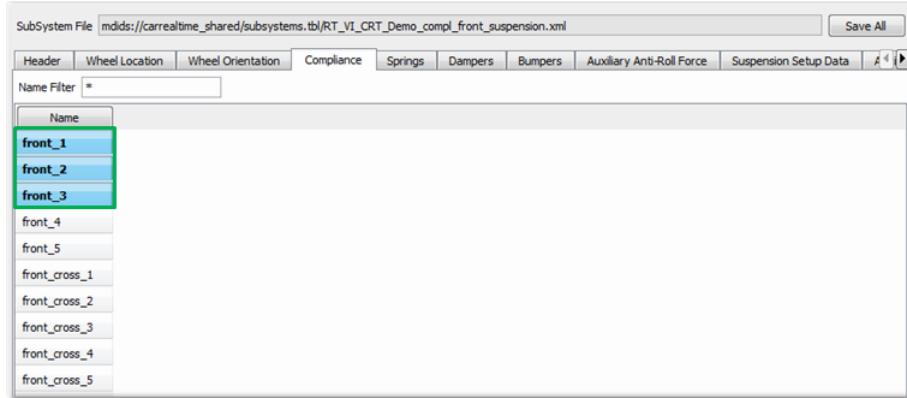
- **Constant** value is set by the **Property Editor**;
- **Table** data can be input/edited using the **Curve Editor**. Each curve has an external load as independent variable (X). It is also possible to specify a dependency on wheel jounce by an extra independent variable (Z) in the **table data** (exception for F_z which is a 2D spline).

In VI-CarRealTime there are two possible options regarding the number of compliance loadcases:

1. One direct compliance and one cross compliance set: in this case the compliance dependency of external force is expressed at different wheel travel levels (3D table columns), common between left and right wheel;
2. Multiple sets of direct and cross compliance: in this case each compliance set is corresponding to curves representing the dependency of suspension states on applied loads and wheel travel for a single wheel; the other wheel is kept at a fixed value which is shown in the VI-CarRealTime graphical user interface below the side radio box (top right of compliance editor). VI-CarRealTime is performing a multi-dimensional interpolation among all the available sets as a function of: applied force, left wheel travel, right wheel travel.



Example: the case of three sets of direct compliance curves for left and right works as follows:



Left compliance

1. the jounce of the *right* suspension is set to the **lowest** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
2. the jounce of the *right* suspension is set to the **intermediate** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
3. the jounce of the *right* suspension is set to the **highest** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.

Right compliance

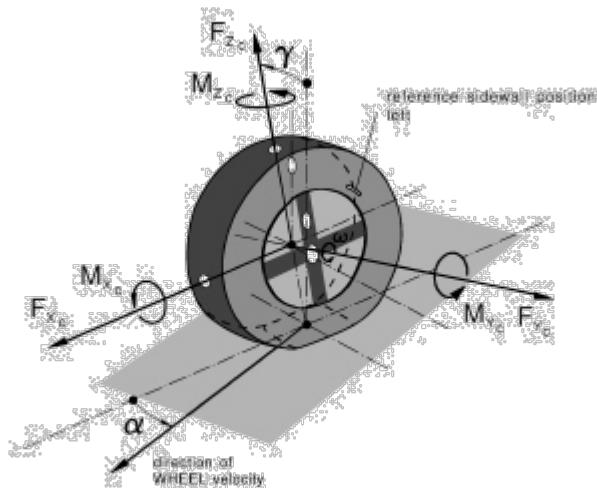
1. the jounce of the *left* suspension is set to the **lowest** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
2. the jounce of the *left* suspension is set to the **intermediate** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
3. the jounce of the *left* suspension is set to the **highest** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.

Compliance Reference Systems:

$F_{x(\text{traction})}$ forces are applied at **ISO-C Axis System**.

- ISO-C Axis System:

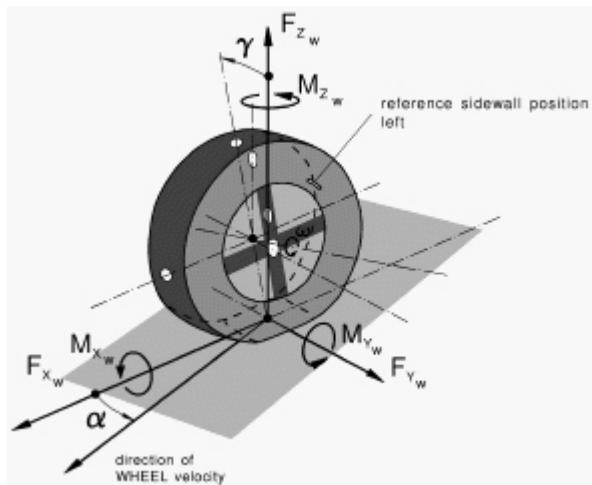
- The origin of the ISO-C axis system lies at the wheel center.
- The **+ x-axis** is parallel to the road and lies in the wheel plane.
- The **+ y-axis** is normal to the wheel plane and, therefore, parallel to the wheel's spin axis.
- The **+ z-axis** lies in the wheel plane and is perpendicular to x and y (such as $z = x \times y$).



$F_{x(\text{braking})}$ F_y T_z and F_z forces are applied at **ISO-W Axis System**.

- ISO-W Contact-Patch Axis System:

- The origin of the ISO-W contact-patch system lies in the local road plane at the tire contact point.
- The **+ x-axis** lies in the local road plane along the intersection of the wheel plane and the local road plane.
- The **+ z-axis** is perpendicular (normal) to the local road plane and points upward.
- The **+ y-axis** lies in the local road plane and is perpendicular to the + x-axis and + z-axis (such as $y = z \times x$).



Suspension properties are measured using the same convention adopted for wheel location and wheel orientation for suspension kinematics.

Note: a compliance curve is actually used by the solver only if the related activity flag has been set to on in [Suspension Data Compliance](#) panel.

Springs

The Spring is the main elastic component in the suspension.

Two types of springs are available in VI-CarRealTime:

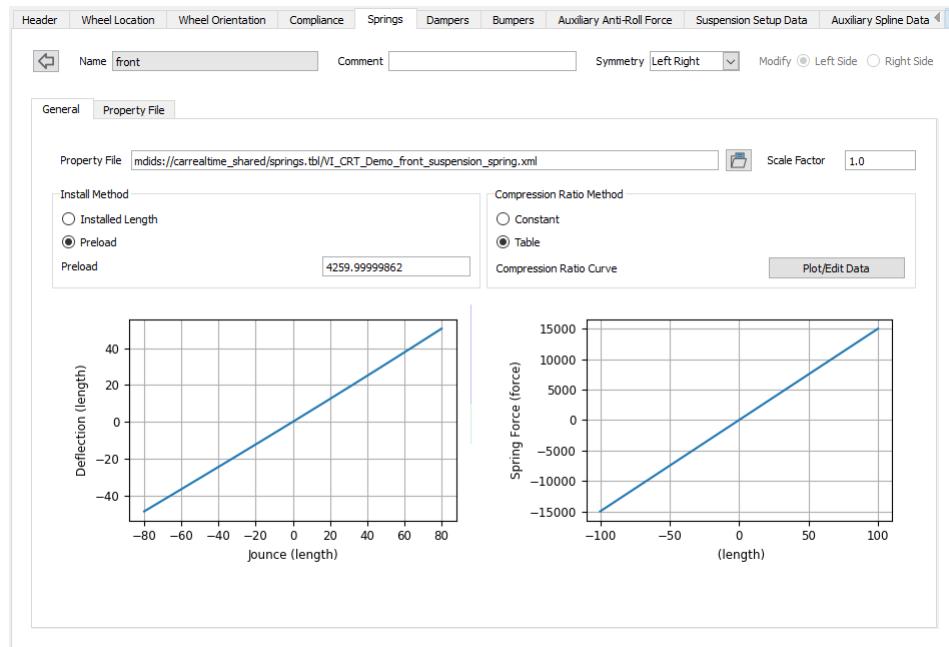
- [Coil springs](#)
- [Air springs](#)

The component properties of a coil spring are described by:

- **General panel**
used to project at wheel center the effect of spring component:

- [Install method](#)
- [Compression ratio method](#)

- **Property File panel**
used to describe component specific properties.



Suspension subsystem spring data:

Installation method

There are two possible installation methods for springs that can be selected using the Property Editor:

- **Installed Length**

This parameter describes the length of the spring at design time; the preload is computed depending on spring free length.

- **Preload**

It is the force acting at spring ends at design time.

Depending on the chosen method, Preload or Installed Length files display.

Installed Length and Preload are two different approaches for defining the same thing, that is how the spring is mounted on the vehicle. VI-CarRealTime internal spring always use the installed length, so if preload is set, the corresponding installed length is computed based on spring characteristics. If preload is specified rather than the installed length, then the installed length must be calculated by inverse table lookup. The value of the force when the spring compression is equal to (free length – installed length) should equal the preload.

Note: being Preload and Installed length parameters referring to the same xml file entry (InstallValue) and being the units variable depending on the type of spring (Translational or Rotational) and on the Installation Method chosen, the field units will not be shown. It is assumed that they are entered in a unit system congruent with the other subsystem data.

Compression ratio method

It is used to convert the spring component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

Value is set by **Property Editor** and expresses the ratio between the wheel jounce and component deflection.

- **Table**

Data can be input/edited using the **Curve Editor**. The curve has wheel jounce as independent variable (X) and expresses the spring deflection(Y).

Property File

The spring xml [property file](#) is read and the data are used to fill up the Properties panel. The user can modify the spring element characteristics and then save the modification in the spring property file.

- **Scale Factor**

scaling factor which multiplies the spring characteristic curve.

Method

- **Linear**

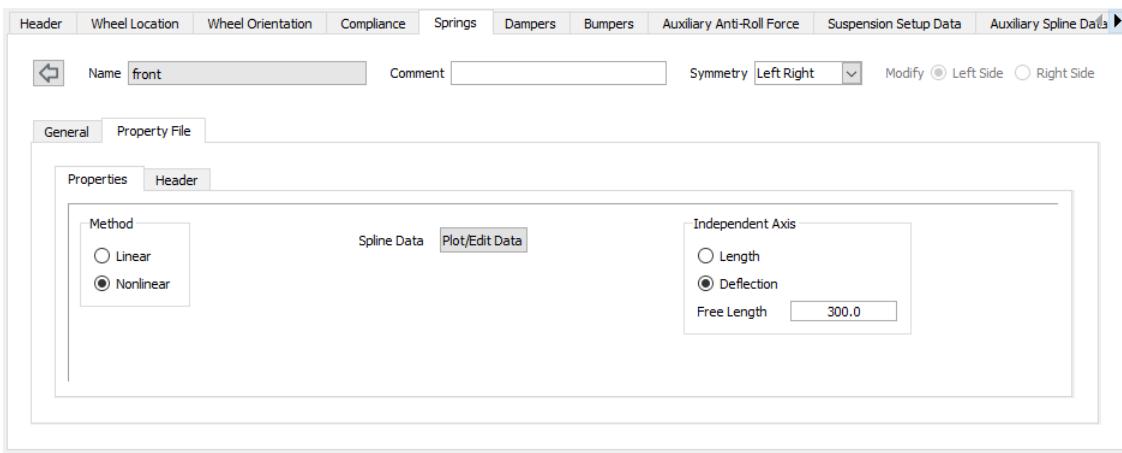
Allows to define a linear spring property file characteristic. When such toggle is selected the user is allowed to define the spring Free Length and the spring Linear Rate.

- **Nonlinear**

Allows to define a non-linear spring property file characteristic. When such toggle is selected the user is allowed to define the Spline Data, that is the spline characteristic curve. Furthermore the Independent Axis type must be chosen, between:

- **Length:** it is the "real" length of the spring, the distance between the spring ends. Since in VI-CarRealTime the spring is conceptual, the term "length" loses its intrinsic meaning: the spring length in VI-CarRealTime is 0 in design condition, so when *Length* is used as independent axis, the Spline Data must be entered considering spring length at design equals to 0.

- **Deflection:** it is the spring deformation with respect to its design length. When *Deflection* is selected as Independent Axis, also the *Free Length* parameter must be entered in order to properly define the spring characteristic.



Symmetry

Spring properties include symmetry option; the user can select it inside the option menu using the Property Editor. The GUI also features the capability of defining which side the properties will belong to. The effect of the selection is to copy the data content of the selected side into the paired one.

VI-CarRealTime supports both rotational and translational spring properties.

Available methods for linear spring property file data are:

- **Linear**

user will specify linear rate and spring free length.

- **Non Linear**

spring data will be input by a curve editable using the Curve Editor; other data needed are the independent axis in spline data (Length or Deflection) and Free Length.

For rotational springs the available parameter is the linear torsional stiffness.

The component properties of an air spring are described by:

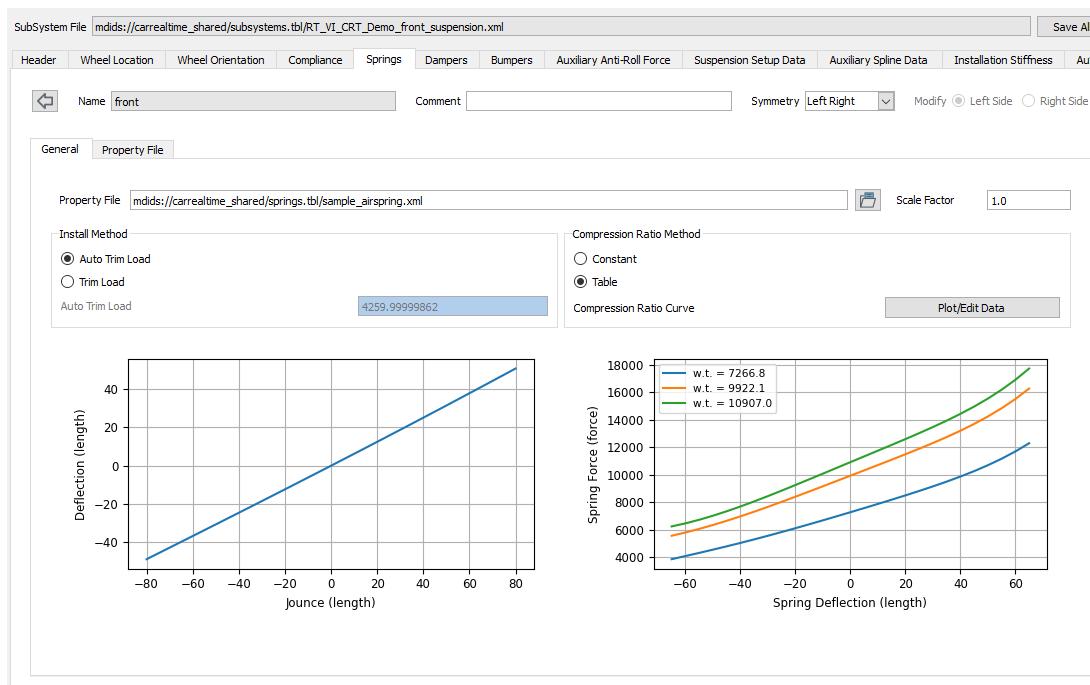
- **General panel**

used to project at wheel center the effect of air spring component:

- [Install method](#)
- [Compression ratio method](#)

- **Property File panel**

used to describe component specific properties.



Suspension subsystem air spring data:

Install method

There are two possible installation methods for air springs that can be selected using the Property Editor:

- **Auto Trim Load**

A PID controller is used during the static analysis in order to have at the static equilibrium an air spring deflection equals to 0 (that is an air spring length equals to the design time length).

- **Trim Load**

It is the force in the air spring when the suspension is at the trim height.

Compression ratio method

It is used to convert the air spring component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

Value is set by **Property Editor** and expresses the ratio between the wheel jounce and component deflection.

- **Table**

Data can be input/edited using the **Curve Editor**. The curve has wheel jounce as independent variable (X) and expresses the spring deflection(Y).

Property File

It sets the path of the property file containing component data. The air spring component data can be edited in the Property File tab.

- **Scale Factor**

scaling factor which multiplies the spring characteristic curve.

Symmetry

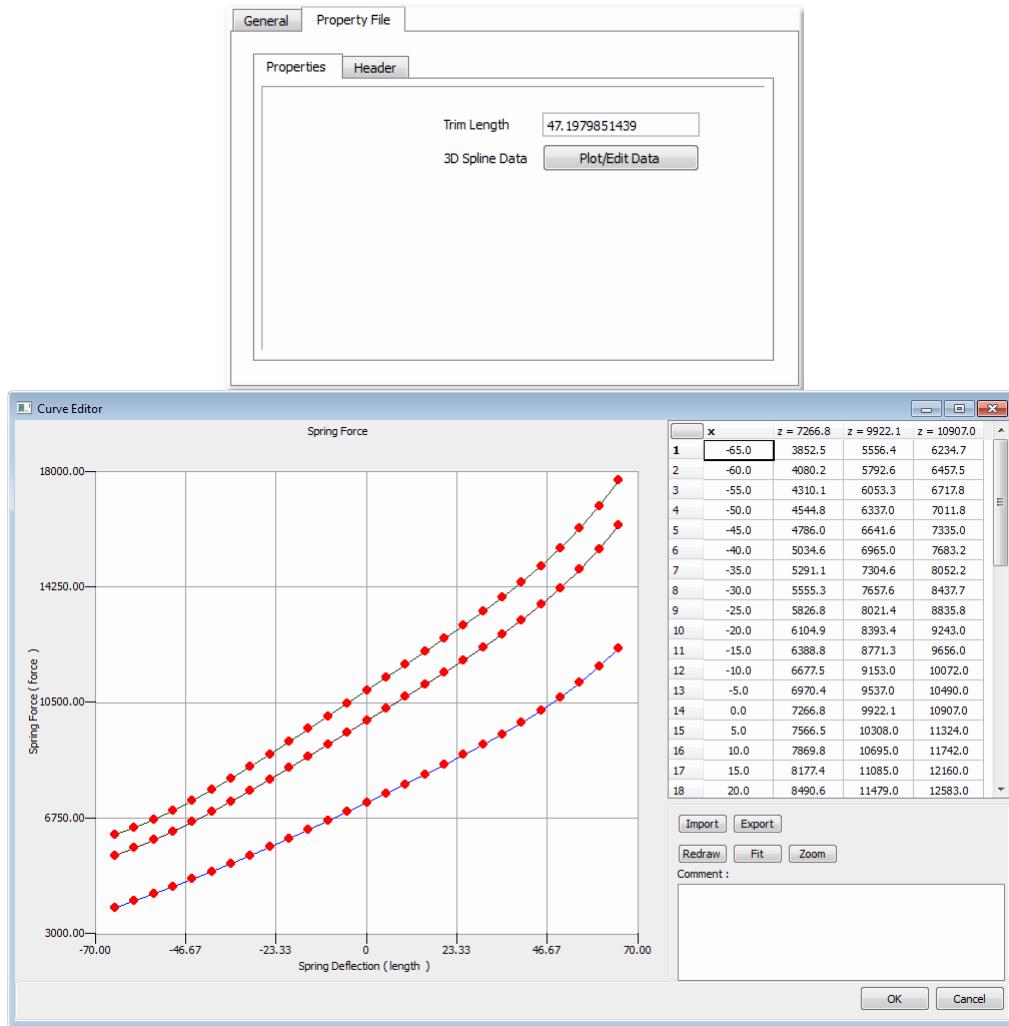
Spring properties include symmetry option; the user can select it inside the option menu using the Property Editor. The GUI also features the capability of defining which side the properties will belong to. The effect of the selection is to copy the data content of the selected side into the paired one.

Property File data:

The air spring property file is defined as a 3D spline [$Y = f(X, Z)$] where:

- X is the air spring deflection;
- Z is the load in the air spring when the suspension is at trim height. The load corresponds to the load defined in the [Install Method](#) panel.

The **Trim Length** is defined as the difference between the length of the air spring when the suspension is at trim height and its design time length. Spring design time length is 0 in VI-CarRealTime model.



The air spring force is computed as:

$$F_{airspring} = SPLINE(trimLength - deflection, trimLoad)$$

Dampers

The Damper is the main elastic component in the suspension.

In VI-CarRealTime the component properties are described by:

- **Suspension data**

used to project the effect of damper component at wheel center:

- [Compression ratio method](#)

- **Property File**

used to describe component specific properties.



Suspension subsystem damper data:

Compression ratio method

It is used to convert damper component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

the value is set by the **Property Editor** and expresses the ratio between the wheel jounce and the component deflection.

- **Table**

data can be input/edited using the **Curve Editor**. The curve has the wheel jounce as an independent variable (X) and expresses the damper deflection(Y).

Property File

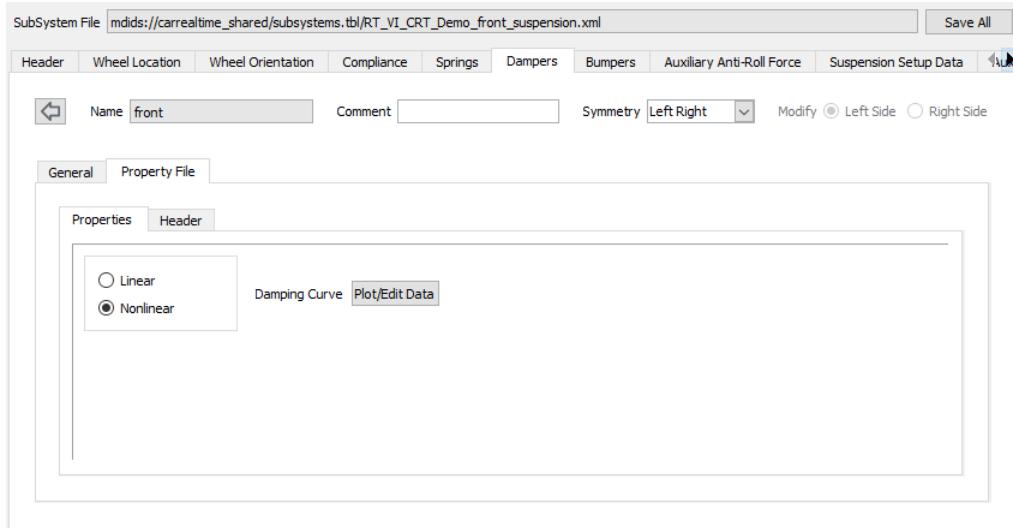
The Suspension data Damper Property Editor shows the path of the property file containing component data. Damper component data can be edited in a separate tab.

- **Scale Factor**

scaling factor which multiplies the damper characteristic curve.

Symmetry

Damper properties include symmetry option; the user can select it by the option menu in the **Property Editor**. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.



Available methods for property file damping are:

- **Linear**

the user has to specify the linear rate.

- **Non Linear**

damper data will be input by a curve editable using the Curve Editor. VI-CarRealTime solver supports both 2D damper and 3D damper curves. 2D damper curve defines the damper element force as a function of [damper velocity](#) (positive in extension); 3D damper curve defines the damper element force as a function of [damper velocity](#) (first independent variable, positive in extension) and [damper displacement](#) (second independent variable, positive in extension). Damper displacement is the damper deflection with respect to the design condition (0 damper displacement). An example of 3D damper property file is shipped with carrealtime_shared database: [sample_3D_damper.xml](#).

VI-CarRealTime supports the presence of an auxiliary damper in parallel with the main damper. The auxiliary damper is automatically created in suspension subsystem during model export from Adams/Car if [Auxiliary Damper](#) flag is activated.

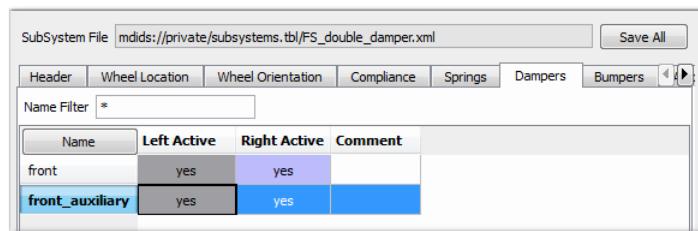
When the auxiliary damper instance is present in a subsystem, further [outputs](#) will be generated in order to review auxiliary damper force, velocity and displacement.

To manually add an auxiliary damper in a suspension subsystem:

- edit suspension subsystem xml file in a text editor;
- copy the entire <DamperPair> block and paste it right below the end of it;
- change the damper name (*name* parameter);
- save the subsystem file.

Dampers panel will now contain two damper instances:

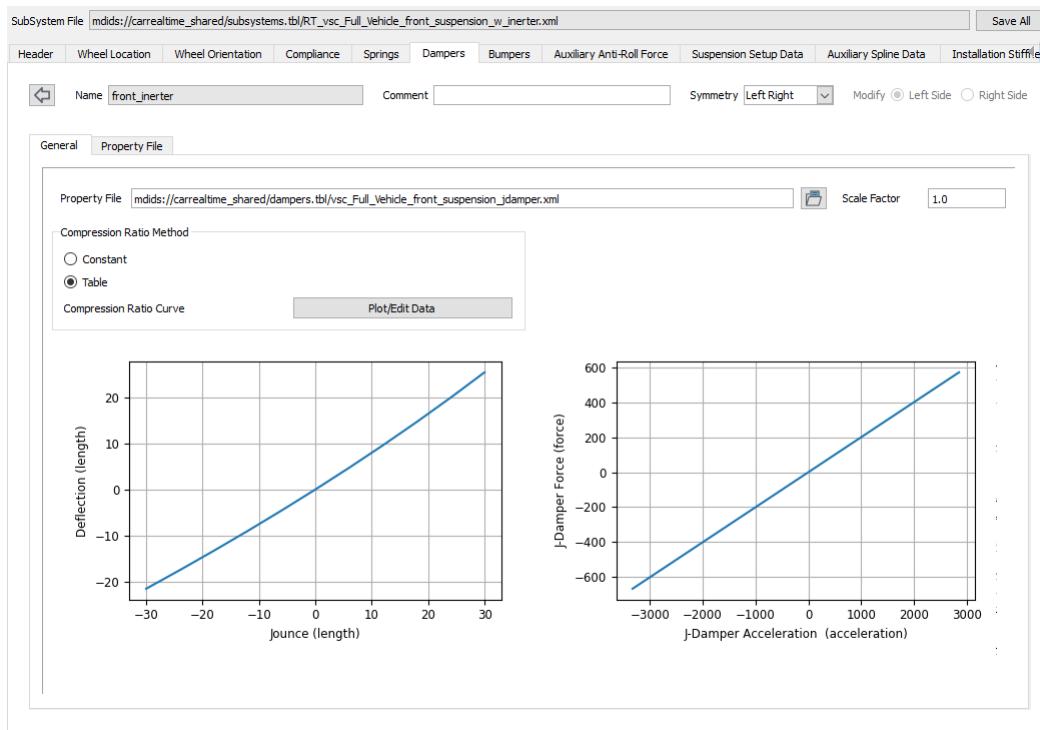
VI-CarRealTime



The VI-CarRealTime model may also include the presence of a third central damper (often present in some race suspension architecture). The GUI will show an appropriate editor with the same features of main dampers. Third damper instance is going to be automatically created when exporting the model from Adams/Car when the original full model includes it or when exporting the model from VI-SuspensionGen using a suspension template supporting the central damper. In the other cases the user will have to add it manually to the suspension xml file. An example of a vehicle including a third damper can be found in VI-CarRealTime shared database (\vsc_Full_Vehicle).

J-Dampers

The VI-CarRealTime model supports the presence of J-damper or *inerters*.



The element is similar to a standard damper having the force proportional to the relative acceleration rather than velocity.

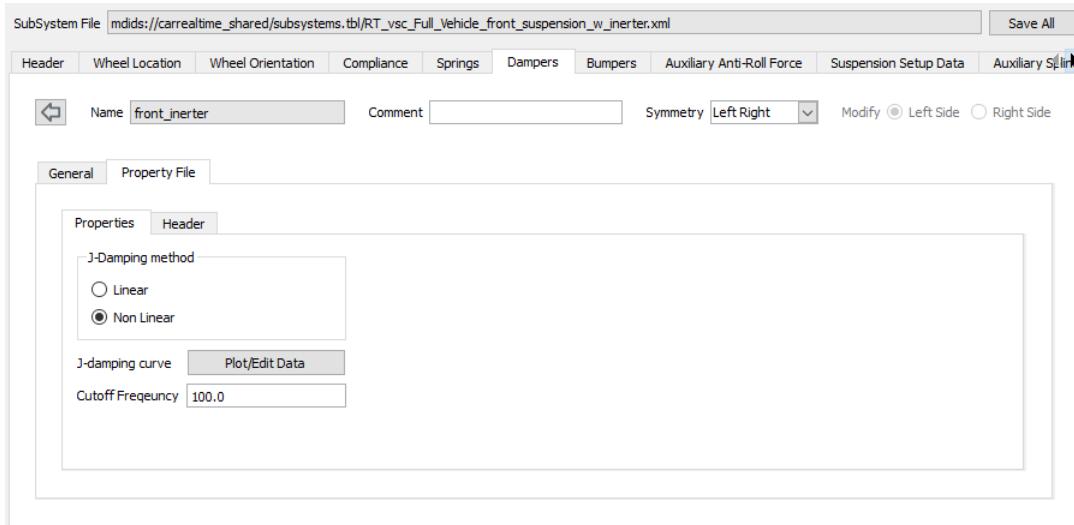
J-Dampers support a specific property file and include the following methods:

- **Linear**
the user has to specify the linear rate.
- **Non Linear**
damper data will be input by a curve editable using the Curve Editor.

This element expose an additional parameter:

- **Cutoff Frequency**

It is the cutoff frequency of the low-pass filter used on the input acceleration to avoid numerical issues during integration.



Note: an example of front suspension subsystem with j-damper element (*RT_vsc_Full_Vehicle_front_suspension_w_inerter.xml*) is shipped with carrealtime_shared database.

Bumpers

Bumper elements are used to limit suspension jounce.

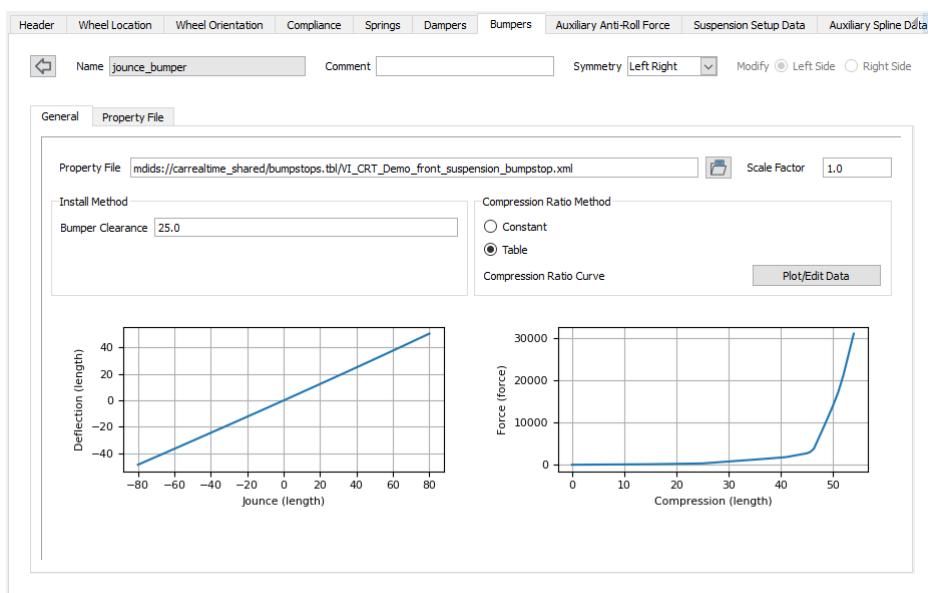
In each VI-CarRealTime suspension model there are two pairs of instances of bumpers, one for jounce and the other for rebound stop.

Header	Wheel Location	Wheel Orientation	Compliance	Springs	Dampers	Bumpers	Auxiliary Anti-Roll Force	Suspension Setup Data	A []
<input type="text" value="Name Filter *"/>									
<input type="text" value=""/>									
Name	Left Active	Right Active	Comment						
rebound_bumper	yes	yes							
jounce_bumper	yes	yes							

The bumper properties are described by:

- **Suspension data**, used to project at wheel center the effect of spring component:
 - Bumper clearance;
 - Compression ratio method.
- **Property File**, used to describe component specific properties.

VI-CarRealTime



Suspension subsystem spring data:

Compression ratio method

It is used to convert the bumper component deformation to an equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

value is set by **Property Editor** and express the ratio between the wheel jounce and component deflection.

- **Table**

data can be input/edited using the **Curve Editor**. The curve has wheel jounce as an independent variable (X) and expresses the bumper deflection (Y).

Bumper Clearance

Describes the deformation that the component must undergo before engaging and applying a force. For models including suspension adjustments the value refers to conditions after the setup, in the other cases to design conditions.

Property File

It sets the path of the property file containing component data. Bumper component data can be edited in a separate tab.

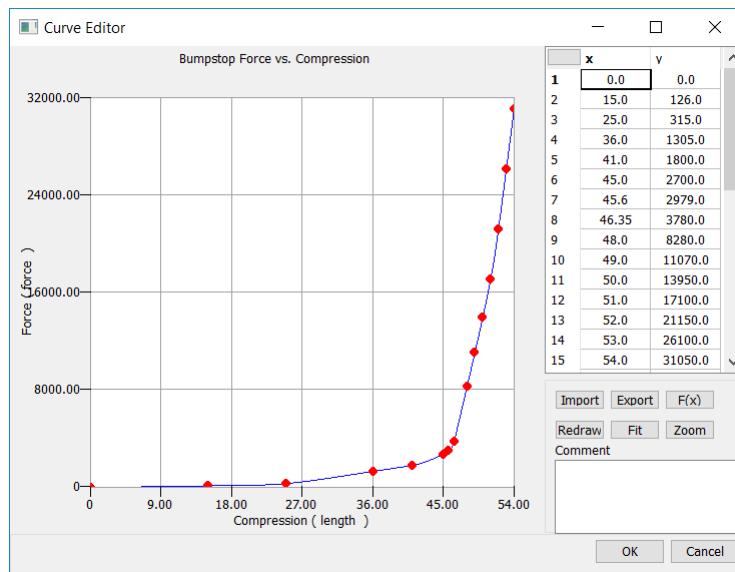
- **Scale Factor**

scaling factor which multiplies the bumper characteristic curve.

Symmetry

Bumper properties include symmetry option; the user can select it via the option menu in Property Editor. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.

Property File data will be input by a curve editable using the Curve Editor, expressing the component force as a function of deformation after engagement.



The VI-CarRealTime model may also include the presence of a third central elastic element (often present in some race suspension architecture). The GUI will show an appropriate editor with the same features of paired bumpers (third_bumper). Third bumper instance is going to be automatically created when exporting the model from Adams Car when the original full model includes it. In the other cases user will have to add it manually to suspension xml file. An example of a vehicle including third bumper can be found in VI-CarRealTime shared database (\vsc_Full_Vehicle).

Auxiliary Anti Roll Force

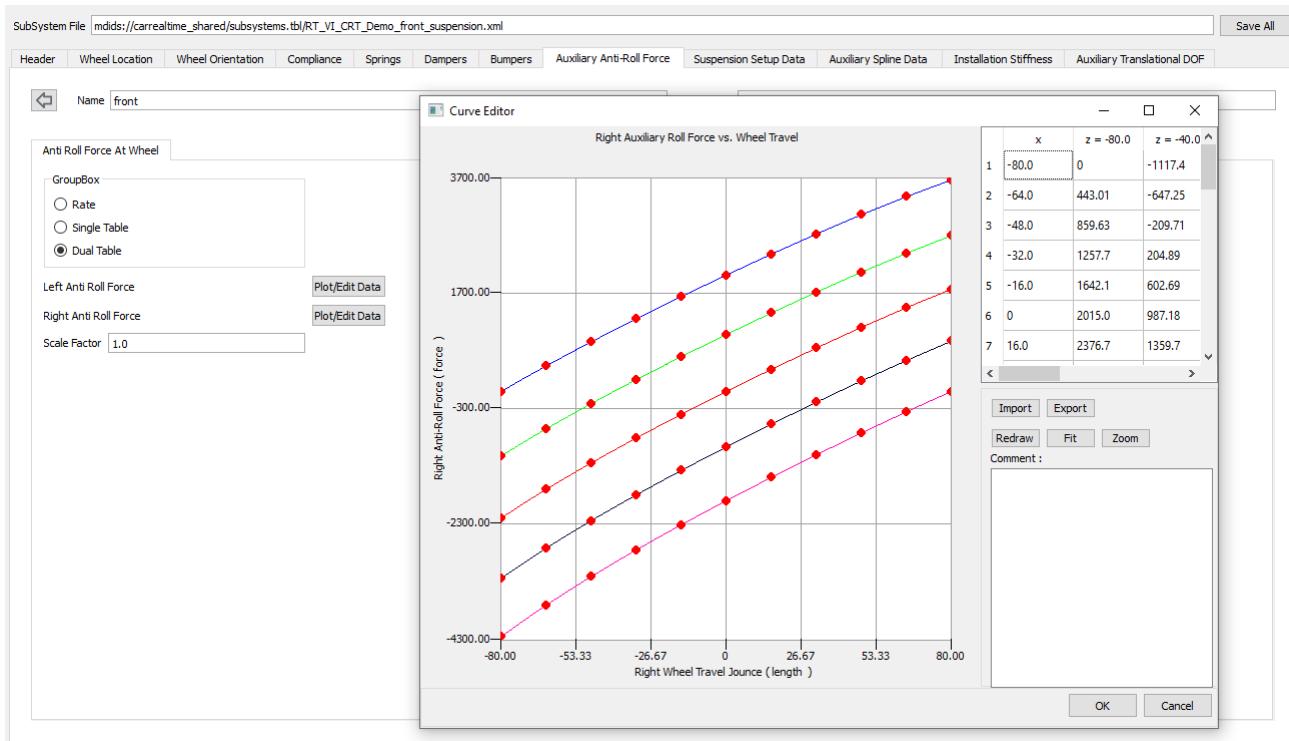
Auxiliary anti roll forces are used in VI-CarRealTime to introduce the effect of elastic elements connecting left and right paired wheels (typically anti roll bars).

Depending on the conceptual suspension model the force is projected at the wheels as a pair of opposite vertical forces acting between the wheel (unsprung mass) and the body chassis.

The value of the force is related to the left/right wheel jounce difference and can be input using two methods:

- **Rate**
the force intensity is computed by the product of the rate value and the L/R jounce difference.
- **Single Table**
the force is introduced using a spline having as first independent variable (X) the L/R jounce difference (delta jounce). It is also possible to have a second independent variable (Z) which is the L/R average jounce. This allows to take into account the non linear behaviour of the anti roll bar due to suspension geometry.
- **Dual Table**
the force is introduced using two splines having as first independent variable (X) the wheel travel of one wheel and as second independent variable (Z) the other wheel travel. The Dual Table option supports the possibility of asymmetric auxiliary anti roll forces.
- **Scale Factor**
scaling factor which multiplies the anti roll force characteristic curve.

VI-CarRealTime



Note: It is possible to specify a proportional damping applied to the anti roll auxiliary forces using the auxiliary_roll_damping tree in [System Parameters](#).

Anti-Roll Bar

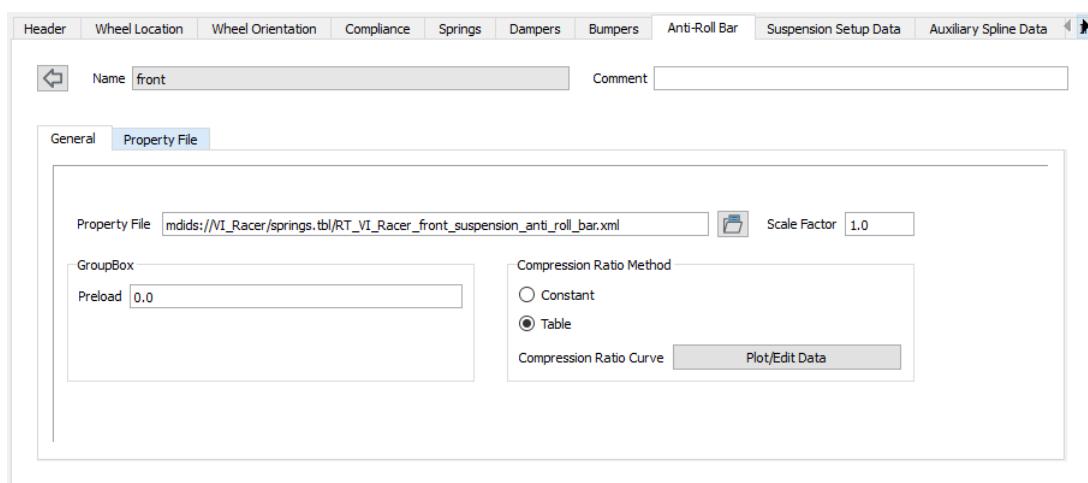
Anti-Roll Bar panel is used to characterize the anti-roll bar element of the suspension.

The panel is generated whenever the anti-roll bar of a suspension is characterized as an **element**, otherwise the [Auxiliary Anti-Roll Force](#) panel will be used to characterize the suspension anti-roll effect.

Anti-Roll Bar panel appears when the suspension is created:

1. by using VI-CarRealTime Adams plug-in and the model to be converted has the [cos_ARB_force](#) communicator.
2. by using VI-SuspensionGen tool

In such case the anti-roll bar is characterized with its own stiffness and preload and by using a motion ratio the force at element is applied at wheel.



The component properties of the anti-roll bar are described by:

- **General panel**

used to project at wheel center the effect of the anti-roll bar component:

- [Preload](#)
- [Compression ratio method](#)

- **Property File panel**

used to describe component specific properties.

Preload

It is the force acting at the anti-roll bar element at design time.

Compression ratio method

It is used to convert the anti-roll bar component deformation to the equivalent wheel bounce, since in VI-CarRealTime the anti-roll bar element is conceptual. Available methods are:

- **Constant rate**

Value is set by **Property Editor** and expresses the ratio between the wheel bounce and component rotational deflection.

- **Table**

Data can be input/edited using the **Curve Editor**. The curve has wheel bounce as independent variable (X) and expresses the anti-roll bar rotational deflection(Y).

Property File

The anti-roll bar xml [property file](#) is read and the data are used to fill up the Properties panel. The user can modify the anti-roll bar element characteristics and then save the modification in the related property file.

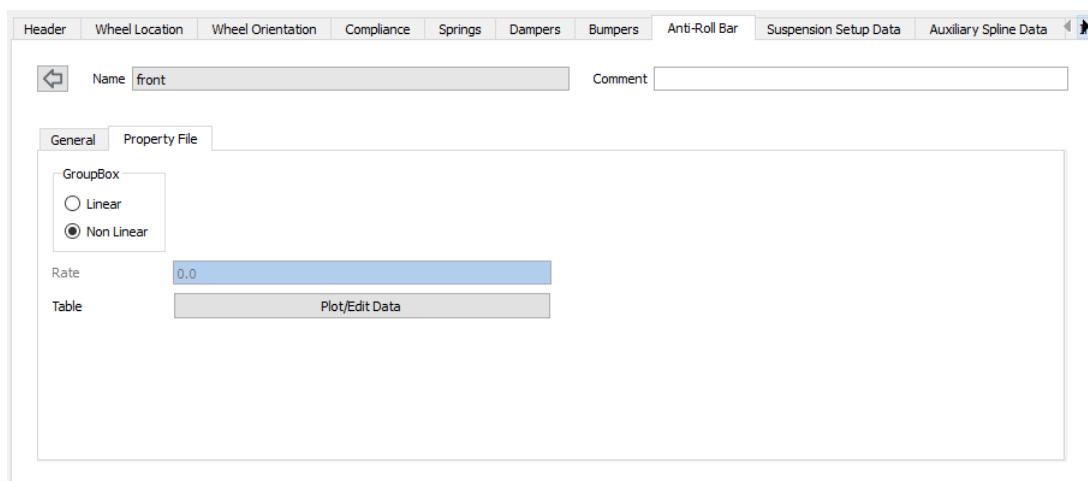
Method

- **Linear**

Allows to define a linear anti-roll bar property file characteristic.

- **Nonlinear**

Allows to define a non-linear anti-roll bar property file characteristic. The element torque is mapped as a function of element rotational deformation.

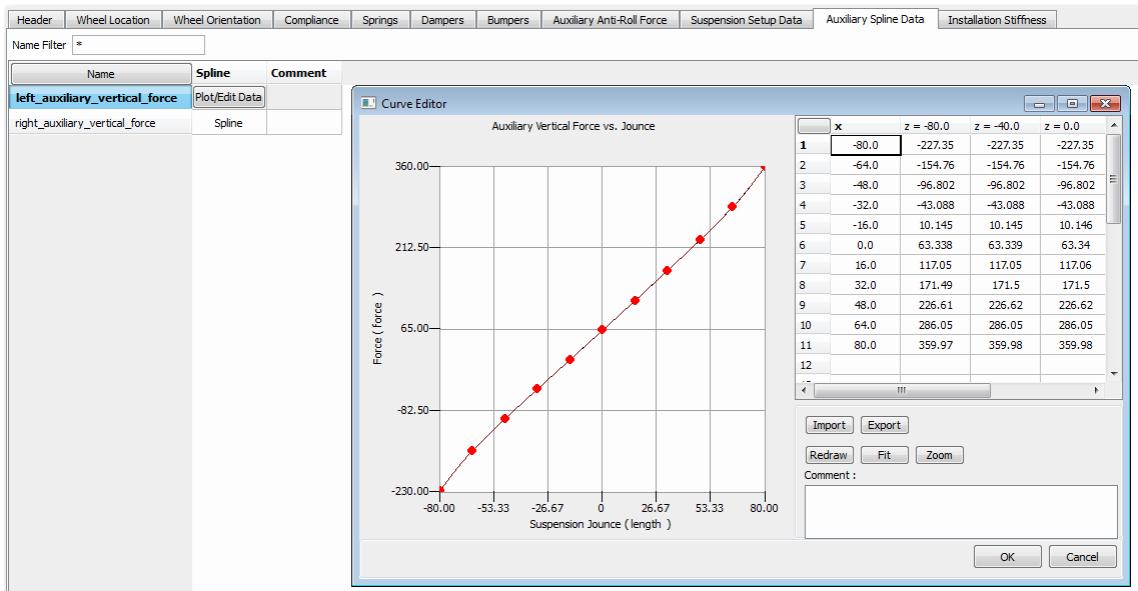


Auxiliary Vertical Force

Auxiliary vertical forces introduce the effect of suspension vertical elasticity not due to standard components (springs, bumpers etc.).

It is defined by two sets of spline data expressing the wheel auxiliary vertical force as a function of wheel jounce. Optionally a second independent variable (Z) can be used to support the auxiliary vertical Force dependency of both left and right axle wheels travels.

Spline data can be edited using the Curve Editor.



Suspension Setup Data

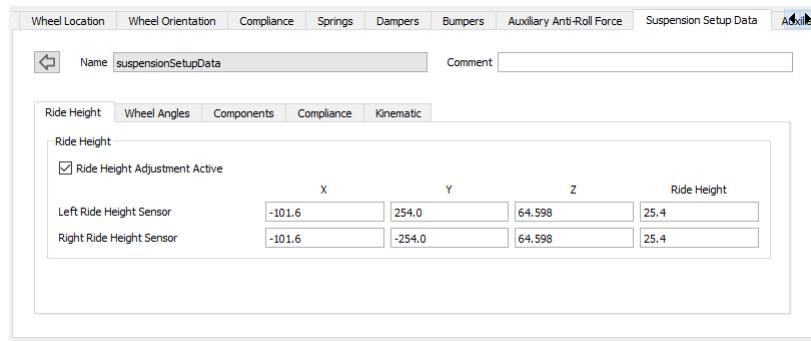
VI-CarRealTime features the capability of adjusting suspension and vehicle characteristics. Please refer to [Adjustments](#) topic for all the details.

The following subpanels are available:

- [Ride Height](#)
- [Wheel Angles](#)
- [Components](#)
- [Compliance](#)
- Kinematic

The following panels are available:

- [Ride Height](#)



Ride Height Adjustment Active

Toggle button which allows to activate the ride height adjustment. In order for the ride height adjustment to be performed, both front and rear suspension toggles must be checked.

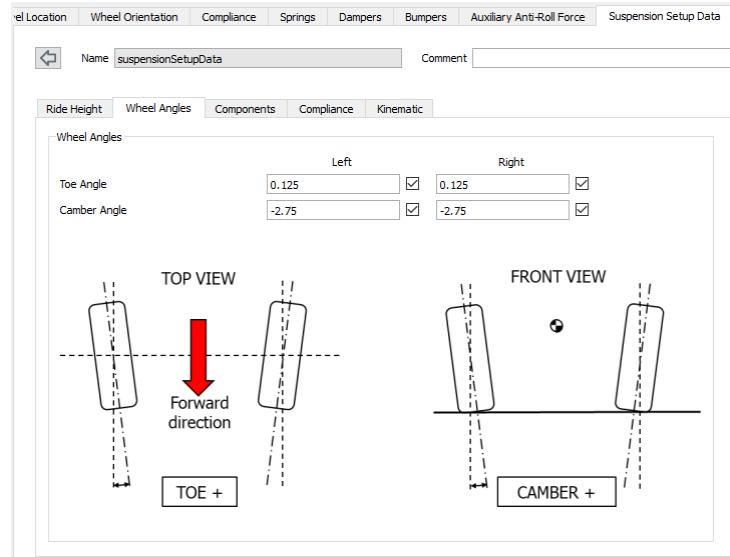
Left Ride Height Sensor

It allows to define the location, in design condition, of the left ride height sensor (X,Y,Z coordinates in [Vehicle Reference System](#)) and to set the target Ride Height value. The sensor runtime position is computed as simulation result also when the ride height adjustment is disabled.

Right Ride Height Sensor

It allows to define the location, in design condition, of the right ride height sensor (X,Y,Z coordinates in [Vehicle Reference System](#)) and to set the target Ride Height value. The sensor runtime position is computed as simulation result also when the ride height adjustment is disabled.

- **Wheel Angles**



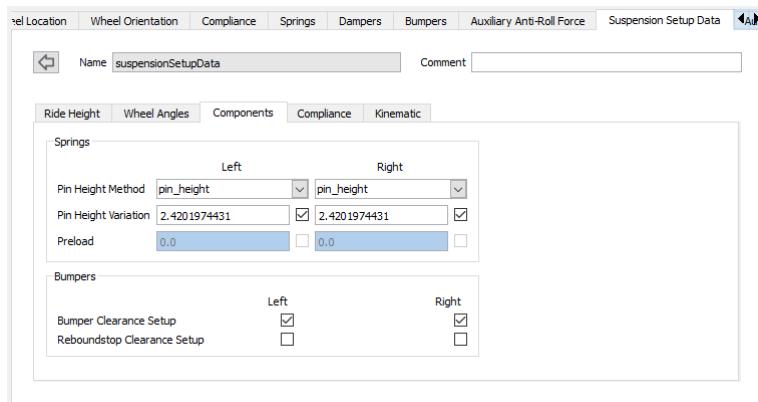
Toe Angle

Set, when the related toggle button is checked, left and right target toe angle value.

Camber Angle

Set, when the related toggle button is checked, left and right target camber angle value.

- **Components**



Pin Height Method

Option menu which allows to define the pin height adjustment method. Available choices are `pin_height` and `preload`. Such adjustment is performed only for suspension whose [setup mode](#) is set to 1.0 (pushrod).

Pin Height Variation

When the related toggle is checked, the field allows to define the target spring length variation. Such value is the spring length difference between design condition and after setup condition.

Preload

It defines the target preload of the spring.

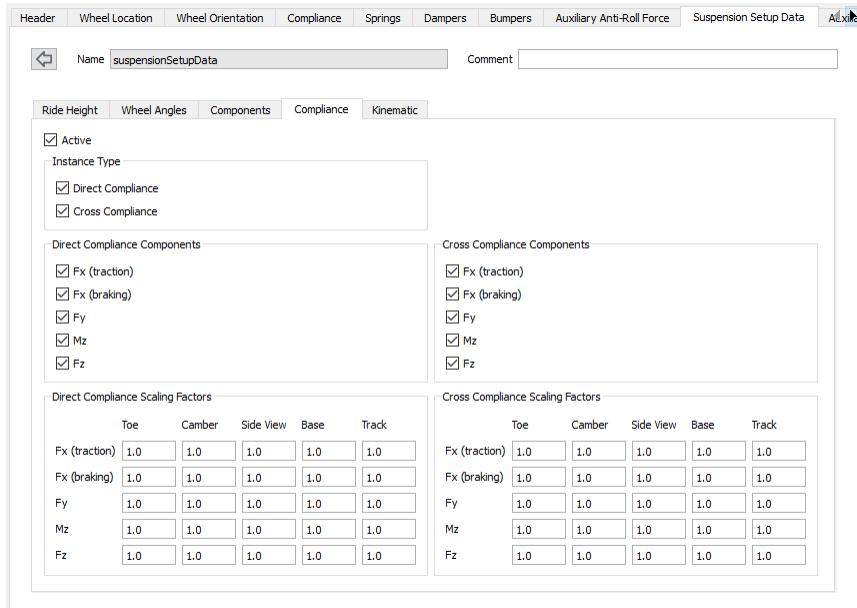
Bumper Clearance Setup

It activates bumpstop clearance adjustment.

Reboundstop Clearance Setup

It activates reboundstop clearance adjustment.

- **Compliance**



Active

Toggle button to activate/deactivate all the [compliance](#) curves.

Direct Compliance

Toggle button to activate/deactivate all the direct [compliance](#) curves. The button is enabled only when Active toggle button is checked.

Cross Compliance

Toggle button to activate/deactivate all the cross [compliance](#) curves. The button is enabled only when Active toggle button is checked.

Direct Compliance Components

Toggle buttons to deactivate singular [direct compliance](#) curves. The buttons are enabled only when the related Instance Type toggle button is checked.

Cross Compliance Components

Toggle buttons to deactivate singular [cross compliance](#) curves. The buttons are enabled only when the related Instance Type toggle button is checked.

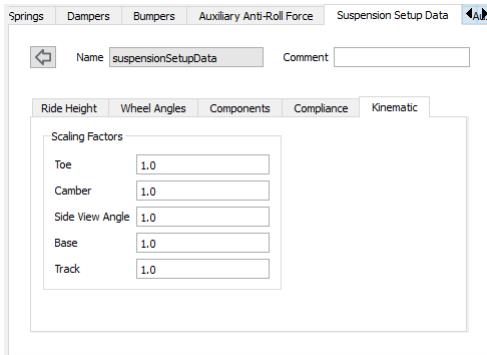
Direct Compliance Scaling Factors

For each [direct compliance](#) map it is possible to define a scaling factor coefficient which multiplies the related map values.

Cross Compliance Scaling Factors

For each [cross compliance](#) map it is possible to define a scaling factor coefficient which multiplies the related map values.

- **Kinematic**

**Scaling Factors**

For each kinematic map ([Wheel Location](#) and [Wheel Orientation](#)) it is possible to define a scaling factor coefficient which multiplies the related map values.

The adjustment practice, common in racing applications, is implemented in VI-CarRealTime by modifying the values that the adjustable quantities assume after static equilibrium of the vehicle and it is achieved using different methods depending on vehicle and suspension architecture.

The available suspension adjustments are:

- [Ride Height](#)
- [Toe Angle](#)
- [Camber Angle](#)
- [Pin Height](#)
- [Bumpers clearance setup](#)
- [Reboundstop clearance setup](#)

Ride Height

Ride height adjustment is achieved by modifying the vertical distance from the ground of specific points belonging to the body chassis. In sports-car disciplines the practise is important in order to tailor aero forces (which typically are depending on the distance of skid plate from ground), to avoid skid plate impact on the ground, and to tune the vehicle CG vertical position.

The adjustment in VI-CarRealTime is done identifying two sensor points on the front suspension (belonging to the vehicle sprung part) and one sensor point on the rear. For each sensor point it is possible to specify the position in the VI-CarRealTime global reference frame and the desired ride height. Ride height adjustment can be disabled; no RH adjustment is performed when either the front or rear suspension RH adjustment flag is disabled.

In actual vehicles this kind of adjustments is done by means of:

- **Push-rod length adjustments** (mode 1)
 - **Spring preload** (mode 2) for vehicle without pushrods such as McPherson or double wishbone suspensions.

The main difference of the two methods is that in presence of a pushrod the position of chassis part can be positioned w.r.t. the ground without affecting spring component loads.

Both methods are available in VI-CarRealTime.

Setup phase in VI-CarRealTime is performed through a static equilibrium analysis during which all the targets specified (ride height, pin height, cross weight, camber and toe) are all adjusted at the same time by mean of several PID controllers. Parameters to tune the different PID controllers are available in vehicle system file and are editable through [Property Editor](#).

In VI-CarRealTime static equilibrium is performed by means of a dynamic analysis which is continuing until the kinetic energy of the system falls below a given threshold.

Subsystem Definition	Properties	Output channels		
		Name	Value	Comment
system_parameters		ride_height_preload_kp	100000.0	Proportional Gain for Ride Height Adjustment (preload)
trailer		ride_height_preload_kd	10000.0	Derivative Gain for Ride Height Adjustment (preload)
aerodynamics		ride_height_preload_ki	100000.0	Integral Gain for Ride Height Adjustment (preload)
powertrain		ride_height_standard_kp	1.0	Proportional Gain for Ride Height Adjustment
circular_buffer		ride_height_standard_kd	0.0	Derivative Gain for Ride Height Adjustment
statics		ride_height_standard_ki	10.0	Integral Gain for Ride Height Adjustment
body_compliance		mode	1.0	Setup mode: 1-> standard (pushrod) 2-> preload
setup				
ride_height				
pin_height				
cross_weight				
setup_wheels				
mode				
tolerance				
cross_weight_tolerance				
statistics				

- | | | |
|----------------------------------|----------------------|---|
| • mode | set to 1
set to 2 | the ride adjustment mimics the presence of a pushrod in suspension (no spring preload adj)
the spring preload is used to achieve the desired ride height |
| • ride_height_standard_kp | mode 1 | proportional gain for RH Adj |
| • ride_height_standard_kd | mode 1 | derivative gain for RH Adj |
| • ride_height_standard_ki | mode 1 | integral gain for RH Adj |
| • ride_height_preload_kp | mode 2 | proportional gain for RH Adj |
| • ride_height_preload_kd | mode 2 | derivative gain for RH Adj |
| • ride_height_preload_ki | mode 2 | integral gain for RH Adj |

Mixed RH setup (i.e. different setup mode for front and rear suspension) is not supported.

Toe Angle

a value for each corner can be specified and eventually deactivated.

Camber Angle

a value for each corner can be specified and eventually deactivated.

Pin Height

pin height adjustment can be set in VI-CarRealTime using **pin height variation** and **preload** method; Pin height adjustment requires that the setup mode is set to 1 (pushrod) in System Properties ->system_parameters->setup->mode. Value will be ignored when setup mode is set to 2.

When Preload method is selected, the solver computes the reboundstop clearance required to provide the desired preload and then it uses such computed clearance to perform all the others setup stages.

Bumpers clearance setup

a flag for each corner can be set to restore bumpers clearance after setup.

Reboundstop clearance setup

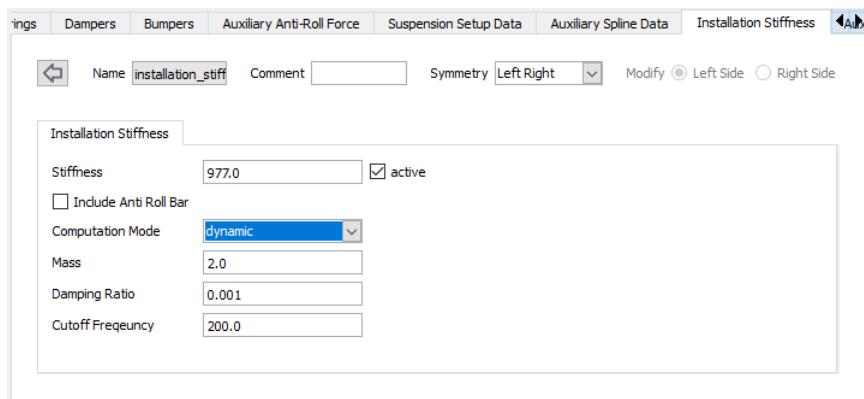
a flag for each corner can be set to restore reboundstop clearance after setup; option is not available when pin height adjustment is done through preload. In fact in that case reboundstop displacement is modified in order to get desired preload after statics.

Installation Stiffness

VI-CarRealTime supports the possibility of introducing the stiffness value of elements arranged in series to the main spring.

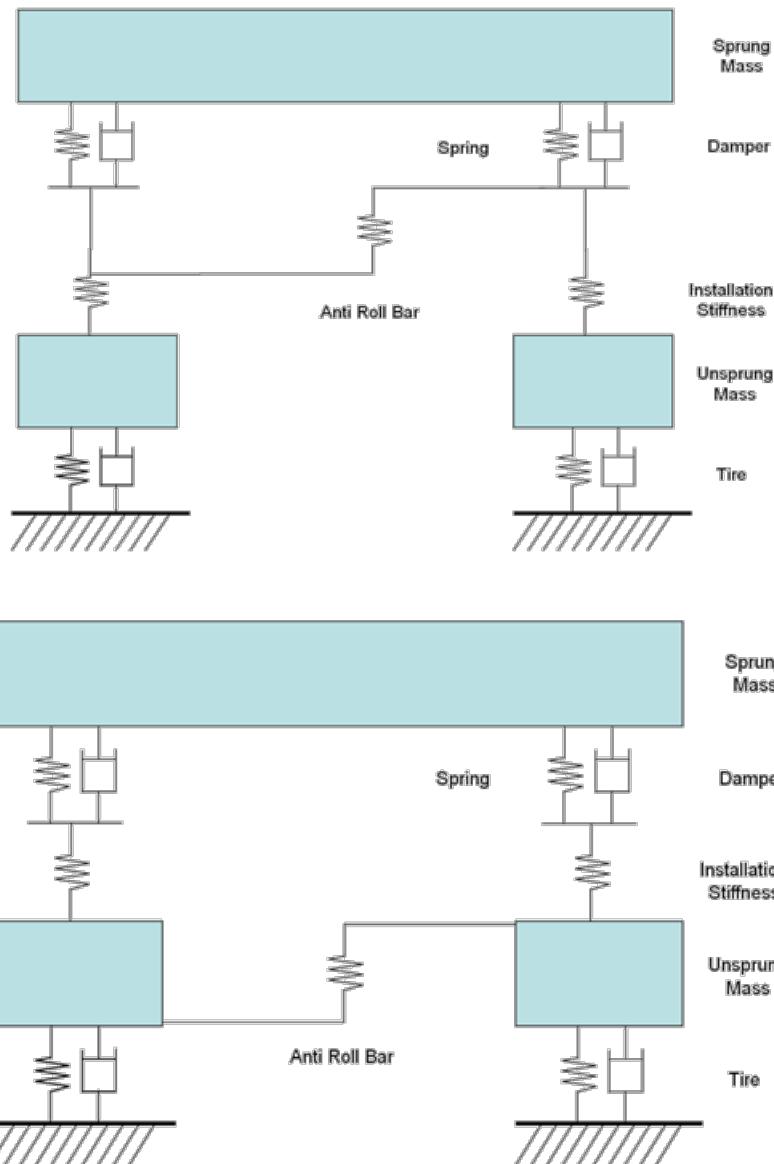
The installation stiffness is used to model the vertical compliance of the suspension on hub side.

The stiffness is measured at the component location.



The following parameters can be set when **active** toggle button is checked:

- **Stiffness**
stiffness (k) of the spring arranged in series with the main spring.
- **Include Anti Roll Bar**
toggle button which includes (first scheme below) or neglect (second scheme below) the anti roll bar effect on installation stiffness.



- **Computation Mode**

allows the user to choose how the solver must compute the installation stiffness during the analysis. The installation stiffness provides an offset (z) which is summed to the suspension jounce (J) for modifying the entering point in the progression curves of the elements (spring, damper, etc), so that the independent variable is no more J but $(J - z)$.

Available options for computation mode are:

- **static**

the offset z is evaluated as:

$$z = \frac{F_{z_{hub}}}{k}$$

- **dynamic**

the offset z is the result of the following differential equation:

$$m\ddot{z} + c\dot{z} + kz = F_{z_{hub}}$$

When dynamic computation mode is selected, the following parameters are available:

- **Mass**

dynamic mass value of the unsprung mass (m in the formula above).

- **Damping Ratio**

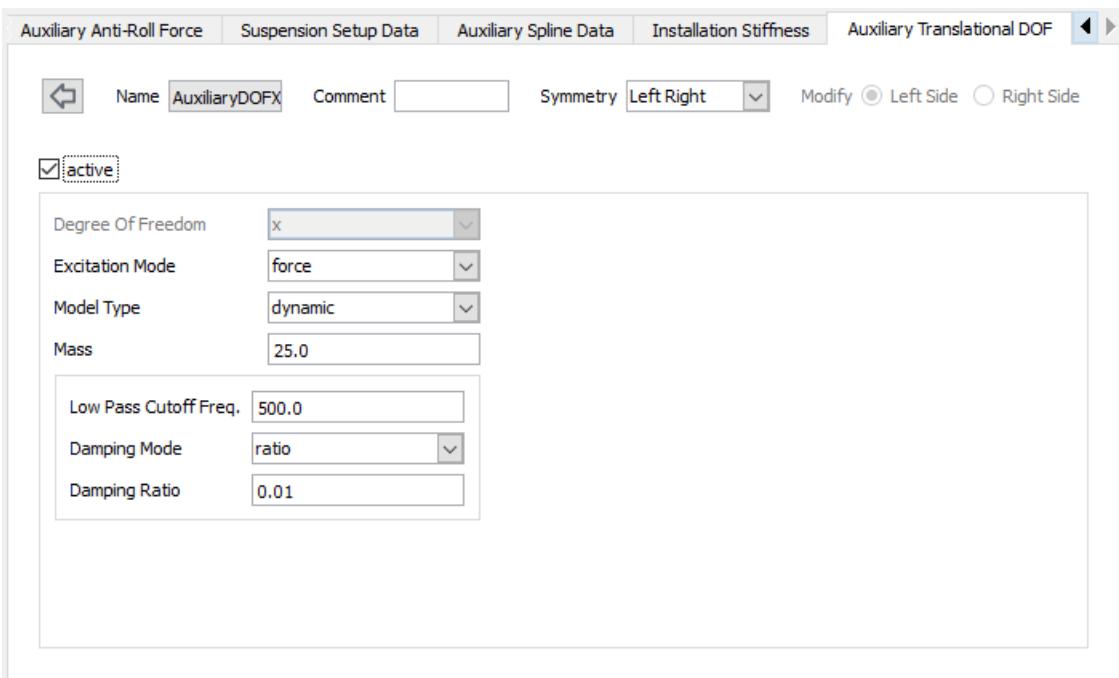
damping ratio ζ (percentage) of the installation system, used to evaluate the term c in the formula above: $c = \zeta \cdot k$

- **Cutoff Frequency**

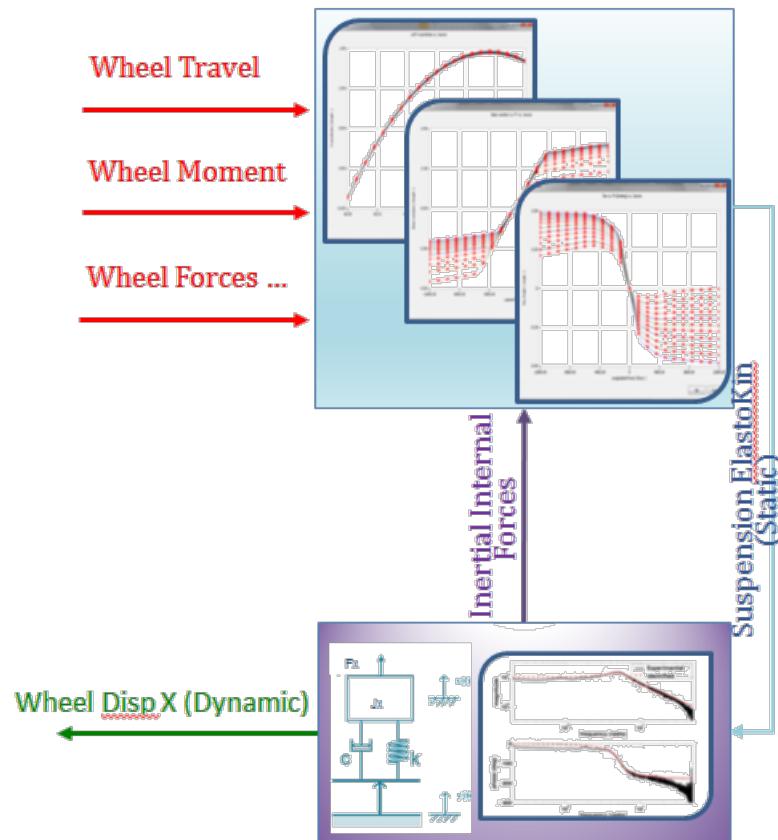
cut-off frequency of the low pass filter used for the system.

Auxiliary Translational DOF

It is possible to enable an additional degree of freedom between unsprung mass and chassis in order to capture the dynamics of the wheel and its vibration phenomena along x direction.



VI-CarRealTime suspension model features look-up tables for kinematic and compliance and those effects are computed/applied in a static manner depending on wheel travel and steering input (for front suspension) and load conditions. By activating the auxiliary translational DOF, an additional system interacting with the suspension system is enabled according to the flow chart below:



The **Excitation Mode** parameter defines how the longitudinal degree of freedom is excited:

- **base**
the auxiliary DOF is excited through the displacement resulting from elasto-kinematics curves look-up tables.
- **force**
the auxiliary DOF is excited from external wheel force.

Three different flavors are available depending on the chosen **Model Type**:

- [Static](#)
- [Dynamic](#)
- [Transfer Function](#)

Static

The longitudinal DOF is computed by using the [compliance maps](#) (0 order compliance).

<input checked="" type="checkbox"/> active	
Degree Of Freedom	x
Excitation Mode	force
Model Type	static

Note: when the static approach is selected, an eventual [compliance_velocity](#) is disregarded.

Dynamic

The resulting compliance of the longitudinal DOF will be computed runtime by solving a second order non-linear differential equation.

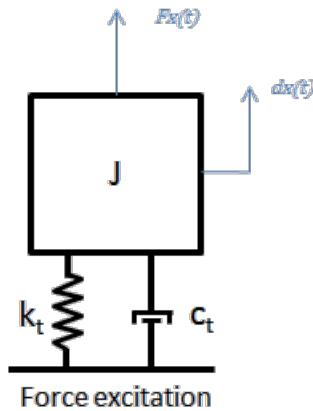
The approach is different depending on the selected **Excitation Mode**:

- **force**
the longitudinal displacement is computed as follows:

$$m\ddot{x} + k_{eq}(x + \epsilon\dot{x}) = F$$

where:

- m : is the equivalent mass for longitudinal degree of freedom;
- k_{eq} : is the equivalent stiffness in the x-direction computed runtime by existent compliance spline;
- ϵ : is the percentage damping;
- F : is the longitudinal force applied to the wheel.

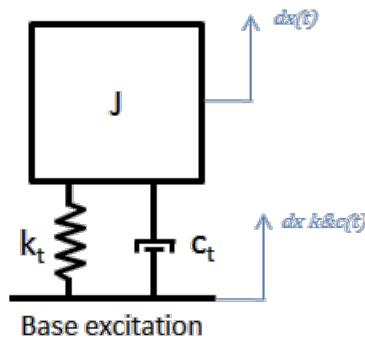


- **base**
the longitudinal displacement is computed as follows:

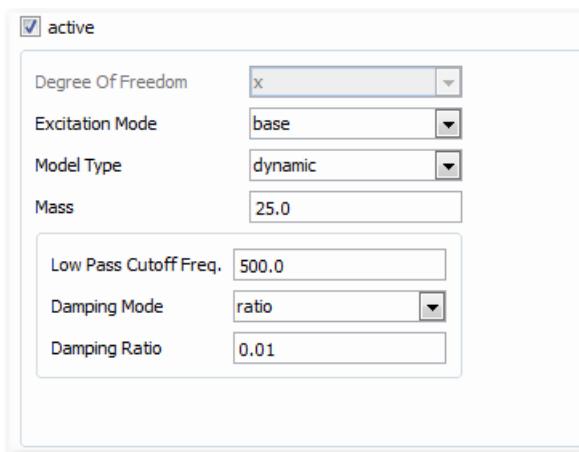
$$m\ddot{x} + k(x - \mu) + c(\dot{x} - \dot{\mu}) = 0$$

where:

- m : is the equivalent mass for longitudinal degree of freedom;
- k : is the stiffness in the x-direction computed runtime by existent compliance spline;
- c : is the damping coefficient;
- μ : is the longitudinal displacement of the constraint between suspension hub and unsprung mass.



The compliance contribute x and its derivative states, velocity and acceleration, are added runtime to the definition of prescribed translational joint which connects unsprung mass to chassis part.



The following parameters applies both for base and force excitation mode:

- Low Pass Cutoff Freq.**
cut off frequency of the runtime low-pass filter used during x compliance contribute computation.
- Damping Mode**
select the damping mode to be used, between **ratio** and **damping**.
- Damping Ratio**
The field is active when Damping Mode is set to *ratio* and according to the formulas described above the damping ratio is:
 ϵ for force excitation mode.
 c/k for base excitation mode.
- Damping**
The field is active when Damping Mode is set to *force* and according to the formulas described above, the damping is:
 $k_{eq} * \epsilon$ for force excitation mode.
 c for base excitation mode.

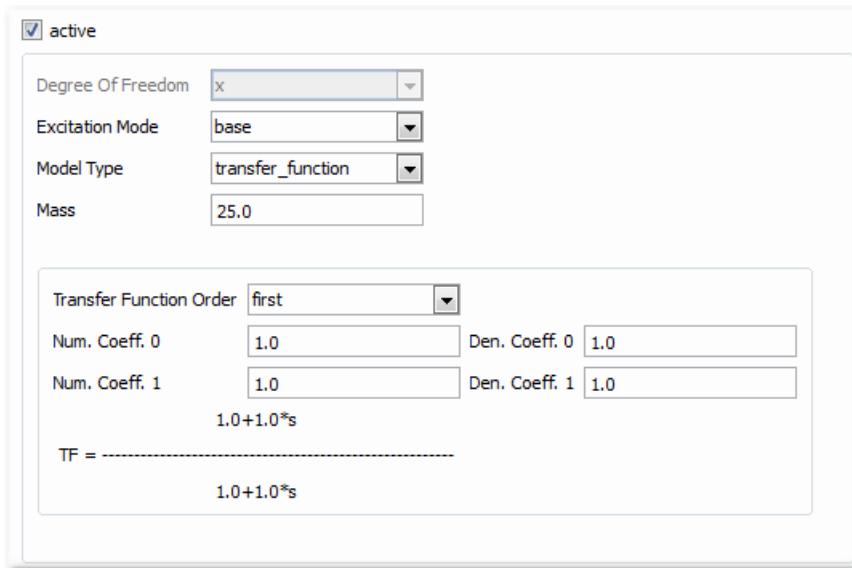
Transfer Function

With Model Type set to *transfer_function*, it's up to the user to define the transfer function that characterizes the dynamic system.

In particular, **Transfer Function Order** option menu must be used to choose the order of the transfer function (first, second, third, forth).

Depending on the order selected, the related coefficients must be set both for numerator and denominator.

Mass field is the equivalent mass of the longitudinal degree of freedom.



Note: the transfer function coefficients are defined per unit of mass.

Steering Subsystem

The Steering subsystem in VI-CarRealTime can be model using different approaches.

In a basic formulation no physical part or linkage is present in the model; the steering and other movements of the wheels are related to steering wheel (or rack) motion and wheel jounce by lookup tables.

Compliance effect of the steering column is also taken into account as depending on kingpin moment.

The VI-CarRealTime steering model is also capable of evaluating steering wheel torque feedback.

Beside to the basic model, an advanced rack-pinion steering model which physically models the steering in terms of mechanical and electric/hydraulic components is available. Such a model has 2 degrees of freedom and allows to capture the dynamic of the steering mechanism.

In the case of usage of the advanced steering model the basic model remains active but the wheel kinematics is controlled by the rack displacement computed in the advanced model. In such a way it is possible to comply with a detailed and fast model at the same time.

The Steering subsystem GUI is composed by the following tabs:

- [Steering System](#)
- [Auxiliary Spline Data](#)
- [Rack-Pinion Steering](#)
- [Components](#)

Steering System

The dialog box is including the basic steering model properties:

- [Kinematics](#)
- [Compliance](#)
- [Kingpin](#)
- [Steering Feedback](#)

- [Inertia and Damping](#)

Steering system kinematics in VI-CarRealTime are described using lookup tables.

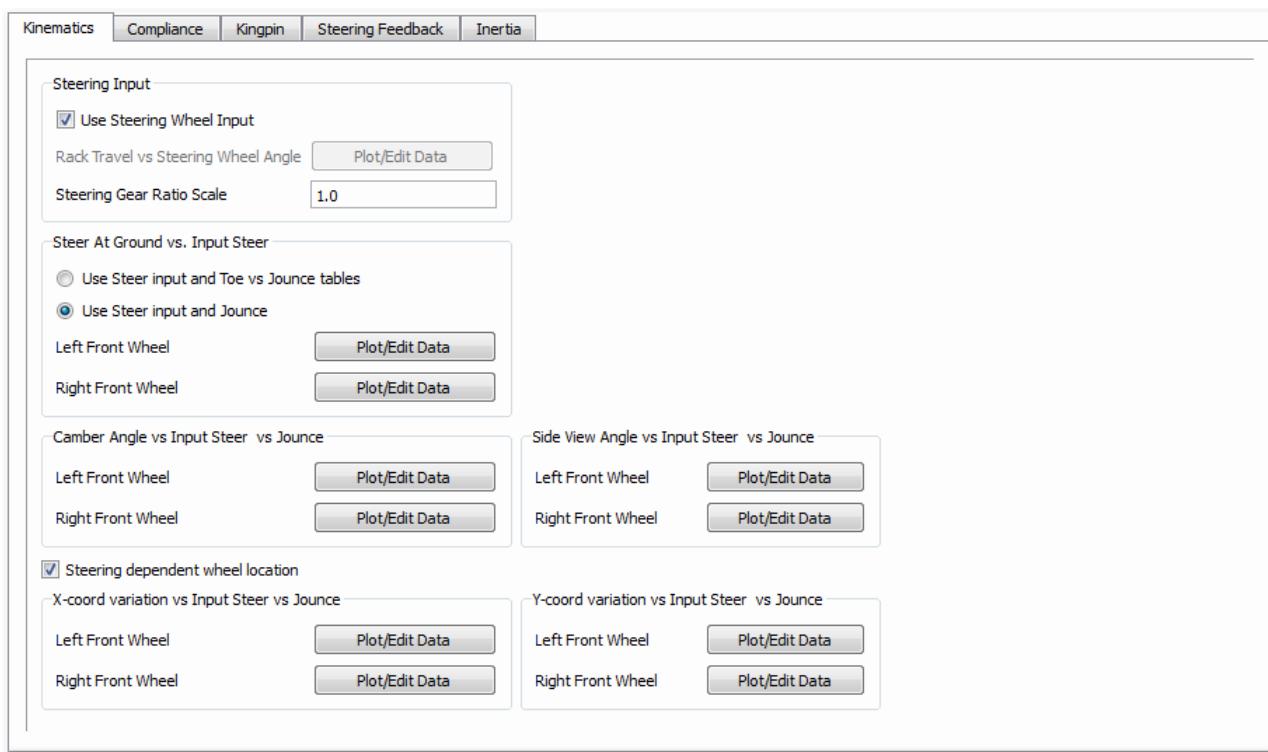
The data stored in the curves introduce the dependency on **steering wheel angle** (or **rack travel**) and **wheel jounce** of the following quantities:

- **Steer At Ground** (Left/Right)
is the angle measured from the vehicle heading to the line formed by the intersection of the wheel plane with the ground plane. The Steer At Ground angle is positive when a wheel is rotated to the left as if the vehicle were making a left turn.
- **Camber angle** (Left/Right)
is the angle the wheel plane has with respect to the vehicle's vertical axis. It is positive when the top of the wheel leans outward from the vehicle body.
- **Side view angle** (Left/Right)
is the [variation](#) of the wheel carrier side-view rotation angle with respect to the caster angle at design time. It is positive for a clockwise rotation, as seen from the left side of the vehicle.
- **Wheel center x-coordinate variation** (Left/Right)
describes wheel base variation in wheel location reference system as a function of wheel jounce.
- **Wheel center y-coordinate variation** (Left/Right)
describes wheel track variation in wheel location reference system as a function of wheel jounce.

The front kinematic is computed:

- by using uniquely the curves in steering subsystem (3D splines), so neglecting the splines defined in front suspension subsystem, when the front suspension kinematic curves are mapped as [independent](#) splines;
- by summing up the contributes of both front suspension kinematic curves and steering kinematic curves when the front suspension kinematic curves are mapped as [dependent](#) splines.

The image below shows the steering system kinematics Property Editor:



Available parameters are:

- **Use Steering Wheel Input**

Check box for activating the steering system kinematics dependency on steering wheel angle: when the toggle button is checked all the aforementioned spline are defined as a function of the steering wheel angle (other than the suspension jounce).

- **Rack Travel vs Steering Wheel Angle**

Table relating the steering wheel input to rack travel kinematics; this spline is used when the **Use Steering Wheel Input** toggle button is unchecked: in this case all the splines which define the steering kinematic are defined as a function of the rack displacement. When **Use Steering Wheel Input** toggle is checked, the spline is anyway used to compute the rack displacement output request.

- **Steering Gear Ratio Scale**

Scaling factor that divides the steering wheel angle computed by VI-Driver during the simulation; the resulting value is used to enter the splines.

- **Steer at ground vs Input Steer**

The box is used to set properties for the steer angle method. Two options are available:

- **Use Steer input and Toe vs Jounce Tables**

Using this option steer angle is defined by the combination of front suspension subsystem curves for toe (dependency on jounce) with the steer angle curve in steering subsystem (steering wheel angle or rack travel dependency). Total steer angle (δ) is obtained b the sum of the two terms (2D splines):

$$\delta_{SaG} = \delta(jounce) + \delta(steering_wheel_angle)$$

- **Use Steer Input and Jounce**

Using this option the steer angle is defined by a 3D spline having the steering wheel angle as first independent variable and the wheel jounce as second independent variable. Total Steer at Ground (δ_{SaG}) can be expressed as:

$$\delta_{SaG} = \delta(\text{steering_wheel_angle/rack_travel}, \text{jounce})$$

- **Camber Angle vs Input Steer vs Jounce**

The box can be used to edit the curves for camber angle dependency on steering wheel angle. Buttons open the Curve Editor; the curves are defined as 3D spline having the steering wheel angle as first independent variable and the wheel jounce as second independent variable. Camber angle (κ) can be expressed as:

$$\kappa = \kappa(\text{steering_wheel_angle/rack_travel}, \text{jounce})$$

- **Side View Angle vs Input Steer vs Jounce**

The box allows the user to edit the curves for side view angle dependency on steering wheel angle. Buttons open the Curve Editor; the curves are defined as 3D spline having has first independent variable the steering wheel angle and as second independent variable the wheel jounce. The side view angle (θ) can be expressed as:

$$\theta = \theta(\text{steering_wheel_angle}, \text{jounce})$$

- **Steering dependent wheel location**

The flag is used to activate the dependency of track and base variation (kinematics) on steering wheel angle (or rack displacement). When the toggle button is unchecked the [curves](#) in suspension subsystem are used.

- **X-coord variation vs Input Steer vs Jounce**

The box can be used to edit the curves for wheelbase dependency on steering wheel angle. Buttons open the Curve Editor; the curves are defined as 3D spline having the steering wheel angle as first independent variable and the wheel jounce as second independent variable. The wheelbase variation (x) can be expressed as:

$$x = x(\text{steering_wheel_angle}, \text{jounce})$$

- **Y-coord variation vs Input Steer vs Jounce**

The box can be used to edit the curves for wheel track dependency on steering wheel angle. Buttons open the Curve Editor; the curves are defined as 3D spline having the steering wheel angle as first independent variable and the wheel jounce as second independent variable. The wheel track variation (y) can be expressed as:

$$y = y(\text{steering_wheel_angle}, \text{jounce})$$

VI-CarRealTime vehicle model include the capability of modeling steering system compliance.

Since steering systems connect left and right wheels of vehicle front suspensions, the steering rack effect and other linkages are included in suspension compliance.

The effect of steering column compliance is included in steering subsystem.

The deformation of the steering column is considered as depending on total kingpin moment and it is used to modify the steering wheel angle input used as independent variable in steering kinematic curves.

The total steering wheel angle input (ζ) is given by the equation:

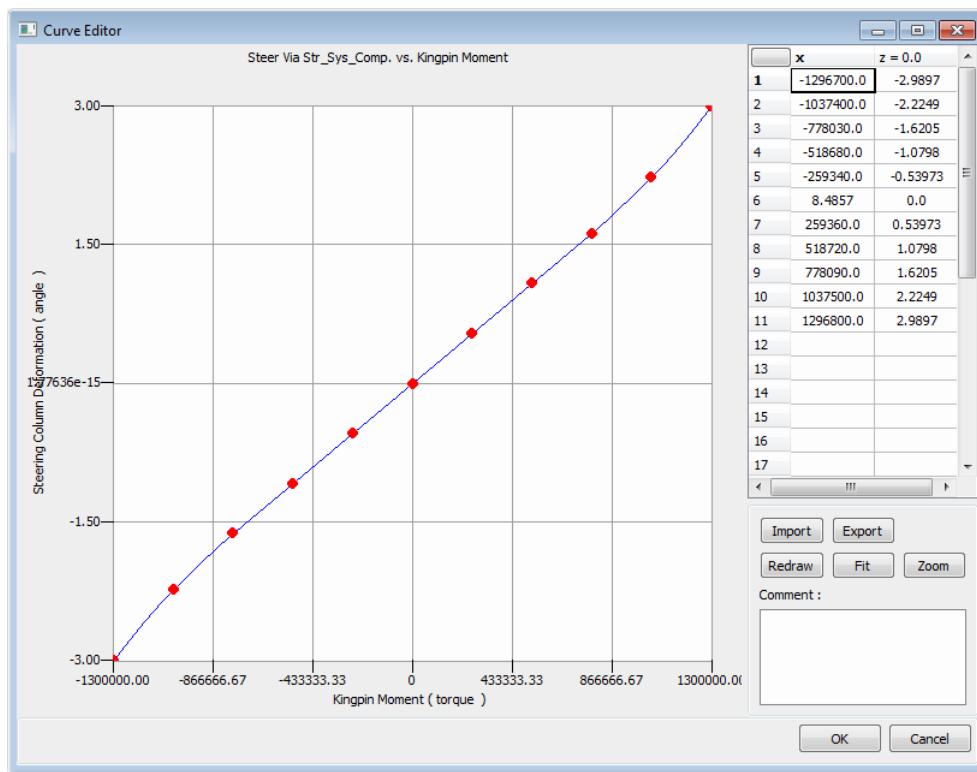
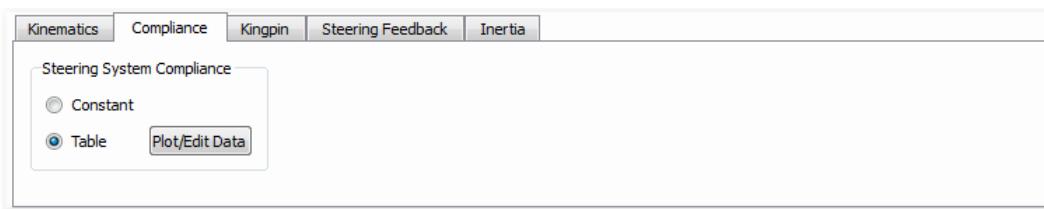
$$\zeta = \zeta_{\text{input}} + \Delta\zeta_{\text{compliance}} (\text{kingpinMoment})$$

where

- ζ_{input} is the steering hand wheel rotation;

- $\Delta\zeta_{\text{compliance}}$ is the steering column deformation.

The steering column deformation can be expressed as a constant rate or as 2D spline data (function of total kingpin moment).



The Kingpin geometry is used in VI-CarRealTime to compute the steering wheel torque feedback.

The Kingpin geometry defines the steering axis position and is described by the following quantities:

- **Caster Angle**

is the angle in the side elevation (vehicle XZ plane) between the steering (kingpin) axis and the vehicle's vertical axis. It is positive when the steer axis is inclined upward and rearward.

The Caster Angle is mapped as a function of Steering Wheel Angle (or Rack Travel) and Suspension Jounce.

- **Kingpin Inclination**

is the angle in the front elevation between the steer axis (the kingpin axis) and the vehicle's vertical axis. It is positive when the steer axis is inclined upward and inward.

The Kingpin Inclination Angle is mapped as a function of Steering Wheel Angle (or Rack Travel) and Suspension Jounce.

- **Arbitrary Point on Steer Axis**

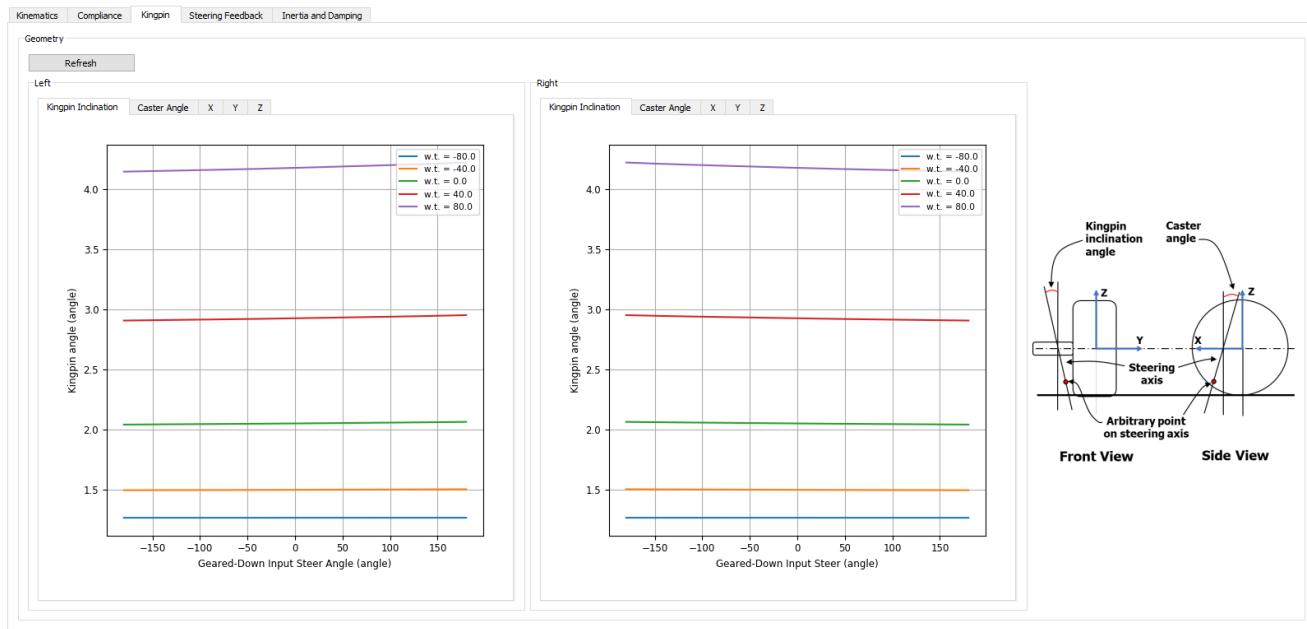
in order to fully define the kingpin axis it is needed to identify a point belonging to the axis.

VI-CarRealTime

The X,Y,Z locations of the point are mapped as a function of Steering Wheel Angle (or Rack Travel) and Suspension Jounce.

The coordinates of the point on kingpin axis are expressed w.r.t. wheel center position at design time in [chassis reference system](#).

The image below shows the Property Editor for kingpin kinematics.



Using the **Refresh** button in the upper part of the panel, the user can perform the calculation of Kingpin geometry using the data available in Steering and Front Suspension Subsystems.

VI-CarRealTime calculates the Kingpin axis as the instant axis of rotation of the wheel carrier parts. To calculate the steer axis, in physical terms, VI-CarRealTime first locks the spring travel and applies incremental steering motion. Then, from the resulting translation and rotation of the wheel carrier part, it is able to calculate the instant axis of rotation for each wheel. The instant axis of rotation corresponds to the Kingpin axis.

For **independent suspensions**, it calculates Base (x), Track (y), Toe (α), Camber (k), Side View Angle (θ) derivatives using kinematics data from Steering Subsystem as:

$$\frac{\partial x(\delta, J)}{\partial t} = \frac{\partial x(\delta, J)}{\partial \delta} \cdot \frac{\partial \delta}{\partial t} + \frac{\partial x(\delta, J)}{\partial J} \cdot \frac{\partial J}{\partial t} \quad (1)$$

Locking the spring travel we can write:

$$\frac{\partial f(J)}{\partial t} + \frac{\partial g(\delta, J)}{\partial t} = \frac{\partial f(J)}{\partial J} \cdot \frac{\partial J}{\partial t} + \frac{\partial g(\delta, J)}{\partial \delta} \cdot \frac{\partial \delta}{\partial t} + \frac{\partial g(\delta, J)}{\partial J} \cdot \frac{\partial J}{\partial t} = 0 \quad (2)$$

where $f(J)$ and $g(\delta, J)$ are the motion ratios of the spring function of Jounce (J) and SWA (δ).

Substituting the equation (2) in equation (1) the following equation is obtained:

$$\frac{\partial x}{\partial \delta} = \left(\frac{\partial x(\delta, J)}{\partial \delta} - \frac{\partial x(\delta, J)}{\partial J} \cdot \frac{\frac{\partial g(\delta, J)}{\partial \delta}}{\frac{\partial f(J)}{\partial J} + \frac{\partial g(\delta, J)}{\partial J}} \right) \frac{\partial J}{\partial t} \quad (3)$$

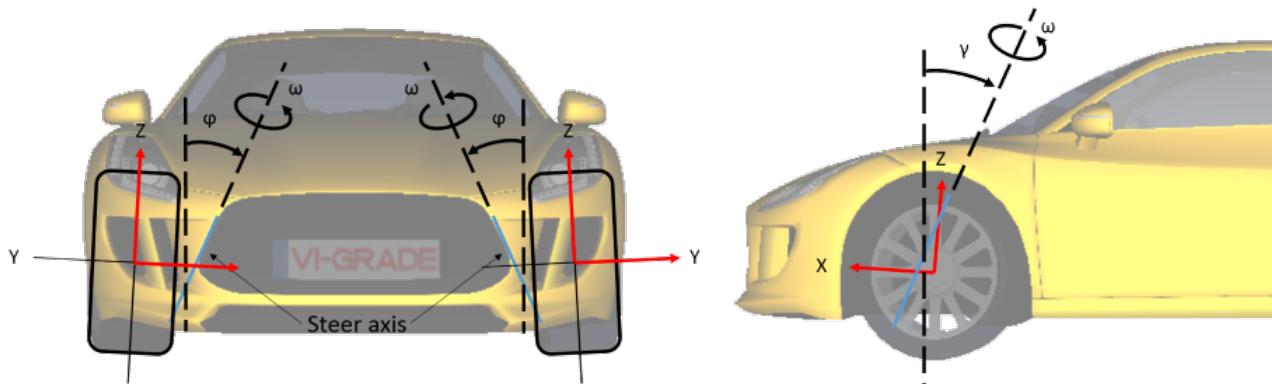
In a similar way, for **dependent suspensions**, it calculates Base (x), Track (y), Camber (κ), Side View Angle (γ) derivatives, this time using kinematics data from both Steering and Front Suspension Subsystem. In the following equations, we will use a 1 to mark data from Steering Subsystem and 2 to mark data from Suspension Subsystem.

$$x_L = x_{1L}(\delta, J_L) + x_{2L}(J_L, J_R) \quad (4)$$

$$\frac{\partial x_L}{\partial t} = \frac{\partial x_{1L}(\delta, J_L)}{\partial \delta} \cdot \frac{\partial \delta}{\partial t} + \frac{\partial x_{1L}(\delta, J_L)}{\partial J_L} \cdot \frac{\partial J_L}{\partial t} + \frac{\partial x_{2L}(J_L, J_R)}{\partial J_L} \cdot \frac{\partial J_L}{\partial t} + \frac{\partial x_{2L}(J_L, J_R)}{\partial J_R} \cdot \frac{\partial J_R}{\partial t} \quad (5)$$

$$\frac{\partial x_L}{\partial \delta} = \frac{\partial x_{1L}(\delta, J_L)}{\partial \delta} + \left(\frac{\partial x_{1L}(\delta, J_L)}{\partial J_L} + \frac{\partial x_{2L}(J_L, J_R)}{\partial J_L} \right) \frac{\partial J_L}{\partial \delta} + \frac{\partial x_{2L}(J_L, J_R)}{\partial J_R} \cdot \frac{\partial J_R}{\partial \delta} \quad (6)$$

In a similar way to independent suspensions, we can write equation (2) for both left and right sides (taking in account that this time spring motion ratio in Suspension Subsystem is function of both left and right jounce) and calculate $\partial J_L / \partial \delta$ and $\partial J_R / \partial \delta$ to substitute in equation (6).



Using this velocities, VI-CarRealTime calculates the angular velocity vector (ω).

Since ω vector is parallel to instant rotation axis, VI-CarRealTime calculate the kingpin inclination angle (φ) as

$$\varphi_L = -\tan^{-1}\left(\frac{\omega_{Ly}}{\omega_{Lz}}\right) \quad (7)$$

$$\varphi_R = \tan^{-1}\left(\frac{\omega_{Ry}}{\omega_{Rz}}\right) \quad (8)$$

and caster inclination angle (γ) as

$$\gamma_L = -\tan^{-1}\left(\frac{\omega_{Lx}}{\omega_{Lz}}\right) \quad (9)$$

$$\gamma_R = -\tan^{-1}\left(\frac{\omega_{Rx}}{\omega_{Rz}}\right) \quad (10)$$

In order to calculate the location of the arbitrary point (P) on kingpin axis, w.r.t the location of the wheel center (Ω), VI-CarRealTime applies rigid body motion equation

$$\vec{v}_\Omega + \vec{\omega} \wedge \vec{\Omega P} = 0 \quad (11)$$

Solving this equation yields:

$$\overrightarrow{\Omega P} = \frac{\vec{\omega} \wedge \vec{v}_\Omega}{\vec{\omega}^2} \quad (12)$$

Finally the algorithm transform the this location in the vehicle system reference.

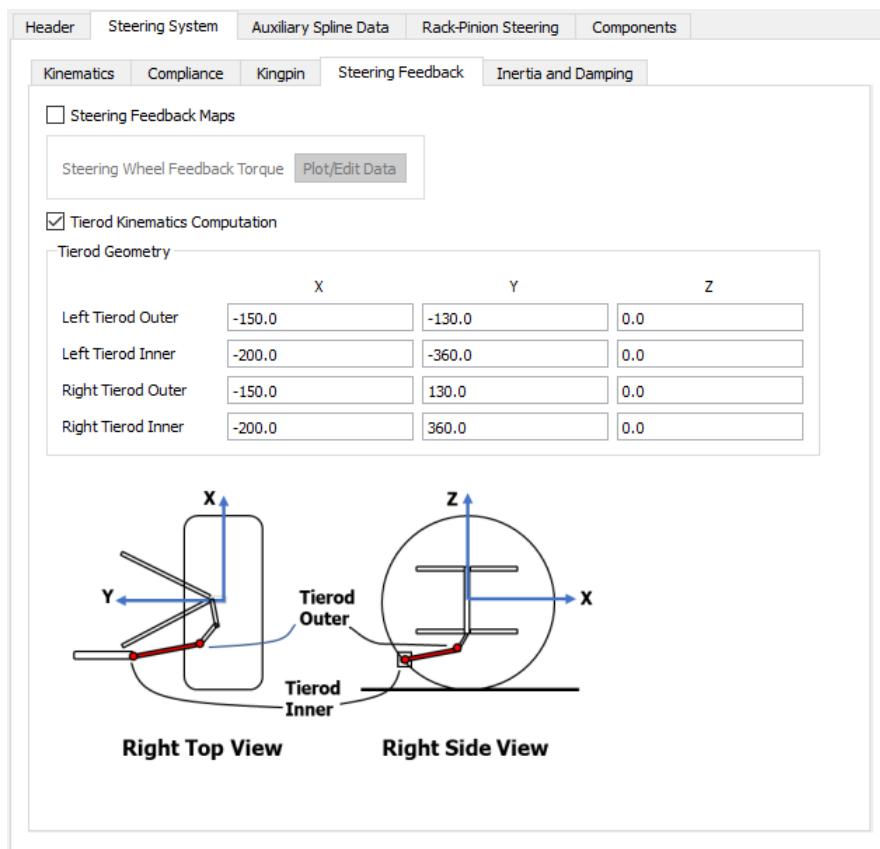
Note: The kingpin location data is not mandatory for defining the vehicle model since the steering behaviour is totally defined by suspension characteristics. It is useful to compute steering wheel torque effort, left and right tierod forces and rack forces.

In the Steering Feedback section of steering subsystem editor it is possible to define the way that VI-CarRealTime computes the steering wheel torque (or the rack force).

Two alternative methods are supported: the [Tierod geometry](#) based computation and the [Steering Feedback Maps](#); the switch between the two methodologies for basic steering model is selected using the **Steering Feedback Maps** toggle.

When active, the Tierod geometry based methodology supports the input of the geometry of the left and right tie-rod in terms of locations of their inner and outer hardpoints.

Using the Tierod computation method, the steering torque is computed under the assumption of ideal mechanical transmission of motion eventually including steering system compliance, from steering wheel to tierods. Effects coming from elements such as friction or steering assist are not considered. For including them it is required either to enter a [Steering Feedback Map](#) or to activate the [Rack-Pinion Steering](#) model.



The **Tierod Geometry** exposes the following fields:

- **Left Tierod Outer**
X,Y,Z coordinates of the left tie-rod outer hardpoint, with respect to a reference frame located in the wheel center and oriented as [vehicle reference frame](#).
- **Left Tierod Inner**

X,Y,Z coordinates of the left tie-rod inner hardpoint, with respect to a reference frame located in the wheel center and oriented as [vehicle reference frame](#).

- **Right Tierod Outer**

X,Y,Z coordinates of the right tie-rod outer hardpoint, with respect to a reference frame located in the wheel center and oriented as [vehicle reference frame](#).

- **Right Tierod Inner**

X,Y,Z coordinates of the right tie-rod inner hardpoint, with respect to a reference frame located in the wheel center and oriented as [vehicle reference frame](#).

The **Tierod Kinematics Computation** toggle button allows the user to switch on or off the Tierod Geometry:

- When tie-rod geometry flag is active the instantaneous torque (M_K) acting about the instantaneous kingpin axis is put in equilibrium with the force acting along the tie-rod direction, according to the following equation:

$$M_K = r \times F_t$$

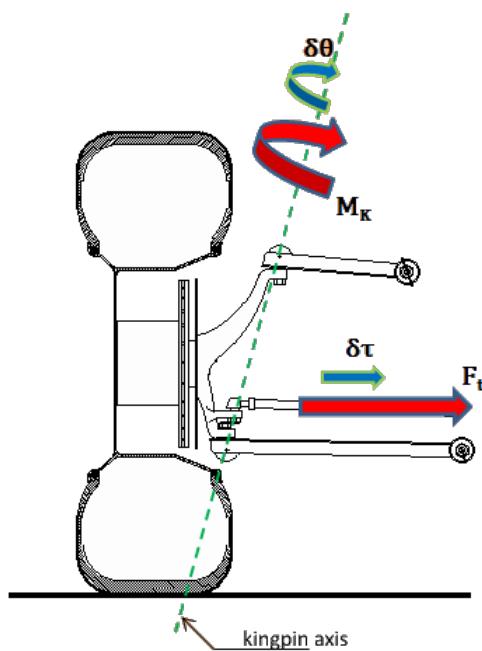
where r is the distance between the instantaneous kingpin axis and the tie-rod outer point. The X,Y,Z components of the tie-rod force are evaluated by projecting F_t vector with respect to the [vehicle location](#) reference system.

- When tie-rod geometry flag is inactive only the Y (in [vehicle location](#) reference system) component is computed (assuming that the rack direction is coincident with the chassis Y); in this case the tie-rod force is computed by using the Principle of Virtual Work, as explained below.

$$M_K \cdot \partial\theta = F_t \cdot \partial\tau$$

where:

- M_K is the moment about the kingpin axis (Kingpin Moment);
- θ is the rotation angle about the kingpin axis and it is a function of the suspension jounce and the rack displacement;
- F_t is the tie-rod force;
- τ is the rack displacement (displacement along the Y direction of the [vehicle reference system](#)).



The tie-rod force is then calculated as:

$$F_t = M_K \cdot \frac{\partial \theta}{\partial \tau}$$

The same applies for both left and right side of the suspension, so that:

$$F_{tL} = M_{K_L} \cdot \frac{\partial \theta_L}{\partial \tau}$$

$$F_{tR} = M_{K_R} \cdot \frac{\partial \theta_R}{\partial \tau}$$

For both cases (tie-rod geometry flag active/inactive) the rack force (F_{rack}) is evaluated as the sum of the left and right tie-rod force:

$$F_{rack} = F_{tL} + F_{tR}$$

The rack force is a function of the suspension jounce (left/right), kingpin moment (left/right) and rack displacement.

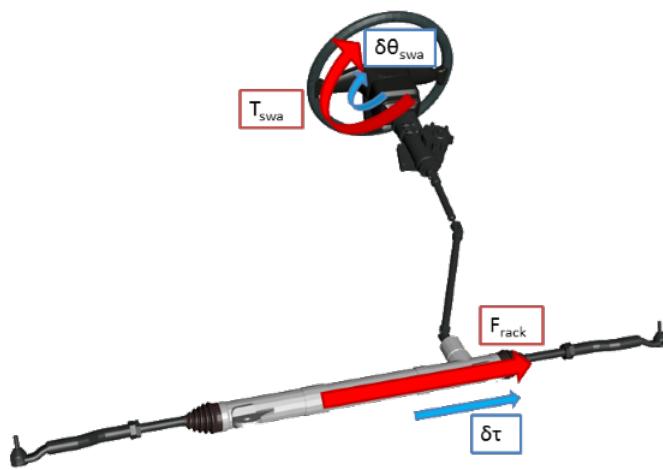
The rack force is used to evaluate the **Steering Wheel Torque** according to the following expression:

$$T_{swa} \cdot \partial \theta_{swa} = F_{rack} \cdot \partial \tau$$

where:

- T_{swa} is the Steering Wheel Torque;
- θ_{swa} is the rotation angle of the steering wheel angle;
- F_{rack} is the rack force;

- τ is the rack displacement (displacement along the Y direction of the [vehicle reference system](#)).



The Steering Wheel Torque is then computed as:

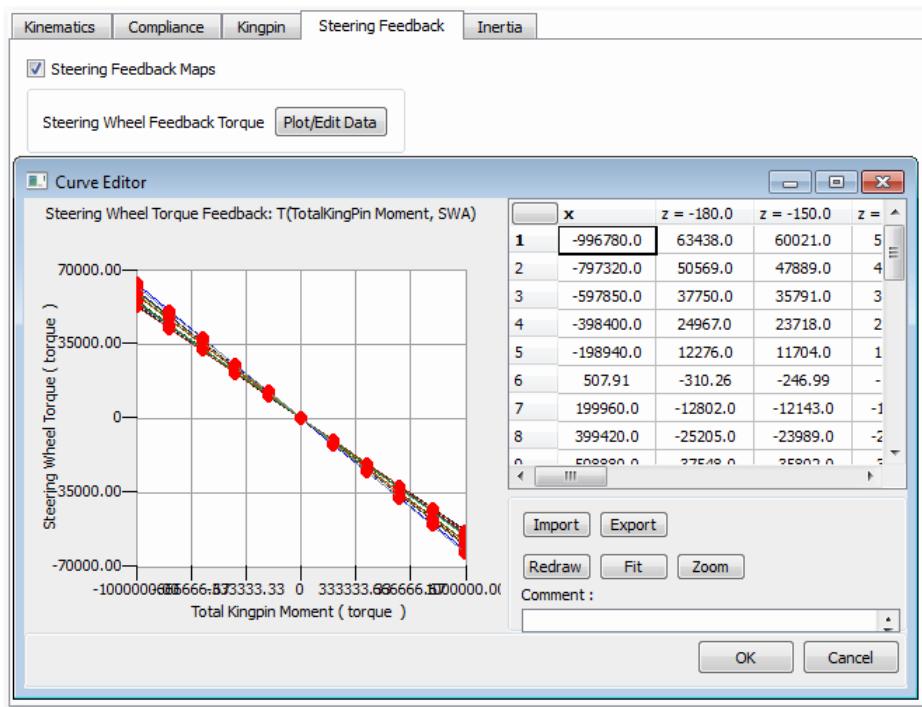
$$T_{swa} = F_{rack} \cdot \frac{\partial \tau}{\partial \theta_{swa}}$$

The Steering Wheel Torque, as the rack force, depends on suspension jounce (left/right), Kingpin moment (left/right) and rack displacement.

As an alternative to the computation of steering wheel torque based on the Tierod geometry, steering wheel torque can be computed in VI-CarRealTime as depending on total kingpin moment and steering wheel angle or rack travel.

It is implemented using a 3D map expressing the torque effort on steering wheel (or rack force) as a function of total kingpin moment (X) and steering displacement input (Y).

The switch between the maps and Tierod based computation is done using the **Steering Feedback Maps** toggle.



Note: In case of steering actuation through rack displacement the rack force map can be supplied as well. It is also possible to include the effect of a [simplified servo system](#) model in the conceptual steering model.

When a detailed model of mechanical and servo system is required it is possible to use the [Rack-Pinion Steering](#) model.

The steering inertia relevant when the steering release maneuvers are performed.

- **Steering Inertia**

equivalent moment of inertia (I) of the steering column along the steering wheel angle rotation axis.

- **Steering Damping**

equivalent rotational damping (b) of the steering column along the steering wheel angle rotation axis.

Steering Inertia	20000.0
Steering Damping	0.0

The following differential equation is used to compute steering wheel angle during steering release maneuvers:

$$I\ddot{\theta} + b\dot{\theta} = T_{steer}$$

where:

θ is the steering wheel angle

T_{steer} is the steering torque

Auxiliary Spline Data

In Auxiliary Spline Data table the following curve is stored:

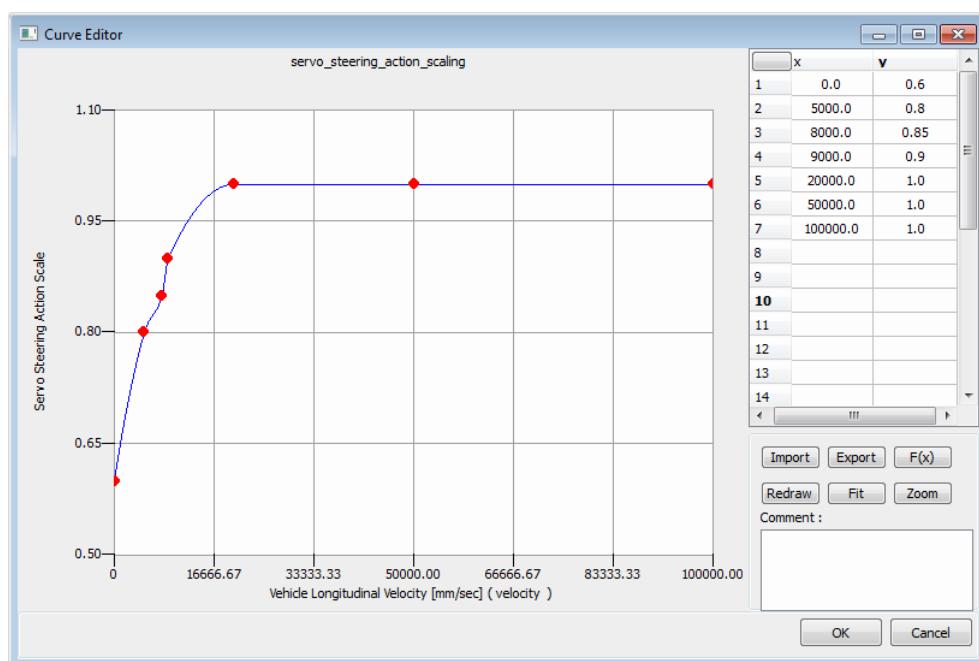
- **servo_steering_action_scaling**

it is a 2D spline which characterizes the servo steering action for the basic steering model as a function of the vehicle longitudinal velocity.

$$\text{servo_steering_action_scaling} = f(\text{vehicle_speed})$$

The servo steering curve map, is active only when the advanced steering model is disabled and it is used to scale through specific factor, depending on vehicle velocity, the torque request at steering wheel. As default the factors are set to 1.0 so it is equivalent to the case when servo assist is disabled.

The steering wheel torque output is computed by the product of the mechanical steering torque and the scaling factor.



Optionally, in Auxiliary Spline Data table the following curves can be stored when exporting models from Adams Car and when specific mapping has been realized in the suspension/steering template:

Header	Steering System	Auxiliary Spline Data	Rack-Pinion Steering	Components
Name Filter *				
Name	Spline	Comment		
left_tierod_force	Plot/Edit Data			
right_tierod_force	Spline			
servo_steering_action_scaling	Spline			

- **left_tierod_forces**

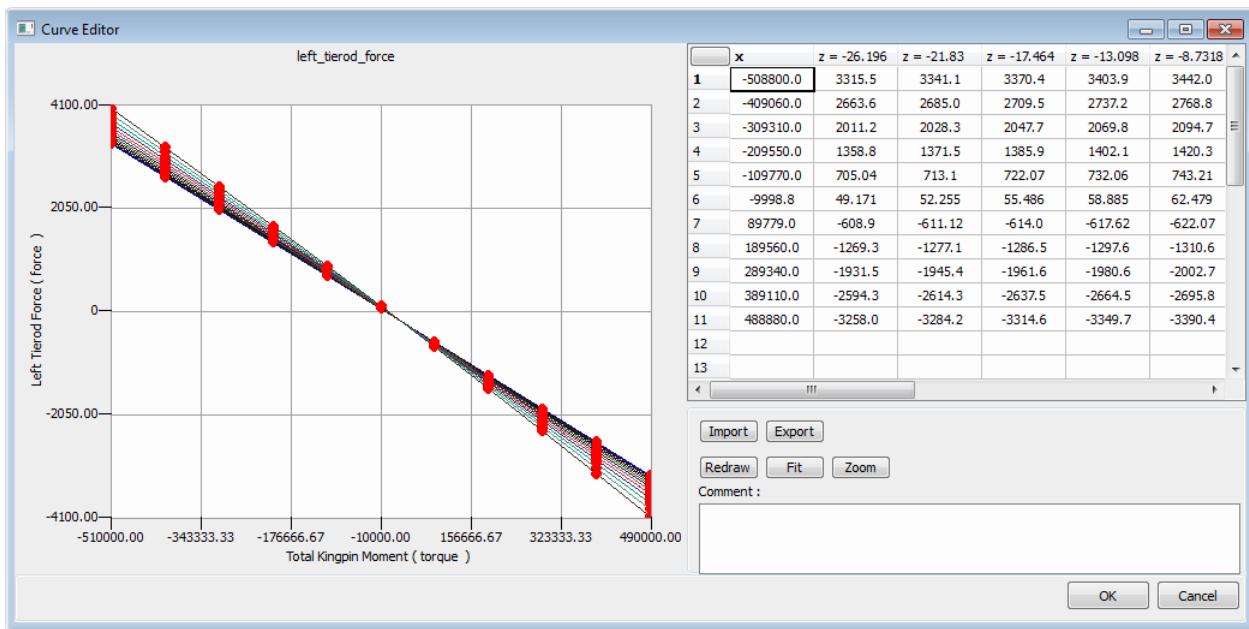
it is a 3D spline which characterizes the left tie-rod force as a function of kingpin moment and rack displacement.

$$\text{left_tierod_force} = f(\text{kingpin_moment_left}, \text{rack_displacement})$$

- **right_tierod_forces**

it is a 3D spline which characterizes the right tie-rod force as a function of kingpin moment and rack displacement.

$$\text{right_tierod_force} = f(\text{kingpin_moment_right}, \text{rack_displacement})$$



Rack-Pinion Steering

A rack-pinion steering model is available in steering subsystem.

The model can be optionally activated and consists of two modules:

- mechanical module
- power assistance module

The aim of mechanical module is to describe the behavior of all the mechanical part of a rack-pinion steering: rack, pinion, upper and lower steering column, torsion bar, hardy disk and mesh.

The power assistance module models two different types of steering assistance: electric power steering assistance ([EPS](#)) and hydraulic power steering assistance ([HPS](#)).

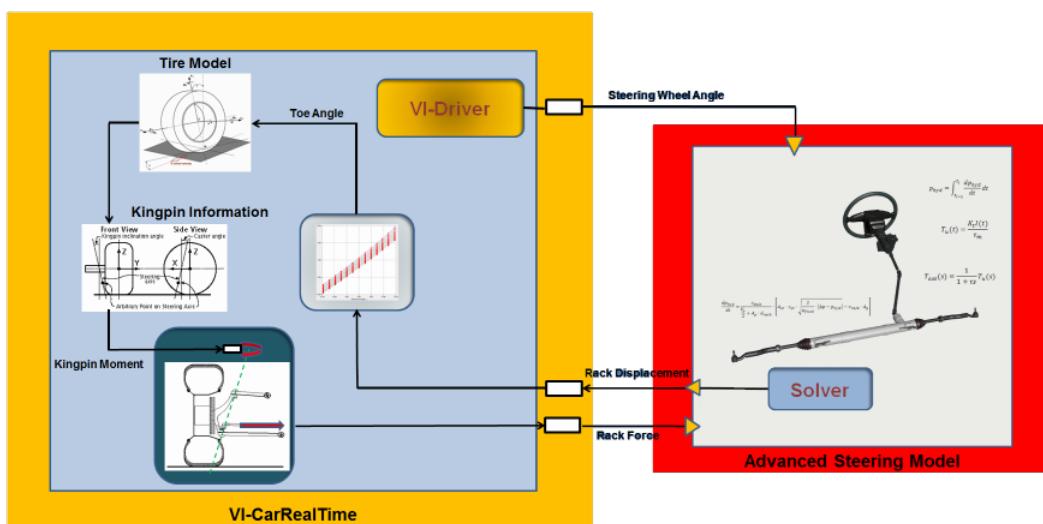
The rack-pinion steering allows to compute the steering wheel torque with higher fidelity with respect to the basic steering because the steering mechanism dynamic contribute is now taken into account: using the basic steering model, steering wheel torque is computed either through the tierod geometry or instantaneous kingpin axis.

Furthermore, rack-pinion model allows the user to define his own power steering assist logic to control the driver steering wheel effort.

The rack-pinion steering model requires kinematic input for *steering wheel angle* and it is capable of computing the resulting *rack displacement* which is used to drive the vehicle model.

VI-Driver can be interfaced with the advanced steering model, supporting both open or closed loop steering demands.

In order to function properly the VI-CarRealTime model needs to be created including the rack kinematics data. In fact by using the schema shown in the figure below, it is possible to decouple the behavior of the steering system from the front suspension steering kinematics.



VI-CarRealTime computes the steering wheel angle demand by using VI-Driver (in open or closed loop) and it communicates it to the advanced steering model which computes the rack displacement including all the steering line dynamics (friction, inertia etc.) and sends the information back to VI-CarRealTime which uses it to look up the suspension kinematic curves.

VI-CarRealTime also features the possibility of computing tierod and rack forces depending on steering geometry and kingpin moment and to feed it back to the advanced steering model for a comprehensive steering dynamics.

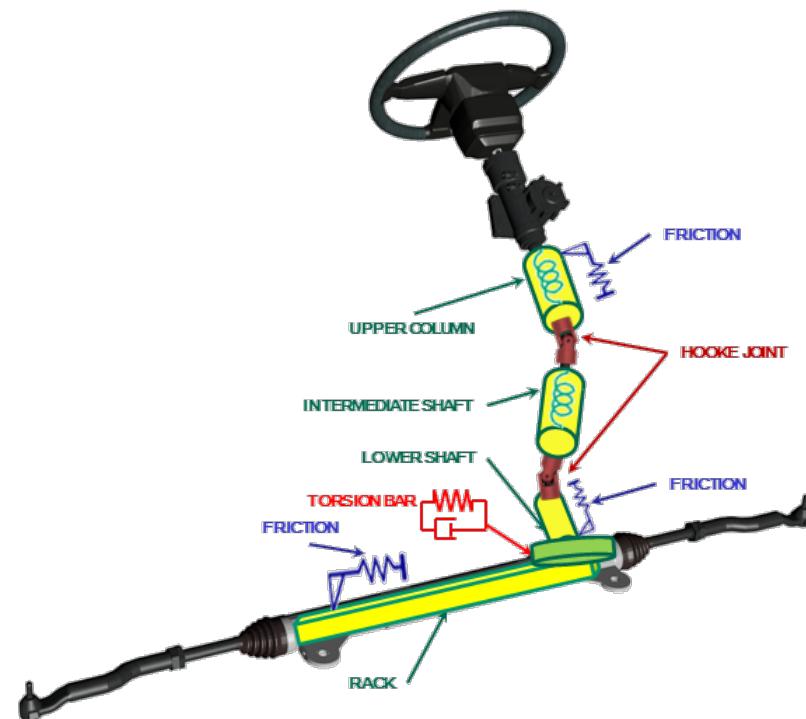
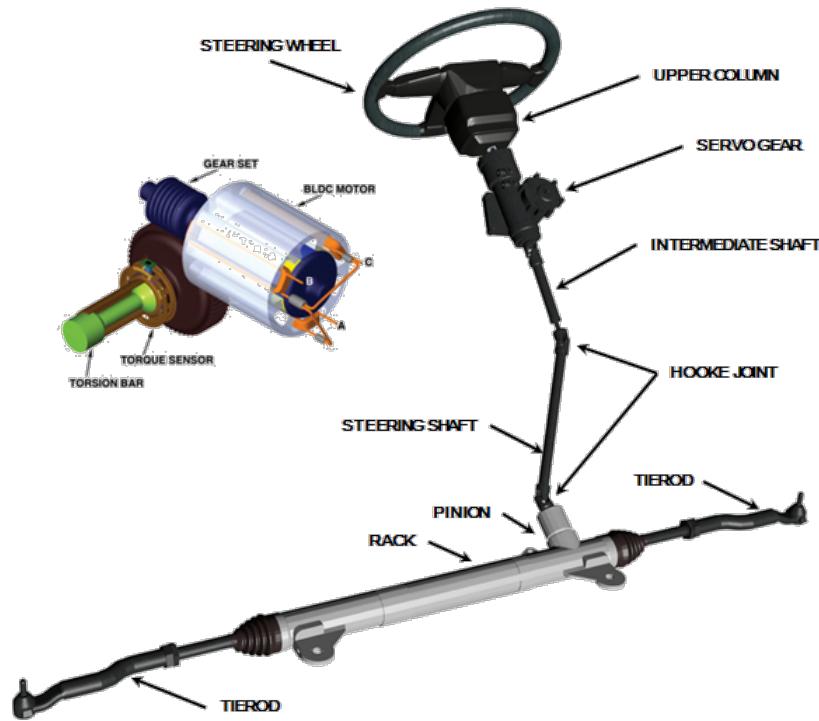
The rack-pinion steering model interface is exposing the following panels:

- [Mechanical Properties](#)
- [Steering Assist](#)
- [Solver](#)

The outputs of the rack-pinion steering module are described in the [Output Channels](#) documentation.

The rack-pinion steering model includes a mechanical model featuring friction elements and an electric power steering module.

The mechanical model of the steering system features a minimum of two degrees of freedom. One is representing the inertia of steering wheel down to the torsion bar and the other is the steering rack with pinion. Additional compliance effects (and relative degrees of freedom) for steering column, torsion bar, hardy disc and mesh can be optionally activated and are represented via linear stiffness; steering column and torsion bar also include linear damping.



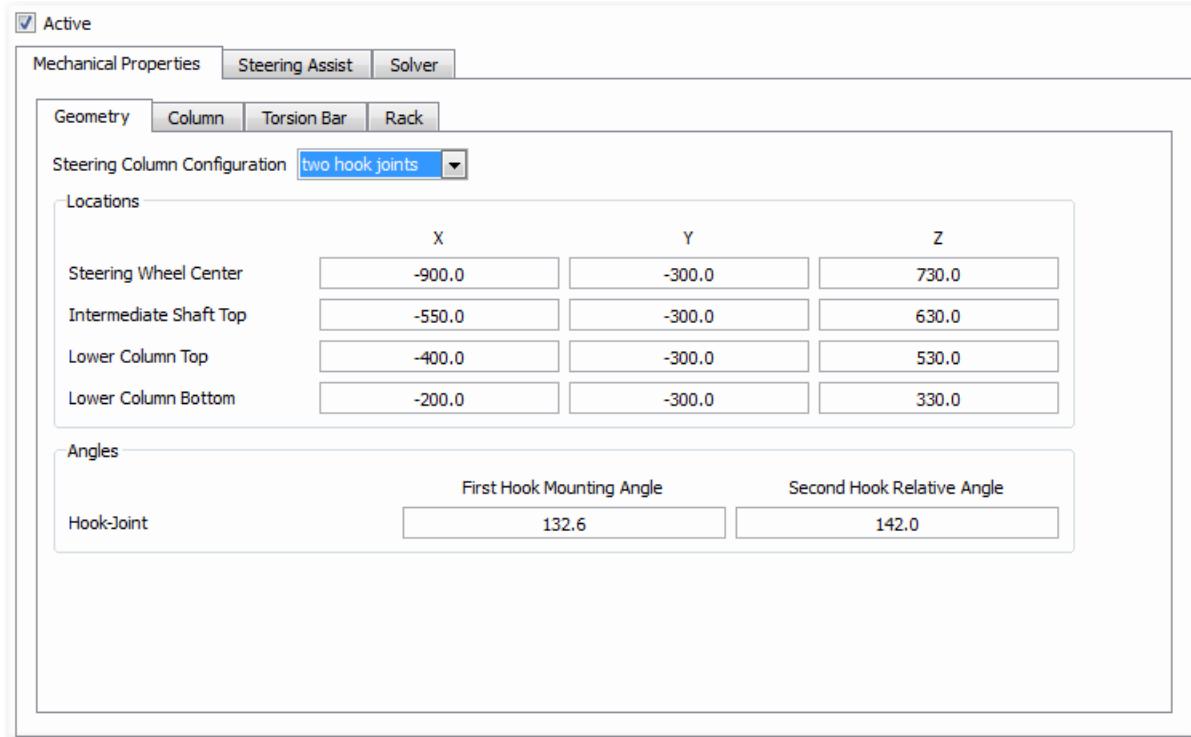
All frictional effects above and below the torsion bar are modeled in combination by using two types of [friction elements](#).

The mechanical properties to be entered in order to set-up the rack-pinion steering subsystem are stored in the following panels:

- [Geometry](#)

- [Column](#)
- [Torsion Bar](#)
- [Rack](#)

The geometry panel exposes the parameters needed to define the layout of the rack-pinion steering.



Three different steering column configurations are available:

- **two hook joints**

Two hook joints are mounted as shown in the image below. The first ([First Hook Mounting Angle](#)) is between upper column part and intermediate column part; the second ([Second Hook Relative Angle](#)) is between intermediate column part and lower column part.

- **two convel joints**

Two constant velocity joints are mounted between the same part as per two hook joints configuration. No need to specify mounting angles for convel joints.

- **three hook joints**

Three hook joints are mounted. The first ([First Hook Mounting Angle](#)) is between upper column part and first intermediate column part; the second ([Second Hook Relative Angle](#)) is between first intermediate column part and second intermediate column part. The third ([Third Hook Relative Angle](#)) is between second intermediate column part and lower column part.

The *Locations* of rack-pinion steering to enter are the following, please refer to following image for parameter definition:

- **Steering Wheel Center**

X,Y,Z location of steering wheel center (*point O*).

- **Intermediate Shaft Top**

X,Y,Z location of the intermediate shaft top (*point A*). It's active for two hook/convvel joints configuration.

- **First Intermediate Shaft Top**

X,Y,Z location of the first intermediate shaft top point. It's active for three hook joints configuration.

- **Second Intermediate Shaft Top**

X,Y,Z location of the second intermediate shaft top point. It's active for three hook joints configuration.

- **Lower Column Top**

X,Y,Z location of the lower column top (*point B*).

- **Lower Column Bottom**

X,Y,Z location of the lower column bottom (*point C*).

Note: there is not a privileged reference system with respect to which the steering coordinates must be entered. Advanced steering model is a separated module with respect to VI-CarRealTime model, so its coordinates are not strictly related to VI-CarRealTime reference system.

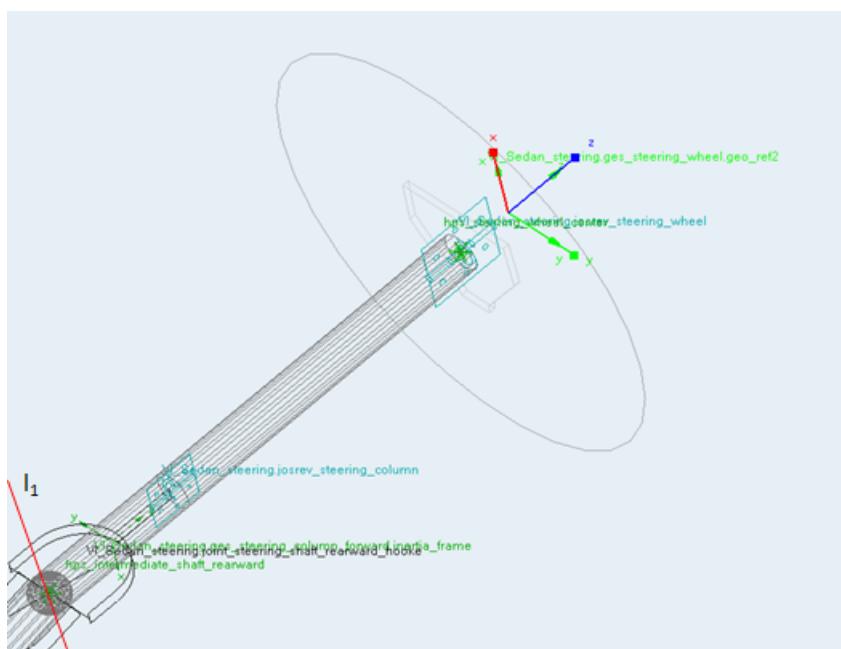
The Angles to enter are:

- **First Hook Mounting Angle**

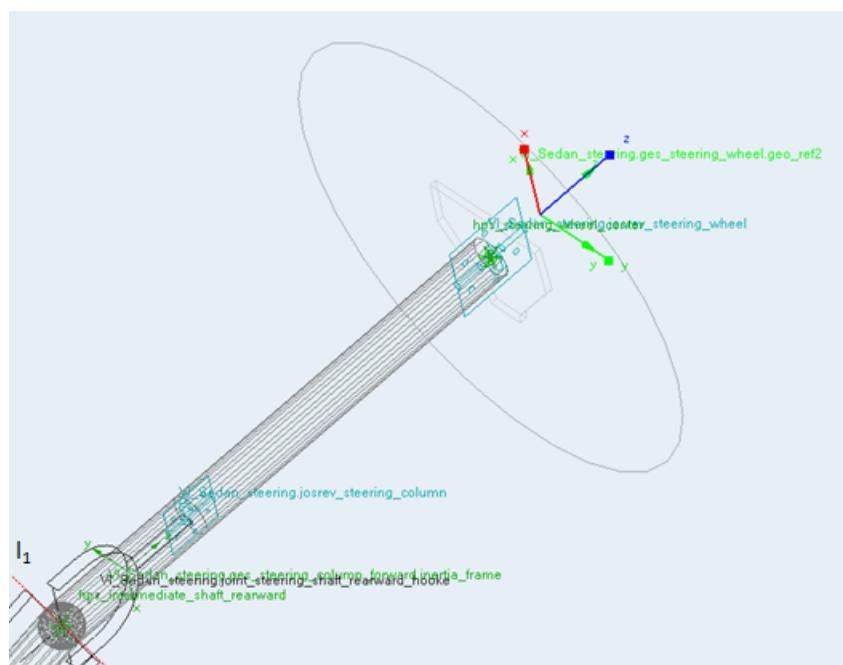
Considering a reference system having Z as shaft direction and Y as 3 to 9 direction of steering wheel angle, hook mounting angle is the angle (θ_1) between the Y axis of the system reference and the axis (line l_1) of the hook joint yoke associated to the shaft. Line l_1 is the perpendicular direction to hook fork.

Angle is positive counter-clockwise when viewed from the driver's point of view.

Example 1: according to the definition, the mounting angle represented in the picture is 90 degree (l_1 parallel to X direction of reference system).



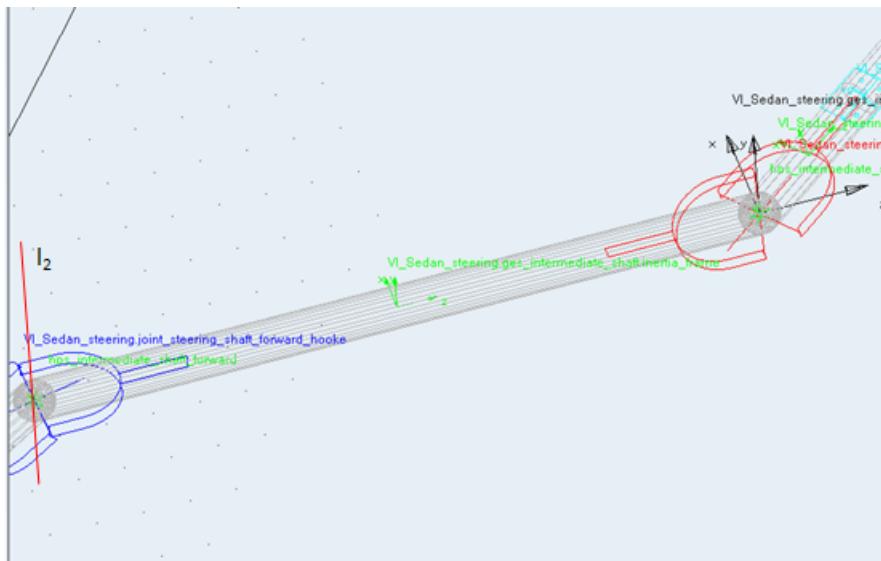
Example 2: according to the definition, the mounting angle represented in the picture below is 0 degree (l_1 parallel to Y direction of reference system).



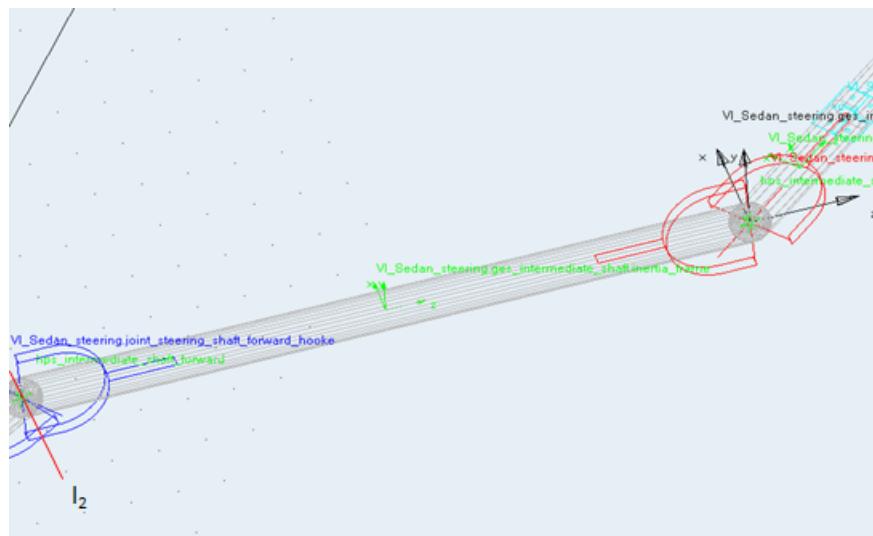
- **Second Hook Relative Angle**

Considering a reference system located on mounting hook position having Z as intermediate shaft direction and Y as hook fork direction, second hook relative angle is the angle (θ_2) between the Y axis of the system reference and the axis (line l_2) of the hook joint yoke associated to the intermediate shaft. Line l_1 is the perpendicular direction to hook fork. Angle is positive counter-clockwise when viewed from the driver's point of view.

Example 1: according to the definition, the second hook relative angle represented in the picture is 0 degree (l_2 parallel to Y direction of reference system).



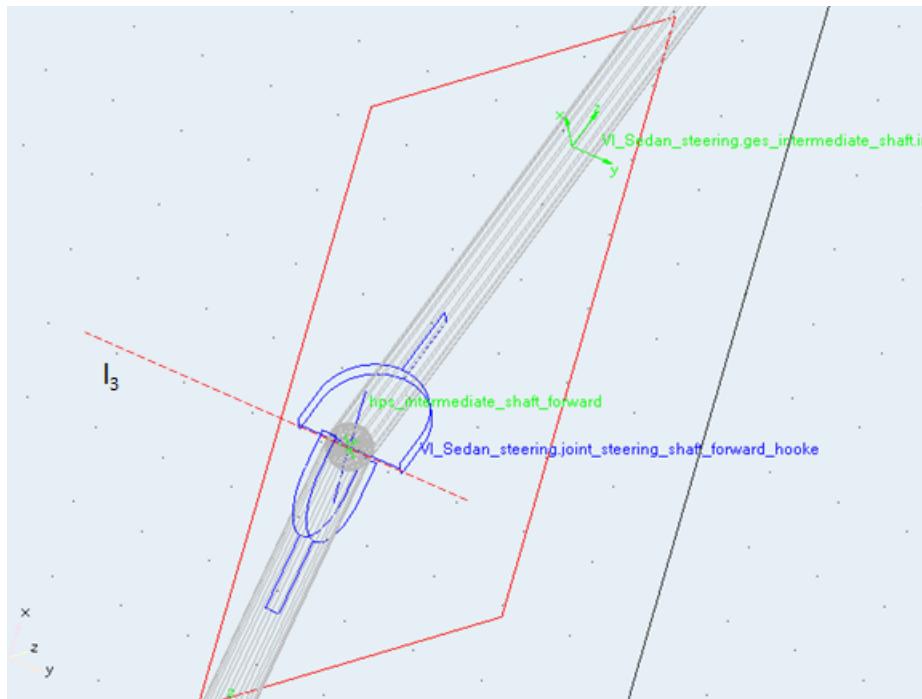
Example 2: according to the definition, the second hook relative angle represented in the picture is 90 degree (l_2 parallel to X direction of reference system). When hook joint relative angle (θ_2) is 90 degrees, for each value of hook joint mounting angle (θ_1), the hook joints yokes which belong to the intermediate shaft have the same direction.



- **Third Hook Relative Angle**

The angle (θ_3) is measured with respect to a plane containing both lower intermediate shaft axis and lower column shaft axis, both attached to the third hook joint; the fork line I_3 of yoke attached to the lower intermediate shaft. Positive counter-clockwise when viewed from the driver's point of view.

Example: according to the definition, the angle represented in the picture is 90 degree (I_3 is the normal of red plane). Otherwise, when line I_3 belongs to the plane, the angle will be 0.



Note: when hook joint relative angle (θ_2) is 90 degrees, for each value of hook joint mounting angle (θ_1), the hook joints yokes which belong to the intermediate shaft have the same direction.

The following panel is used to set up the steering column parameters.

The steering column consists of

- steering wheel plus three shafts (**upper**, **intermediate** and **lower**) for [two_hook/convel_joints](#) configuration.

Inertia	
Steering Wheel Inertia	10000.0
Upper Column Inertia	3333.33333333
Intermediate Column Inertia	3333.33333333
Lower Column Inertia	3333.33333333

- steering wheel plus four shafts (**upper**, **first intermediate**, **second intermediate** and **lower**) for [three hook joints](#) configuration.

Inertia	
Steering Wheel Inertia	10000.0
Upper Column Inertia	3333.33333333
First Intermediate Column Inertia	3333.33333333
Second Intermediate Column Inertia	3333.33333333
Lower Column Inertia	3333.33333333

Active

Mechanical Properties		Steering Assist	Solver																																																						
<input type="button" value="Geometry"/>	<input type="button" value="Column"/>	<input type="button" value="Torsion Bar"/>	<input type="button" value="Rack"/>																																																						
<table border="1"> <thead> <tr> <th colspan="2">Inertia</th> </tr> </thead> <tbody> <tr> <td>Steering Wheel Inertia</td> <td>10000.0</td> </tr> <tr> <td>Upper Column Inertia</td> <td>3333.33333333</td> </tr> <tr> <td>Intermediate Column Inertia</td> <td>3333.33333333</td> </tr> <tr> <td>Lower Column Inertia</td> <td>3333.33333333</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="2">Dynamics</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> Rigid</td> <td>Upper Column Stiffness</td> <td>7000.0</td> <td>Upper Column Damping</td> <td>50.0</td> </tr> <tr> <td><input type="checkbox"/> Rigid</td> <td>Lower Column Stiffness</td> <td>7000.0</td> <td>Lower Column Damping</td> <td>50.0</td> </tr> <tr> <td><input type="checkbox"/> Rigid</td> <td>Hardy Disk Stiffness</td> <td>4880.0</td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="2">Friction</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> Active</td> <td></td> </tr> <tr> <td>Stiffness</td> <td>870.0</td> </tr> <tr> <td>Minimum Torque</td> <td>-100.0</td> </tr> <tr> <td>Maximum Torque</td> <td>100.0</td> <td><input type="checkbox"/> Coulomb Friction</td> </tr> <tr> <td><input type="checkbox"/> Maxwell Element</td> <td>Modified Formulation</td> <td>Coulomb Static Mu</td> <td>0.0</td> </tr> <tr> <td>Maxwell Element Stiffness</td> <td>0.0</td> <td>Coulomb Dynamic Mu</td> <td>0.0</td> </tr> <tr> <td>Maxwell Element Damping</td> <td>0.0</td> <td>Coulomb Max. Torque</td> <td>0.0</td> </tr> <tr> <td>Maxwell Element Maximum Torque</td> <td>0.0</td> <td>Coulomb Transition Vel.</td> <td>0.0</td> </tr> </tbody> </table>				Inertia		Steering Wheel Inertia	10000.0	Upper Column Inertia	3333.33333333	Intermediate Column Inertia	3333.33333333	Lower Column Inertia	3333.33333333	Dynamics		<input checked="" type="checkbox"/> Rigid	Upper Column Stiffness	7000.0	Upper Column Damping	50.0	<input type="checkbox"/> Rigid	Lower Column Stiffness	7000.0	Lower Column Damping	50.0	<input type="checkbox"/> Rigid	Hardy Disk Stiffness	4880.0			Friction		<input checked="" type="checkbox"/> Active		Stiffness	870.0	Minimum Torque	-100.0	Maximum Torque	100.0	<input type="checkbox"/> Coulomb Friction	<input type="checkbox"/> Maxwell Element	Modified Formulation	Coulomb Static Mu	0.0	Maxwell Element Stiffness	0.0	Coulomb Dynamic Mu	0.0	Maxwell Element Damping	0.0	Coulomb Max. Torque	0.0	Maxwell Element Maximum Torque	0.0	Coulomb Transition Vel.	0.0
Inertia																																																									
Steering Wheel Inertia	10000.0																																																								
Upper Column Inertia	3333.33333333																																																								
Intermediate Column Inertia	3333.33333333																																																								
Lower Column Inertia	3333.33333333																																																								
Dynamics																																																									
<input checked="" type="checkbox"/> Rigid	Upper Column Stiffness	7000.0	Upper Column Damping	50.0																																																					
<input type="checkbox"/> Rigid	Lower Column Stiffness	7000.0	Lower Column Damping	50.0																																																					
<input type="checkbox"/> Rigid	Hardy Disk Stiffness	4880.0																																																							
Friction																																																									
<input checked="" type="checkbox"/> Active																																																									
Stiffness	870.0																																																								
Minimum Torque	-100.0																																																								
Maximum Torque	100.0	<input type="checkbox"/> Coulomb Friction																																																							
<input type="checkbox"/> Maxwell Element	Modified Formulation	Coulomb Static Mu	0.0																																																						
Maxwell Element Stiffness	0.0	Coulomb Dynamic Mu	0.0																																																						
Maxwell Element Damping	0.0	Coulomb Max. Torque	0.0																																																						
Maxwell Element Maximum Torque	0.0	Coulomb Transition Vel.	0.0																																																						

The *Inertia* parameters are:

- Steering Wheel Inertia**
moment of inertia along the axis of rotation of the steering wheel.
- Upper Column Inertia**
moment of inertia along the axis of rotation of the upper column shaft.
- Intermediate Column Inertia**

moment of inertia along the axis of rotation of the intermediate column shaft.

- **Lower Column Inertia**

moment of inertia along the axis of rotation of the lower column shaft.

The steering shaft can optionally include stiffness elements. When option is activated the corresponding shaft is split into two parts having relative rotational degree of freedom and half inertia (each) of the values input through the column inertia parameter.

An additional compliance element can be added to the steering line through the hardy disk element which is typically introduced to remove unwanted torsional vibration of the steering shafts. In the model the hardy disk is located at the lower end of the lower column.

Stiffness values of the steering column and hardy disk are introduced by means of the following fields:

- **Upper Column Stiffness**

stiffness of the upper part of the steering column (compliance is enabled when **Rigid** toggle button next to the field is unchecked).

- **Lower Column Stiffness**

stiffness of the lower part of the steering column (compliance is enabled when **Rigid** toggle button next to the field is unchecked).

- **Hardy Disk Stiffness**

stiffness of the hardy disk (compliance is enabled when **Rigid** toggle button next to the field is unchecked).

The upper and lower steering column parts may include a rotational damping element which is activated when the corresponding shaft is not set to rigid. *Damping* values of the steering column that can be specified are:

- **Upper Column Damping**

damping of the upper part of the steering column (enabled when **Rigid** toggle button next to the corresponding Stiffness field is unchecked).

- **Lower Column Damping**

damping of the lower part of the steering column (enabled when **Rigid** toggle button next to the corresponding Stiffness field is unchecked).

The upper steering column features the possibility of including friction effect in the *Friction* group box through the following parameters:

- **Stiffness**

Friction stiffness coefficient k_{SF} of the [ESF](#) model.

- **Maximum Torque**

maximum torque that the [ESF](#) friction of the steering column can generate

- **Minimum Torque**

minimum torque that the [ESF](#) friction of the steering column can generate.

- **Maxwell Element Activity Check**

Check box for the activation of the Maxwell element feature in the column friction.

- **Maxwell Element Modified Formulation Check**

Check box for the activation of the modified formulation in the Maxwell element of the column friction (including no stiffness dependency and velocity dependency based on hyperbolic tan function).

- **Maxwell Element Stiffness**

torsion spring stiffness k_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Damping**

Torsion damping coefficient c_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Maximum Torque**

Maximum value of the torque $T_{M,lim}$ of the [ESF+Maxwell](#) model.

- **Coulomb Friction Activity Check**

Check box for the activation of the pseudo-Coulomb friction component.

- **Coulomb Friction Static Mu**

Pseudo-Coulomb friction static coefficient.

- **Coulomb Friction Dynamic Mu**

Pseudo-Coulomb friction dynamic coefficient.

- **Coulomb Friction Transition Velocity**

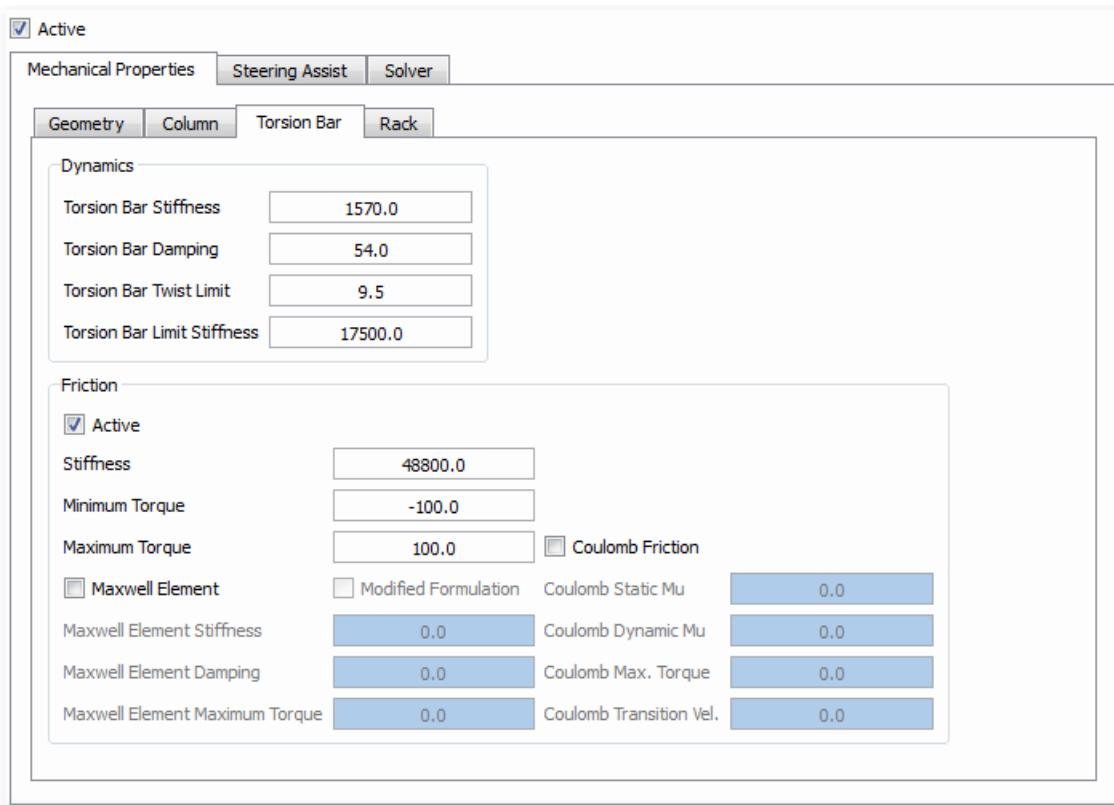
Transition rotational velocity for static to dynamic switch.

- **Coulomb Friction Max Torque**

Maximum Torque for pseudo Coulomb friction component.

The dialog box is used to set up the torsion bar parameters.

Torsion bar is modeled using a bi-linear spring element including linear damping. The change of slope of the stiffness characteristic is set by the torsion bar twist limit.



The *Dynamic* parameters of the torsion bar are:

- **Torsion Bar Stiffness**

stiffness of the torsion bar above the torsion bar twist limit.

- **Torsion Bar Twist Limit**

transition value of the torsion bar twist angle for the torsion bar stiffness (torsion bar limit stiffness is used for twist angles greater than the specified values) .

- **Torsion Bar Limit Stiffness**

stiffness of the torsion bar when the twist limit angle is exceeded.

- **Torsion Bar Damping**

damping of the torsion bar.

The torsion bar features the possibility of including friction effect in the *Friction* group box through following parameters:

- **Stiffness**

friction stiffness coefficient k_{SF} of the [ESF](#) model.

- **Maximum Torque**

maximum torque that the [ESF](#) friction of the torsion bar can generate.

- **Minimum Torque**

minimum torque that the [ESF](#) friction of the torsion bar can generate.

- **Maxwell Element Activity Check**

Check box for the activation of the Maxwell element feature in the rack friction.

- **Maxwell Element Modified Formulation Check**

Check box for the activation of the modified formulation in the Maxwell element of the torsion bar friction (including no stiffness dependency and velocity dependency based on hyperbolic tan function).

- **Maxwell Element Stiffness**

Torsion spring stiffness k_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Damping**

Torsion damping coefficient c_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Maximum Torque**

Maximum value of the torque $T_{M,lim}$ of the [ESF+Maxwell](#) model.

- **Coulomb Friction Activity Check**

Check box for the activation of the pseudo-Coulomb friction component.

- **Coulomb Friction Static Mu**

Pseudo-Coulomb friction static coefficient.

- **Coulomb Friction Dynamic Mu**

Pseudo-Coulomb friction dynamic coefficient.

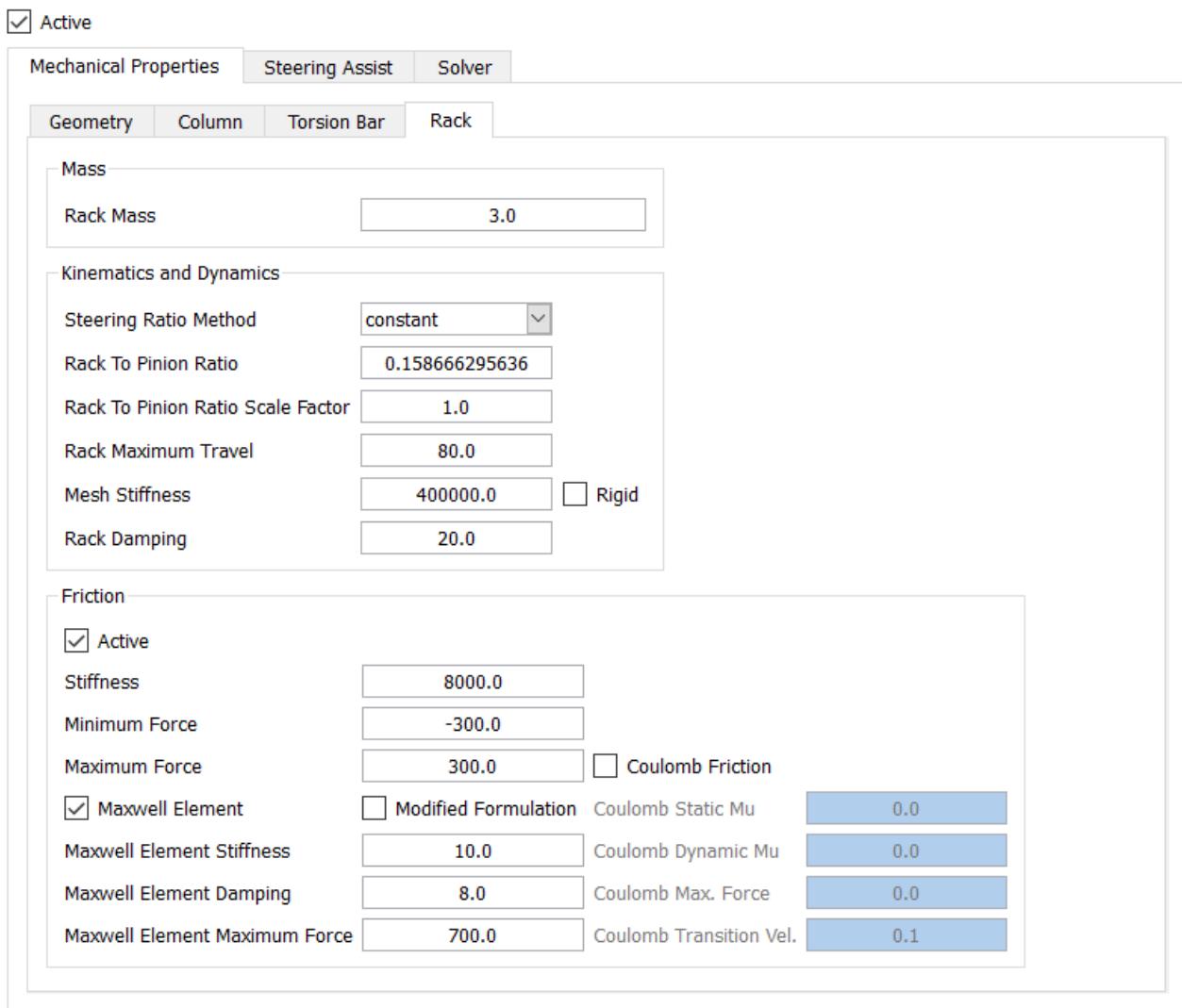
- **Coulomb Friction Transition Velocity**

Transition rotational velocity for static to dynamic switch.

- **Coulomb Friction Max Torque**

Maximum Torque for pseudo Coulomb friction component.

The following dialog box is used to set up the rack parameters.



- **Rack Mass**

mass of the rack part.

- **Steering ratio method**

option menu for constant or variable steering ration.

- **Rack To Pinion Ratio**

reduction ratio from rack displacement to pinion angle; it can be defined by a constant value or by a spline representing the rack to pinion ratio as function of steering wheel rotation.

- **Rack To Pinion Ratio Scale Factor**

scaling factor for the Rack To Pinion Ratio value.

- **Rack Maximum Travel**

maximum stroke of the rack part.

- **Mesh Stiffness**

stiffness of the contact between rack and pinion mesh. It can be optionally set to rigid.

- **Rack Damping**

translational damping of the rack part.

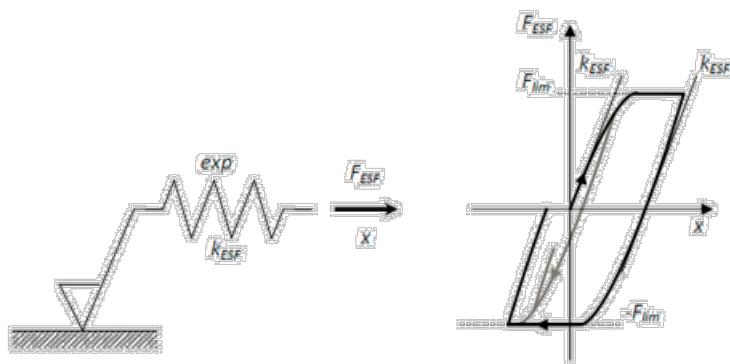
The *Friction* values of the rack are:

- **Stiffness**
friction stiffness coefficient k_{SF} of the [ESF](#) model.
- **Minimum Force**
minimum force the friction of the rack can generate.
- **Maximum Force**
minimum force the friction of the rack can generate.
- **Maxwell Element Activity Check**
check box for the activation of the Maxwell element feature in the rack friction.
- **Maxwell Element Modified Formulation Check**
check box for the activation of the modified formulation in the Maxwell element of the rack friction (including no stiffness dependency and velocity dependency based on hyperbolic tan function).
- **Maxwell Element Stiffness**
spring stiffness k_M of the [ESF+Maxwell](#) model.
- **Maxwell Element Damping**
damping coefficient c_M of the [ESF+Maxwell](#) model.
- **Maxwell Element Maximum Force**
maximum value of the force $F_{M,lim}$ of the [ESF+Maxwell](#) model.
- **Coulomb Friction Activity Check**
check box for the activation of the pseudo-Coulomb friction component.
- **Coulomb Friction Static Mu**
pseudo-Coulomb friction static coefficient.
- **Coulomb Friction Dynamic Mu**
pseudo-Coulomb friction dynamic coefficient.
- **Coulomb Friction Transition Velocity**
transition translational velocity for static to dynamic switch.
- **Coulomb Friction Max force**
maximum force for pseudo Coulomb friction component.

The steering model includes the following models implemented to represent the actual friction behavior of the system.

Exponential Spring Friction Element

In the ESF element the friction force F_{SF} is proportional to the **friction stiffness coefficient k_{SF}** until the friction limit force F_{lim} is reached. If the movement changes the direction at the reversal point, the friction force decreases until the minimum value of $-F_{lim}$.



The following equation describes the rising branch of the hysteresis loop.

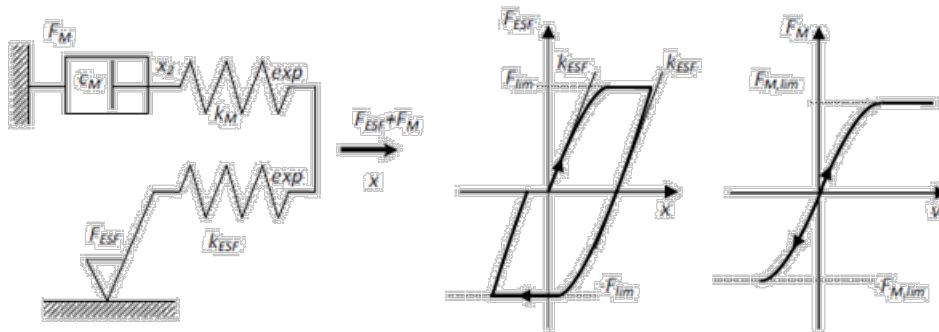
$$F_{SF} = F_{lim}(1 - e^{-f_{ESF} \cdot x})$$

The factor f_{ESF} can be calculated out of the stiffness at zero force. The ESF element needs as input parameters the stiffness at zero force k_{ESF} and the force limit F_{lim} .

$$f_{ESF} = \frac{k_{ESF}}{F_{lim}}$$

Exponential Spring Friction Element with parallel Maxwell element

The ESFM element is enhanced with a parallel nonlinear Maxwell-element to cover dynamic effects.

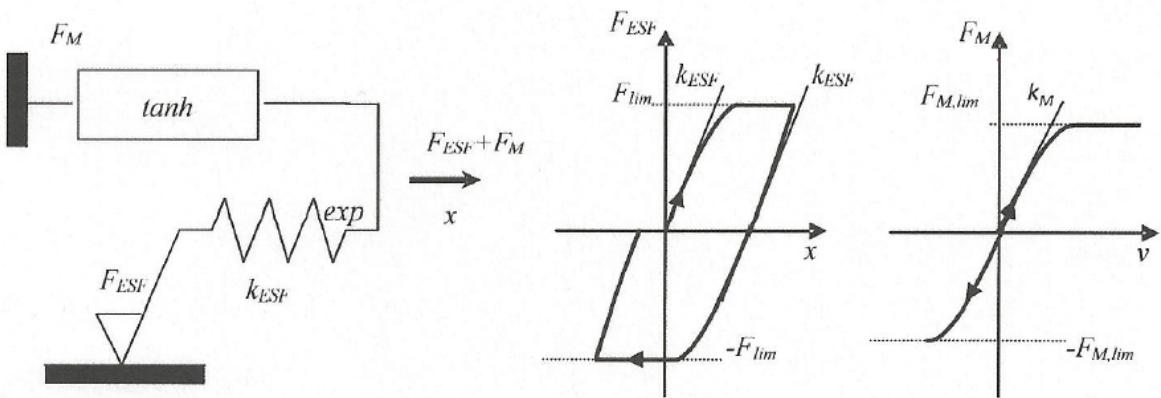


The spring in the Maxwell force path is modeled using the same exponential law as the spring in the ESF-friction element path to limit the dynamic effects at high speeds. The damper with **damping constant c_M** of the Maxwell-element reduces the spring force especially at low speeds. The whole friction force is the sum of both legs. The additional input elements of the Maxwell-element are the **spring stiffness k_M at zero force**, **maximum spring force $F_{M,lim}$** and the **linear damping coefficient c_M** .

A modified formulation for the Maxwell element is optionally available where the friction force is expressed by the equation:

$$F_M = F_{M,lim} \cdot \tanh(k_M \cdot v)$$

according to the following diagram:



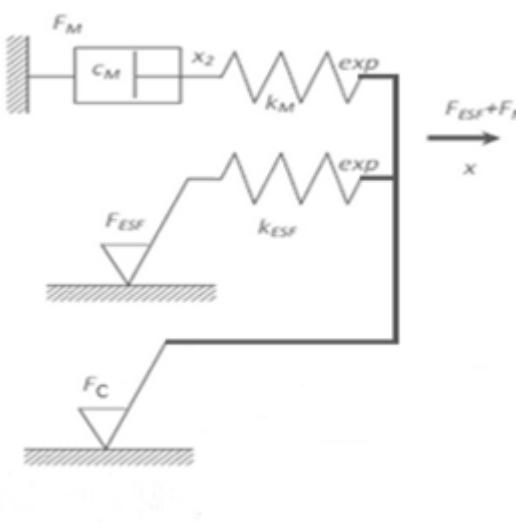
Pseudo Coulomb Friction Element

In order to consider load depending friction effects into elements (translational or rotational), a pseudo-coulomb type of friction is optionally supported where the friction force or torque has a linear dependency on the applied load according to the equation:

$$F_c = \begin{cases} \mu_d \cdot F_n & \text{if } V_s < -V_t \text{ or } V_s > V_t \\ \mu_s \cdot F_n & \text{if } -V_t \geq V_s \geq V_t \end{cases}$$

where:

- F_c is the pseudo Coulomb friction force;
- F_n is the element load;
- μ_d is the dynamic friction coefficient;
- μ_s is the static friction coefficient;
- V_s is the element relative velocity;
- V_t is the transition velocity between static and dynamic friction.



The steering assist represents the assistance to the mechanical module for reducing the steering wheel effort of the driver.

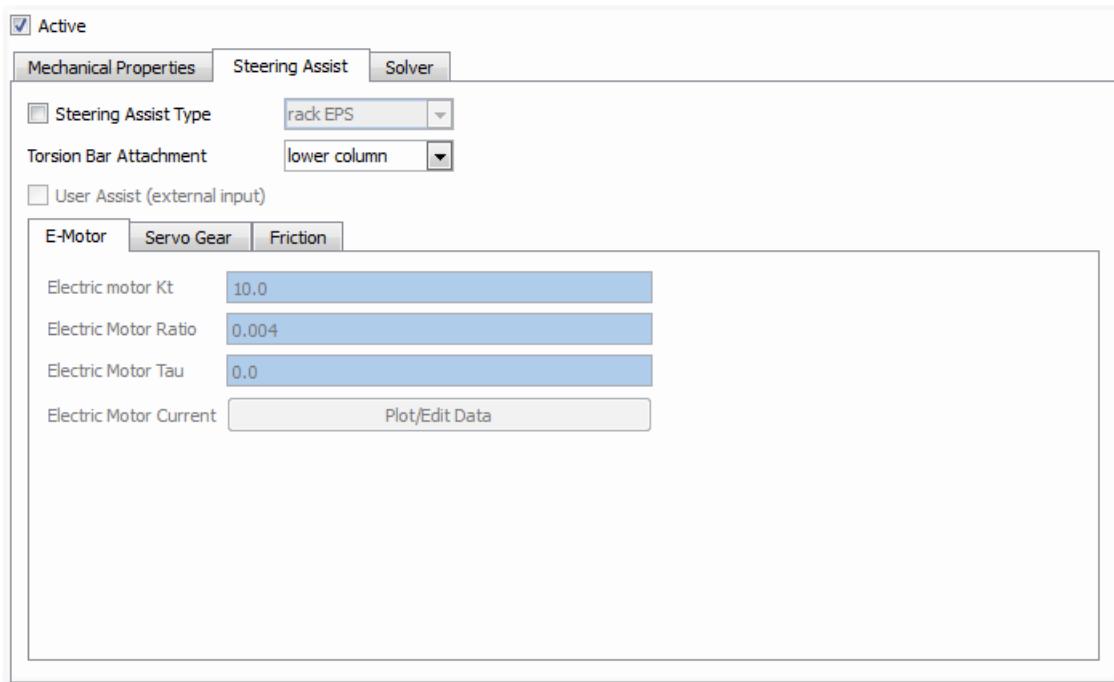
Two different steering assist types are available:

- Electrical Power Steering ([EPS](#))
- Hydraulic Power Steering ([HPS](#))

If **Steering Assist Type** toggle button is unchecked the following parameter is required:

- **Torsion Bar Attachment**

define the part the torsion bar is attached to (lower column / upper column).



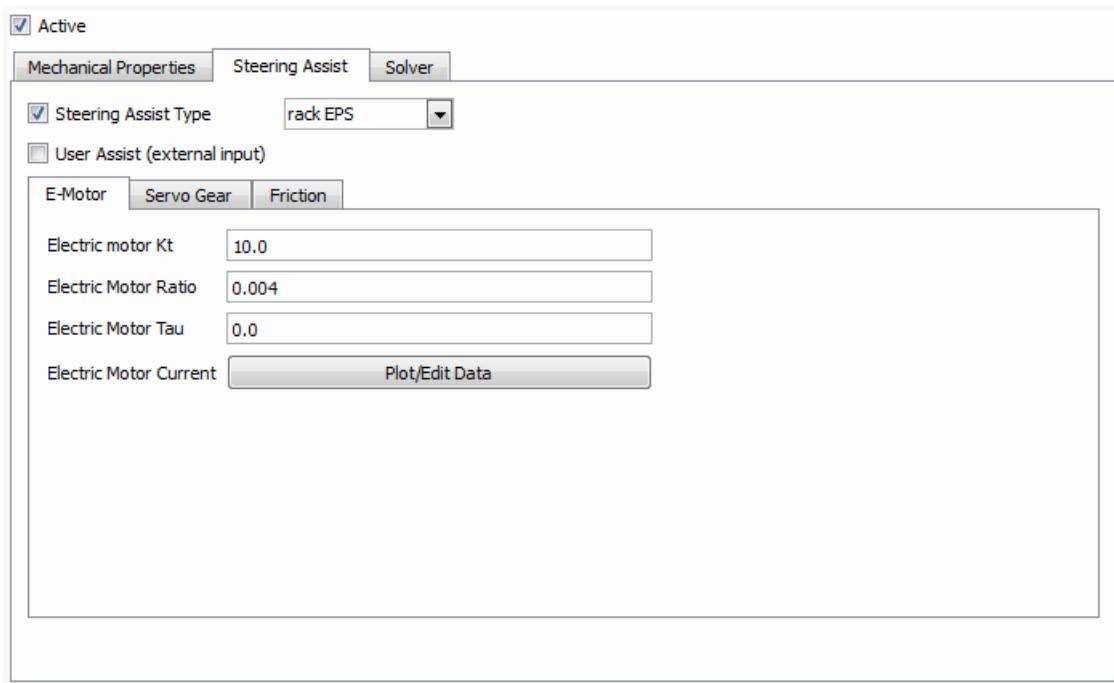
Note: the torsion bar is attached to the upper column when *column EPS* assist type is selected in Steering Assist Type option menu. For the other three options (*rack EPS*, *pinion EPS*, *electro-hydraulic*) the torsion bar is attached to the lower column.

If **Steering Assist Type** toggle button is checked the following types are available:

- **column EPS**
the electric steering assist torque acts on the steering column part.
- **pinion EPS**
the electric steering assist torque acts on the pinion part.
- **rack EPS**
the electric steering assist force acts on the rack part.

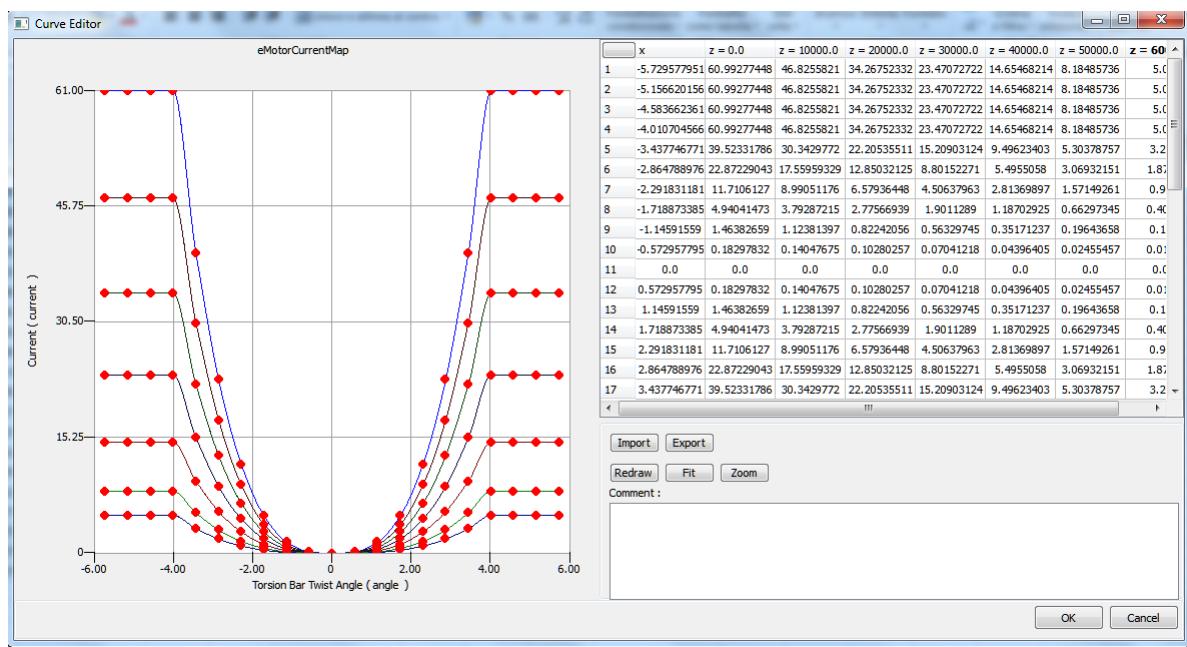
- **User Assist (external input)**

Activating the check box the assist action included in the model will be deactivated and overridden by torque/force value provided through additional [external inputs](#) (i.e. MATLAB/Simulink, FMI module, etc.).



For electric steering assist type the following parameters are available:

- **Electric Motor Kt**
motor torque constant [K_t].
- **Electric Motor Ratio**
reduction ratio of the electric motor [τ_m].
- **Electric Motor Tau**
time constant which characterizes the time response of the electric motor [τ].
- **Electric Motor Current**
3D spline which defines the electric motor current as a function of torsion bar twist angle and vehicle speed [I].
 $eCurrent = f(torsionBar_twistAngle, vehicleSpeed)$



The electric steering assist torque (or [force](#) when it acts on rack part) which acts on the rack part is defined as follows:

$$T_{ass}(t) = L^{-1}[T_{ass}(s)]$$

where $T_{ass}(s)$ is

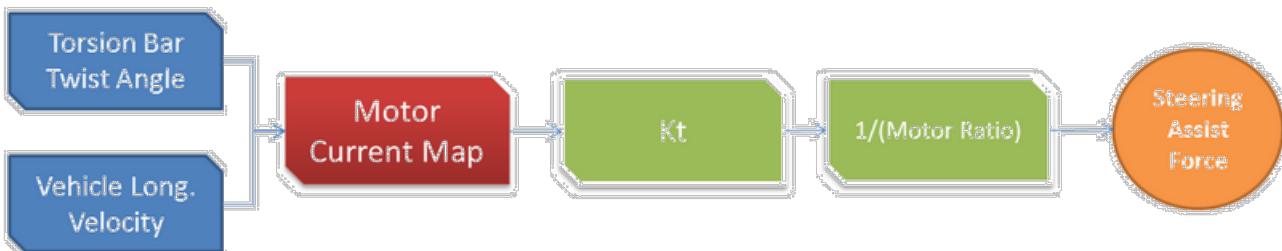
$$T_{ass}(s) = \frac{1}{1 + \tau s} T_u(s)$$

$T_u(s)$ is the Laplace transform of $T_u(t)$

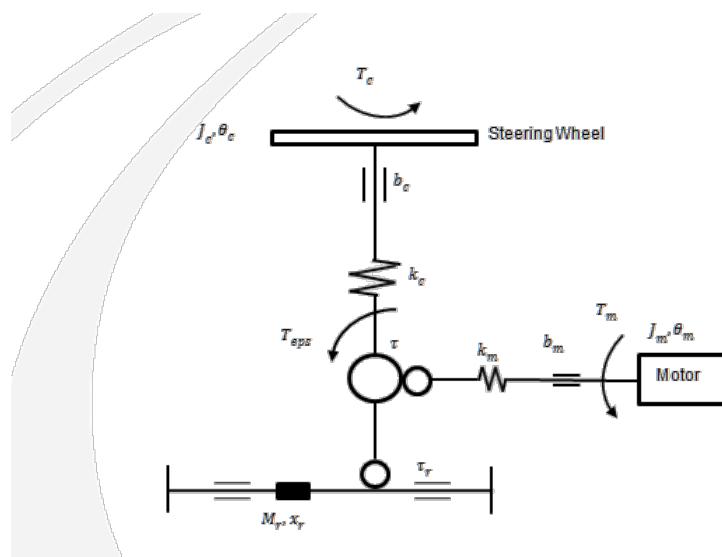
$$T_u(s) = L[T_u(t)]$$

which is defined as

$$T_u(t) = \frac{K_t I(t)}{\tau_m}$$



Electric type steering assist support the possibility of including a model of the Servo gear, according to the functional diagram below:



The equations governing the servo gear element are the following:

$$\begin{cases} T_m - \frac{T_{eps}}{\tau * \eta} - T_{friction} = J_m * \ddot{\theta}_m \\ T_{eps} = K_m * \left(\theta_c - \frac{\theta_m}{\tau} \right) + B_m * \left(\dot{\theta}_c - \frac{\dot{\theta}_m}{\tau} \right) \end{cases}$$

where:

T_m = Torque of electric motor;

T_{eps} = Effective Torque transmitted to steering;

$T_{friction}$ = Friction Torque;

θ_m = motor rotation;

θ_c = upper column rotation;

J_m = motor inertia;

J_c = steering column inertia;

τ = transmission ratio;

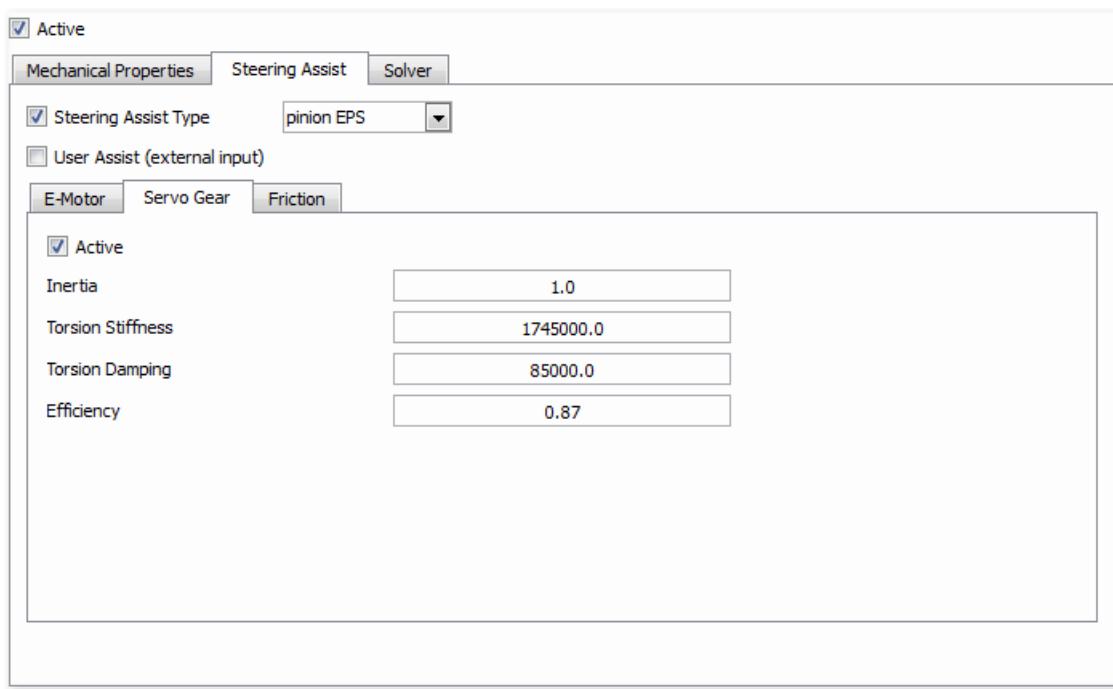
η = servo gear efficiency;

k_m, b_m = stiffness and damping of connection;

k_c, b_c = stiffness and damping of steering column;

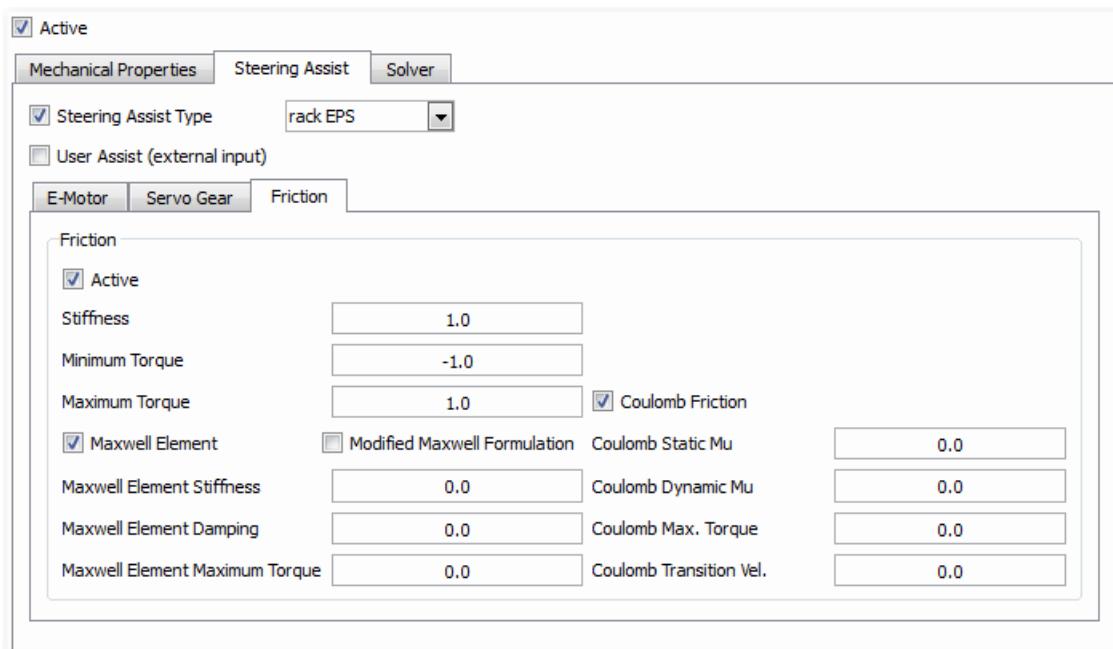
τ_r = rack to pinion transmission ratio.

The servo gear parameters can be input in the dialog below:



- Inertia**
servo gear inertia.
- Torsion stiffness**
torsion stiffness of servo gear to steering constraint.
- Torsion damping**
torsion damping of servo gear to steering constraint.
- Efficiency**
servo gear transmission efficiency.

The electric type of steering assist systems include the possibility of modeling friction according to the following parameters.



Electric power steering friction supports the following parameters:

- **Stiffness**

Friction stiffness coefficient k_{SF} of the [ESF](#) model.

- **Maximum Torque**

maximum torque that the [ESF](#) friction of the element can generate

- **Minimum Torque**

minimum torque that the [ESF](#) friction of the element can generate.

- **Maxwell Element Activity Check**

Check box for the activation of the Maxwell element feature in the power steering friction.

- **Maxwell Element Modified Formulation Check**

Check box for the activation of the modified formulation in the Maxwell element of the power steering friction (including no stiffness dependency and velocity dependency based on hyperbolic tan function).

- **Maxwell Element Stiffness**

torsion spring stiffness k_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Damping**

Torsion damping coefficient c_M of the [ESF+Maxwell](#) model.

- **Maxwell Element Maximum Torque**

Maximum value of the torque $T_{M,lim}$ of the [ESF+Maxwell](#) model.

- **Coulomb Friction Activity Check**

Check box for the activation of the pseudo-Coulomb friction component.

- **Coulomb Friction Static Mu**

Pseudo-Coulomb friction static coefficient.

- **Coulomb Friction Dynamic Mu**

Pseudo-Coulomb friction dynamic coefficient.

- **Coulomb Friction Transition Velocity**

Transition rotational velocity for static to dynamic switch.

- **Coulomb Friction Max Torque**

Maximum Torque for pseudo Coulomb friction component.

- **Electro-hydraulic**

the hydraulic steering assist force acts on the rack part.

<input checked="" type="checkbox"/> Active	Mechanical Properties	Steering Assist	Solver
<input checked="" type="checkbox"/> Steering Assist Type	electro-hydraulic		
Hydraulic Piston Area	502.4		
Hydraulic Bulk Coefficient	550.0		
Hydraulic Piston Volume	82000.0		
Hydraulic Orifice Area	50.24		
Hydraulic Orifice Coefficient	0.6		
Hydraulic Pressure	<input type="button" value="Plot/Edit Data"/>		

For electro-hydraulic steering assist type the following parameters are available:

- **Hydraulic Piston Area**

area of the hydraulic piston [A_p].

- **Hydraulic Bulk Coefficient**

bulk coefficient [c_{bulk}].

- **Hydraulic Piston Volume**

volume of the piston [V_p].

- **Hydraulic Orifice Area**

area of the orifice [A_{or}].

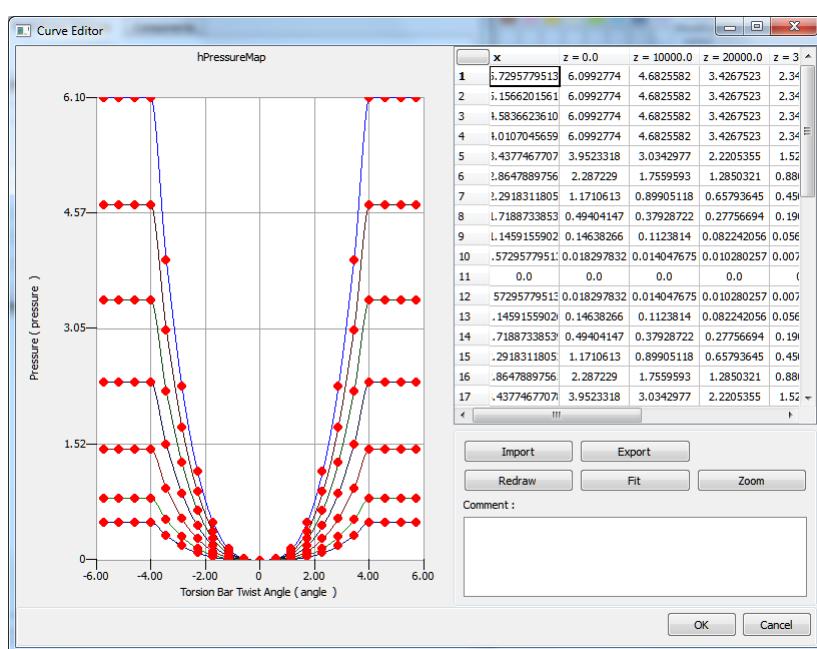
- **Hydraulic Orifice Coefficient**

orifice coefficient [c_{or}].

- **Hydraulic Pressure**

3D spline which defines the delta hydraulic pressure as a function of torsion bar twist angle and vehicle speed [p_{hyd}].

$ePressure = f(torsionBar_twistAngle, vehicleSpeed)$



The hydraulic steering assist force which acts on the rack part is defined as follows:

$$F_{ass} = A_p \cdot p_{hyd}$$

where p_{hyd} is:

$$p_{hyd} = \int_{t_{i-1}}^{t_i} \frac{dp_{hyd}}{dt} dt$$

$$\frac{dp_{hyd}}{dt} = \frac{c_{bulk}}{\frac{V_p}{2} + A_p \cdot d_{rack}} \cdot \left[A_{or} \cdot c_{or} \cdot \sqrt{\frac{2}{\rho_{fluid}} \cdot |\Delta p - p_{hyd}|} - v_{rack} \cdot A_p \right]$$

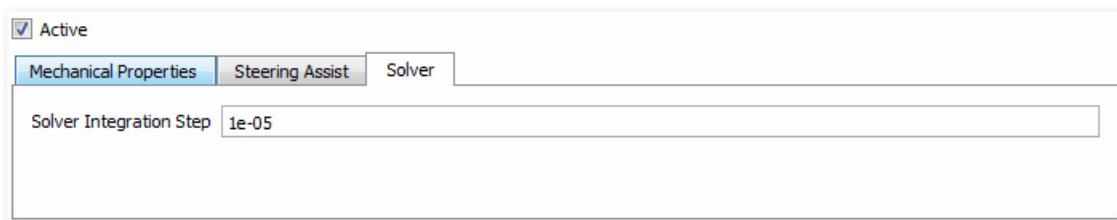
d_{rack} : rack displacement.

ρ_{fluid} : density of the fluid in the hydraulic system.

v_{rack} : rack velocity.

Δp : static pressure between the two rooms of the hydraulic system. It is a function of torsion bar twist angle and vehicle velocity.

The advanced steering model is featuring a specific solver for the motion equations.
The following dialog panel is used to set up the solver parameters.

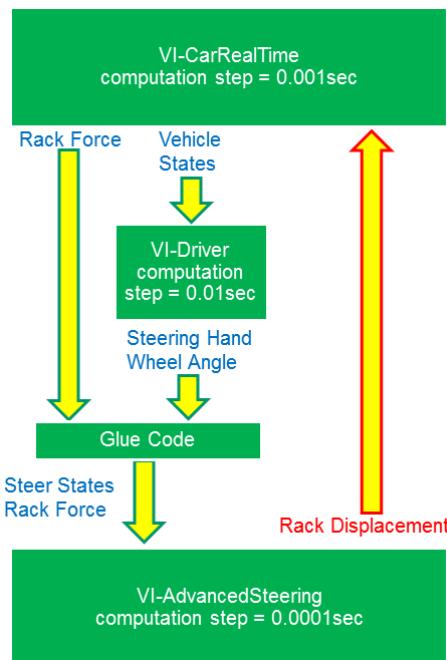


The following parameters can be set for the Solver:

- **Solver Integration Step**

time step at which steering equations of motion are solved.

The diagram below shows how the rack-pinion steering module interfaces with VI-CarRealTime vehicle model and virtual driver.



Using the advanced steering system some additional output channels will be added to VI-CarRealTime standard results file. Please refer to the [specific section](#) for their description.

Components

The Components panel stores the curves which takes into account of the effect of the steering wheel angle on the following elastic elements:

- `front_<left,right>_main_spring`
- `front_<left,right>_main_damper`
- `front_<left,right>_main_bumpstop`
- `front_<left,right>_main_reboundstop`
- `front_<left,right>_anti_roll_force`

For the first 4 elements (spring, damper, bumpstop, reboundstop) the following panels are available:

- [Kinematic Data](#)
- [Kingpin Moment Contribute](#)

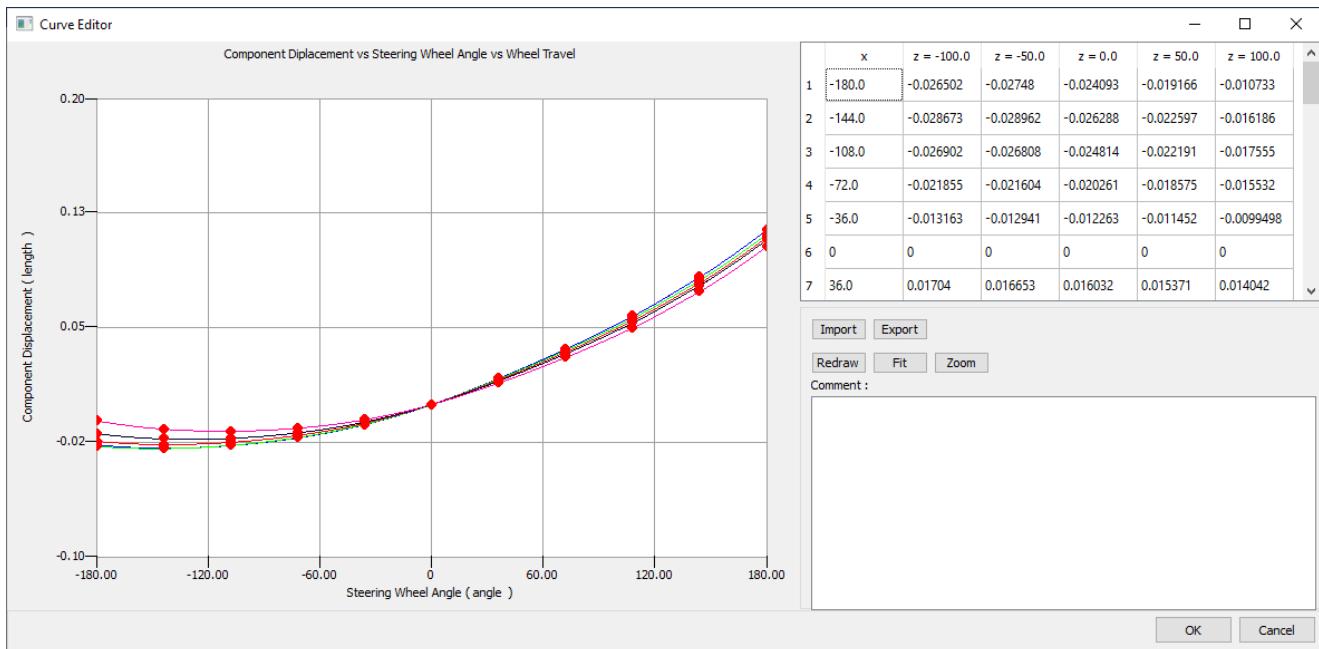
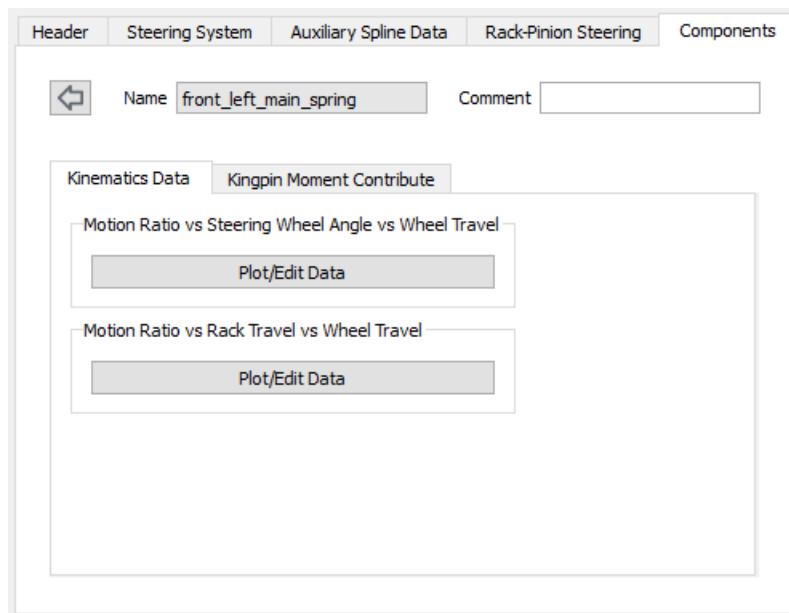
For the anti roll force element the following panels are available:

- [Force Data](#)
- [Kingpin Moment Contribute](#)

Kinematic Data panel defines the following two splines:

- **Motion Ratio vs Steering Wheel Angle vs Wheel Travel**
differential motion ratio, with respect to the motion ratio of the related elastic element, mapped as a function of the steering wheel angle and wheel travel. The value of such 3D spline is algebraically summed to the motion ratio curve of the related elastic element. Such spline is used when [Use Steering Wheel Input](#) check box is checked.
- **Motion Ratio vs Rack Travel vs Wheel Travel**
differential motion ratio, with respect to the motion ratio of the related elastic element, mapped as a function of the rack travel and wheel travel. The value of such 3D spline is algebraically summed to the motion ratio curve of the related elastic element. Such spline is used when [Use Steering Wheel Input](#) check box is unchecked.

VI-CarRealTime



The Kingpin Moment Contribute panel allows the user to activate the component participation to the total kingpin moment.

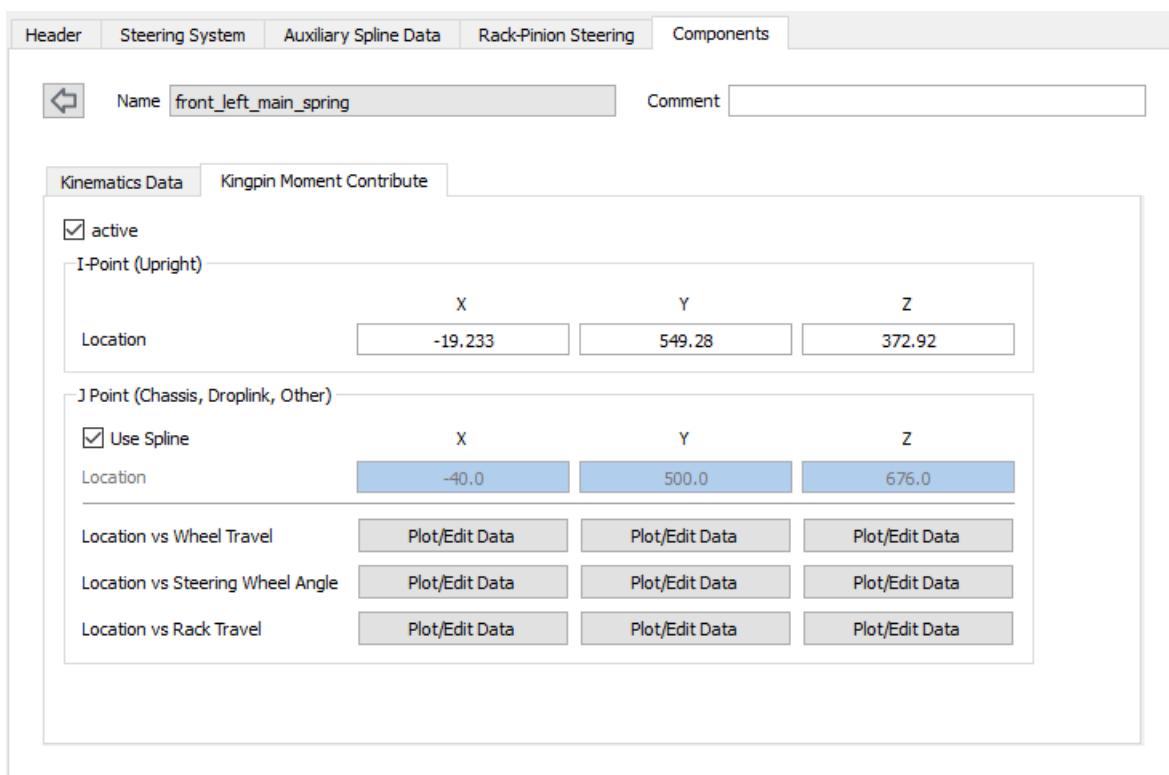
Since in VI-CarRealTime the suspension is conceptual, the solver can't automatically detect if one or more elements are connected directly to the upright. If this is the case, the component force will contribute to generate kingpin moment about the kingpin axis.

Note: when [Steering Feedback Map](#) toggle is on, the steering wheel torque (or the rack force) is not computed by VI-CarRealTime solver, since the values are automatically extracted by interpolating the spline. In such case, the contribute due to a component is already included in the maps; thus activating the kingpin moment component computation using this panel can alter the total steering wheel torque (or rack force).

When the active flag is checked, VI-CarRealTime solver will compute the kingpin moment generated by the element itself.

The following parameters are available:

- **active**
activation flag to enable the kingpin moment component computation.
- **I Point (Upright)**
defines the design time location, in [Vehicle Reference System](#), of the component attachment point to the upright. The runtime location of the point will be computed by the solver depending on the XFORMs of the upright suspension part.
- **J Point (Chassis, Droplink, Other)**
defines the design time location, in [Vehicle Reference System](#), of the component attachment point to the chassis or other.
- **Use Spline**
when the flag is active, the J Point location will be defined through the splines defined below, instead of design time location.
- **Location vs Wheel Travel**
3 sets of 3D splines which map the J Point location (X,Y,Z), in [Vehicle Reference System](#), with respect to wheel travel and opposite wheel travel.
- **Location vs Steering Wheel Angle**
3 sets of 3D splines which map the J Point location (X,Y,Z), in [Vehicle Reference System](#), with respect to steering wheel angle and wheel travel.
- **Location vs Rack Travel**
3 sets of 3D splines which map the J Point location (X,Y,Z), in [Vehicle Reference System](#), with respect to rack travel and wheel travel.



Note: if the model has been exported from Adams Car using VI-CarRealTime plug-in, the Kingpin Moment Contribute panel is automatically filled up for all the components. For all the details, please refer to [Kingpin Moment Contribute paragraph](#).

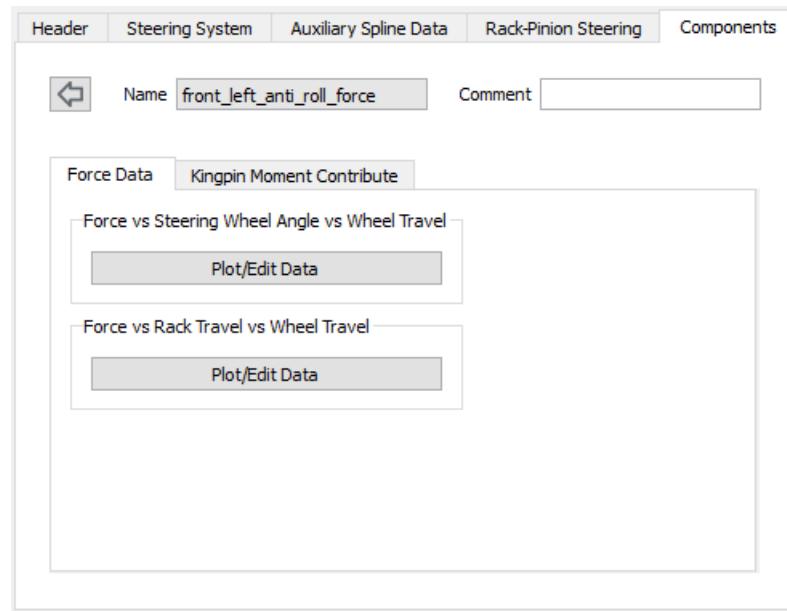
Force Data panel defines the following two splines:

- **Force vs Steering Wheel Angle vs Wheel Travel**

differential force, with respect to the [Auxiliary Anti-Roll Force](#), mapped as a function of the steering wheel angle and wheel travel. The value of such 3D spline is algebraically summed to the [Auxiliary Anti-Roll Force](#) curves. Such spline is used when [Use Steering Wheel Input](#) check box is checked.

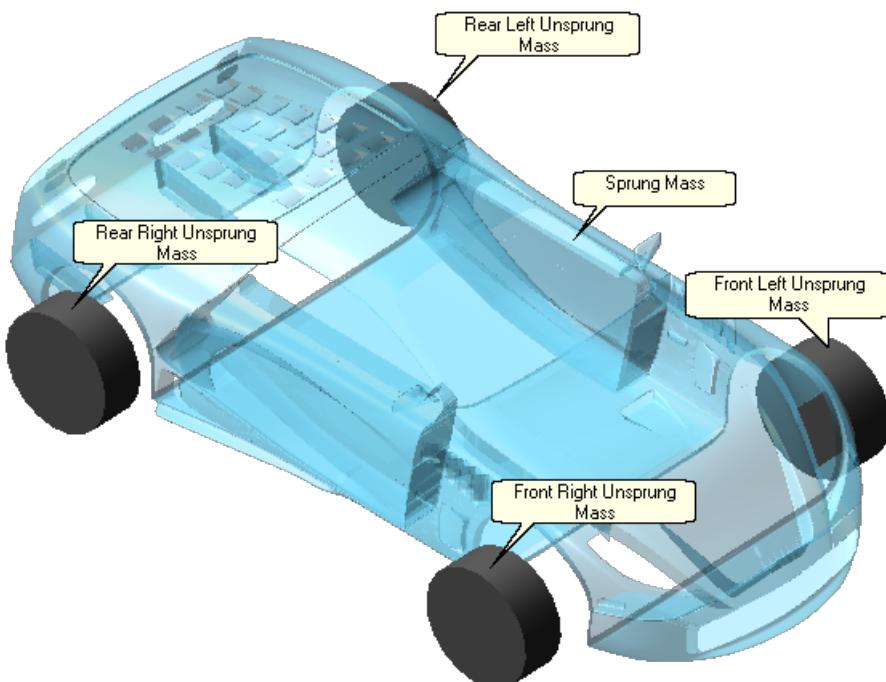
- **Force vs Rack Travel vs Wheel Travel**

differential force, with respect to the [Auxiliary Anti-Roll Force](#), mapped as a function of the rack travel and wheel travel. The value of such 3D spline is algebraically summed to the [Auxiliary Anti-Roll Force](#) curves. Such spline is used when [Use Steering Wheel Input](#) check box is checked.



Body Subsystem

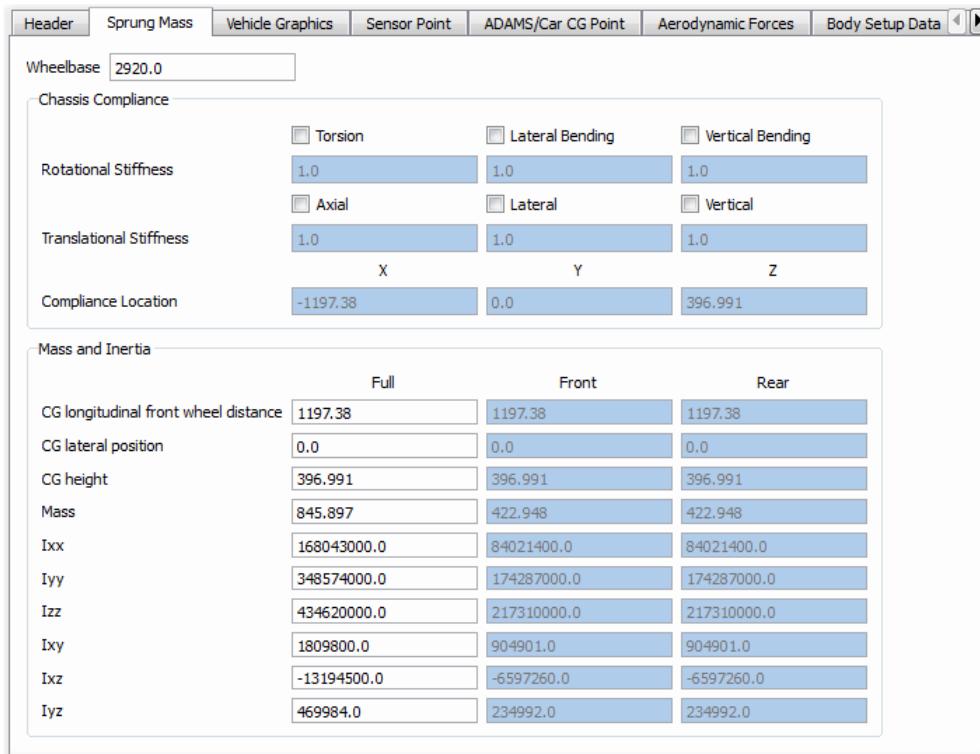
The body subsystem properties collect mass, inertia, setup and accessory information about the sprung mass part of VI-CarRealTime vehicle model.



[Sprung Mass](#)[Vehicle Graphics](#)[Sensor Point](#)[Adams Car CG Point](#)[Aerodynamic Forces](#)[Body Setup Data](#)[Cross Weight Adjustment](#)[Ride Height Maps](#)[User Sensor](#)[Rigid Part](#)[Bushings](#)[Skidplate Point](#)

Sprung Mass

The sprung mass Property Editor give access to the sprung mass inertia property of vehicle chassis. It also contains the reference to the Adams command file for vehicle model animation in Adams/PPT, the vehicle wheelbase and chassis compliance data.



Wheelbase

Distance between front and rear wheel center in [Global Reference](#) X direction.

Chassis compliance

In vehicle model it is possible to activate chassis compliance; the action adds selectively up to six DOF to the model splitting the sprung mass in two parts; mass and inertias for the two halves will need to be specified.

Activity flag

Activates chassis torsional compliance.

Rotational Stiffness

Sets the chassis rotational stiffness values (for torsion, lateral and vertical bending direction).

Translational Stiffness

Sets the chassis translational stiffness values (for axial, lateral, and vertical direction).

Compliance Location

Defines the location of the chassis split. It is expressed in [VI-CarRealTime Global Reference](#).

Mass and Inertia

The container is used to set mass and inertia mass of vehicle sprung mass.

Note: since the vehicle model is conceptual the sprung mass part is not the same thing of the body chassis part, since a portion of the suspension masses and inertia is seen as belonging to the sprung part.

CG longitudinal front wheel distance

Longitudinal distance from front wheel center and sprung mass CG (or front/rear halves).

CG lateral position

Lateral position of sprung mass CG (or front/rear halves). It is expressed in [Global Reference](#) system.

CG height

Vertical position of sprung mass CG (or front/rear halves). It is expressed in [Global Reference](#) system.

Mass

Mass of sprung part (or front/rear halves)

I_{xx}

I_{xx} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

I_{yy}

I_{yy} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

I_{zz}

I_{zz} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

I_{xy}

I_{xy} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

I_{xz}

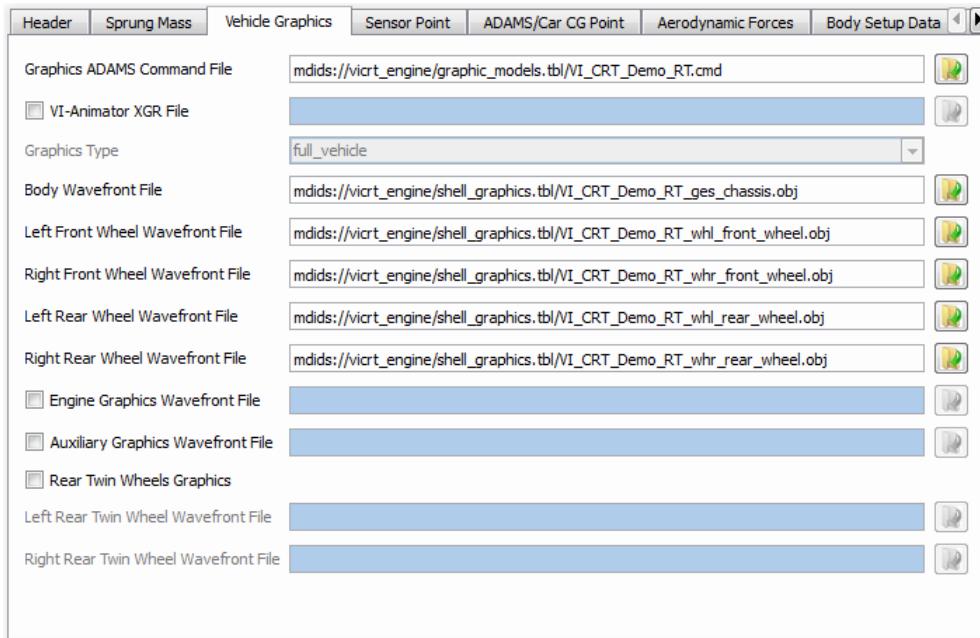
I_{xz} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

I_{yz}

I_{yz} moment of inertia of sprung part (or front/rear halves). It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system

Vehicle Graphics

The Vehicle Graphics Property Editor allows the user to refer to an external graphic file (or files) in order to define the graphics of the vehicle. Both an Adams command file (cmd) and a list of wavefront files are supported for the graphic.



The fields of this panel are:

- **Graphics ADAMS Command File**

Path of Adams command file containing the graphic model for Adams/PPT animations.

When **VI-Animator XGR File** toggle button is checked:

- **VI-Animator XGR File**

This field allows the user to choose the Graphics File (XGR) embedding all the graphics of the parts.

When **VI-Animator XGR File** toggle button is unchecked:

- **Graphics Type**

Type of graphics allowed (*full_vehicle* , *single_axle_trailer* , *dual_axle_trailer*).

Note: the fields below are enabled when a *full_vehicle* assembly is loaded.

- **Body Wavefront File**

This field allows the user to set the waveform graphics file of the body of the vehicle.

- **Left Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front left wheel of the vehicle.

- **Right Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front right wheel of the vehicle.

- **Left Rear Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the rear left wheel of the vehicle.

- **Right Rear Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the rear right wheel of the vehicle.

- **Engine Graphics Wavefront File**

When the related toggle button is checked, this field allows the user to set the waveform graphics file of the engine of the vehicle.

- **Auxiliary Graphics Wavefront File**

When the related toggle button is checked, this field allows the user to set the wavefront graphics file for auxiliary graphic parts of the vehicle.

When **Rear Twin Wheels Graphics** toggle button is checked the following fields are enabled:

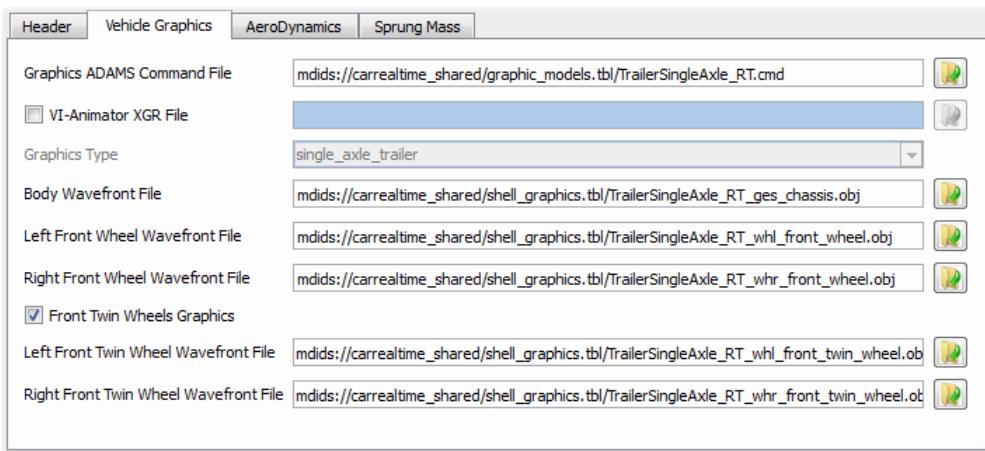
- **Left Rear Twin Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the rear left twin wheel of the vehicle.

- **Right Rear Twin Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the rear right twin wheel of the vehicle.

For a *single axle trailer* the following wavefront files are available:



- **Body Wavefront File**

This field allows the user to set the wavefront graphics file of the trailer.

- **Left Front Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the front left wheel of the trailer.

- **Right Front Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the front right wheel of the trailer.

When **Front Twin Wheels Graphics** toggle button is checked the following fields are enabled:

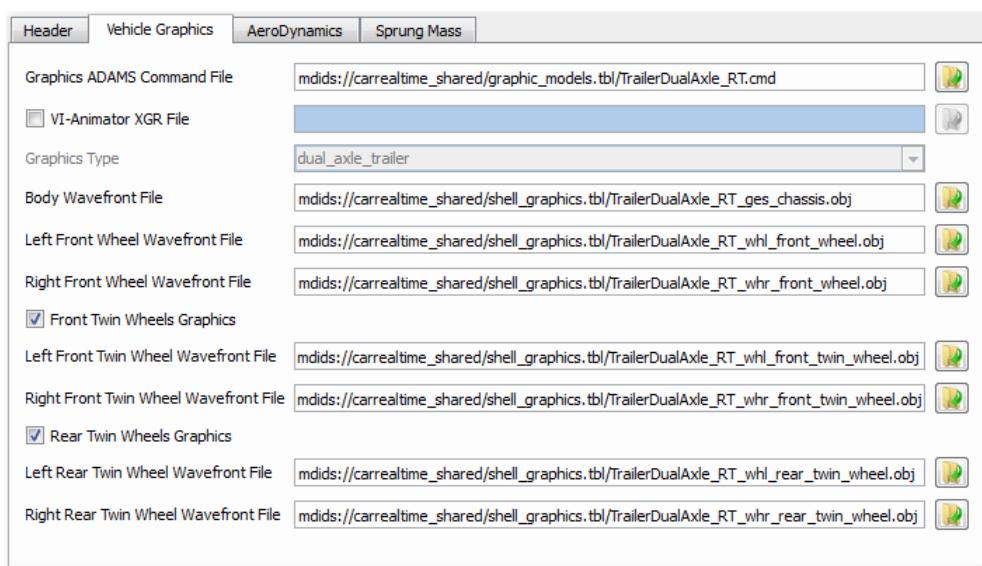
- **Left Front Twin Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the front left twin wheel of the trailer.

- **Right Front Twin Wheel Wavefront File**

This field allows the user to set the wavefront graphics file of the front right twin wheel of the trailer.

For a *dual axle trailer* the following wavefront files are available:



- Body Wavefront File**

This field allows the user to set the waveform graphics file of the trailer.

- Left Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front left wheel of the trailer.

- Right Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front right wheel of the trailer.

When **Front Twin Wheels Graphics** toggle button is checked the following fields are enabled:

- Left Front Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front left twin wheel of the trailer.

- Right Front Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front right twin wheel of the trailer.

When **Rear Twin Wheels Graphics** toggle button is checked the following fields are enabled:

- Left Rear Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the rear left twin wheel of the trailer.

- Right Rear Twin Wheel Wavefront File**

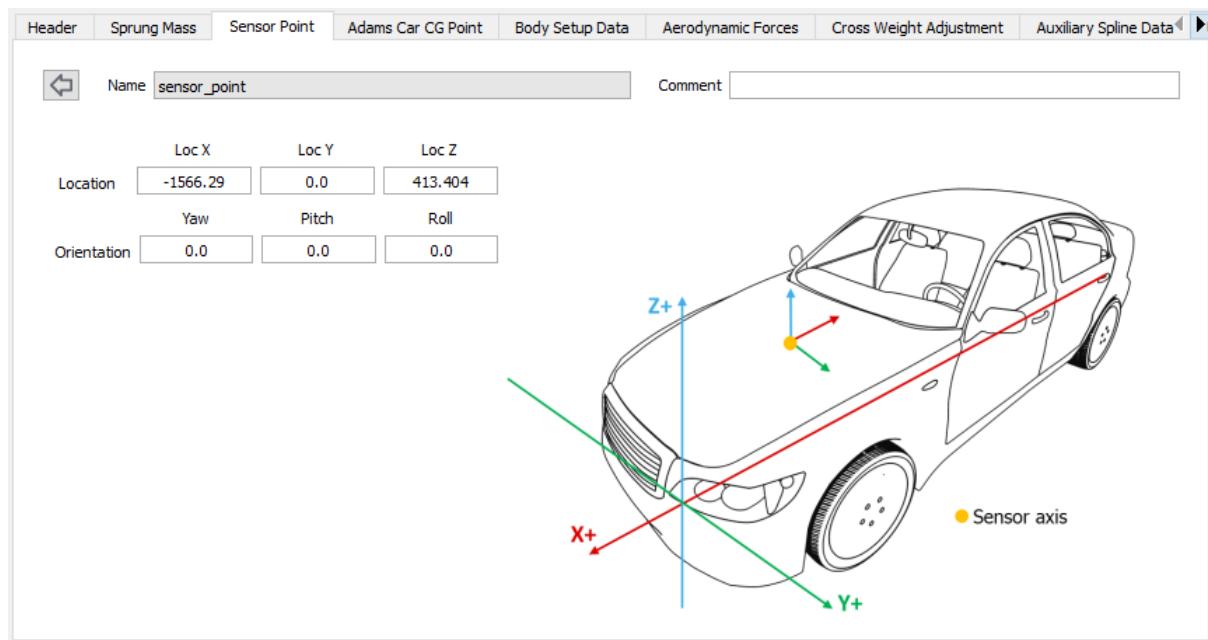
This field allows the user to set the waveform graphics file of the rear right twin wheel of the trailer.

Sensor Point

Vehicle models can have one more sensor point to monitor position, velocities and accelerations during the simulations.

The Sensor Point Property Editor is used to set the position and orientation of sensors. Quantities are expressed in [Global Reference](#) frame.

VI-CarRealTime

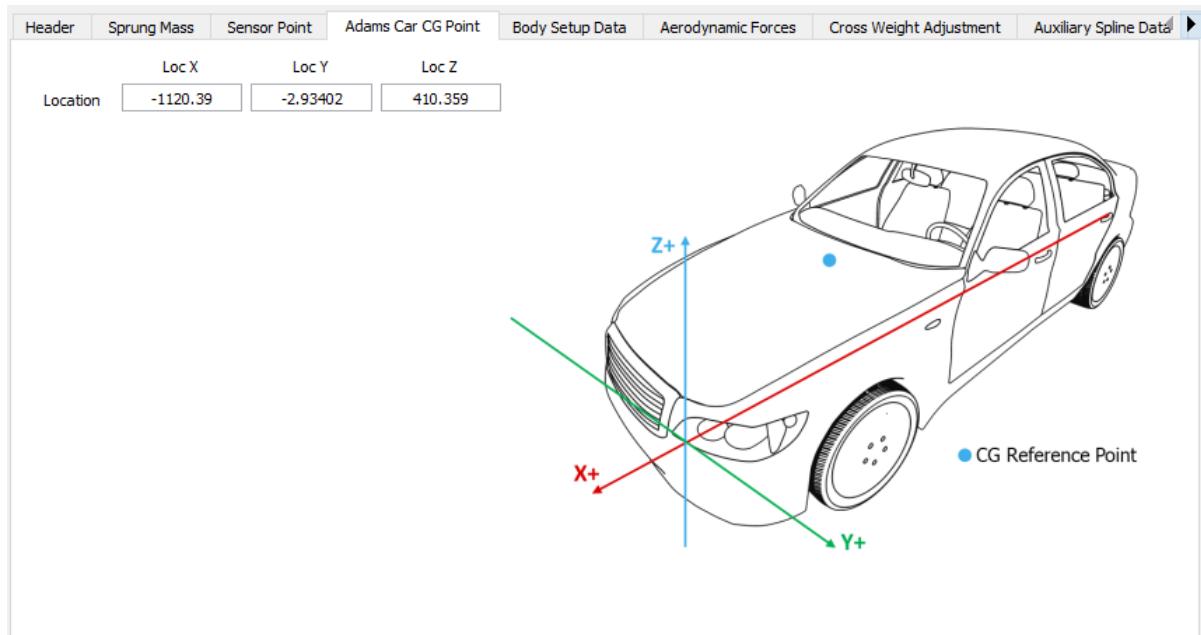


Further information on adding extra sensor point can be found [here](#).

Adams Car CG Point

The Adams Car CG Point Property Editor is used to set the position of chassis part CG. It is used to compute chassis_displacement, chassis_velocity and chassis_acceleration request channels during simulation. Depending on the portion of suspension parts belonging to sprung mass it may be not coincident with sprung part CG.

When extracting vehicle from Adams Car the value is set to the position of center of mass of part identified by the cos_body communicator.



Aerodynamic Forces

VI-CarRealTime include the following aerodynamic forces that can be optionally deactivated:

- **Front Downforce**
- **Rear Downforce**
- **Drag Force**
- **Front Side Force**
- **Rear Side Force**
- **Roll Torque**
- **Pitch Torque**

The forces are applied at points specified by user (force location fields) and are depending on:

- aero map
- ride height

Aero maps are specified using an external [property file](#) which expresses the dependency of the single force component on ride height (measured at a specific sensor point for each component) and vehicle velocity. Aerodynamic property file can also be used to set the link to a custom library for aerodynamic force computation. Please refer to [Aero Forces](#) documentation for more information on aerodynamic model used in VI-CarRealTime and [customization](#).

Point of application of drag and down forces can be set by the user:

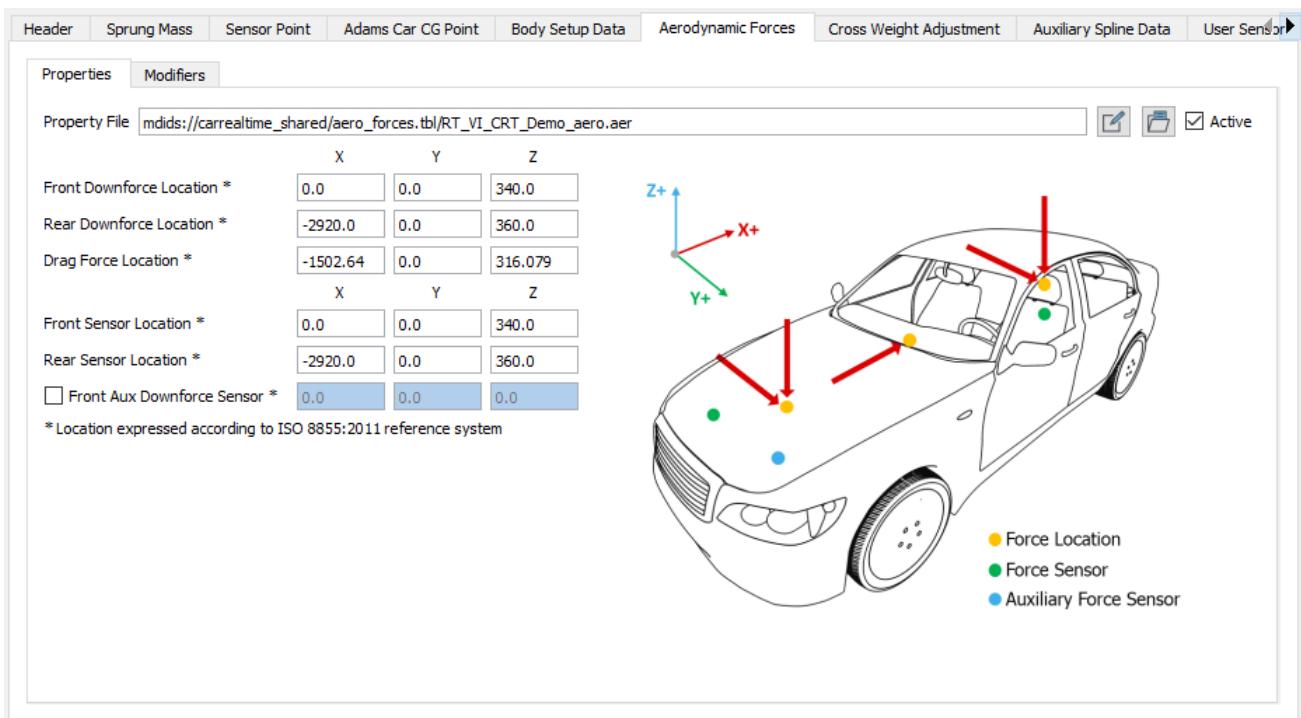
- **Front Downforce Location**
X,Y,Z coordinates* of the front down force application point.
- **Rear Downforce Location**
X,Y,Z coordinates* of the rear down force application point.
- **Drag Force Location**
X,Y,Z coordinates* of the drag force application point.
- **Front Sensor Location**
X,Y,Z coordinates* of the front sensor with respect to whom the aerodynamic maps are defined.
- **Rear Sensor Location**
X,Y,Z coordinates* of the rear sensor with respect to whom the aerodynamic maps are defined.
- **Front Aux Downforce Sensor**
X,Y,Z coordinates* of an optional auxiliary sensor which is used, together with the Front Sensor Location, to compute the Front Ride Height values.

(*) in [Vehicle Reference System](#) coordinates.

Ride height sensor positions for aerodynamic force can also be set by the user through the **Front** and **Rear Sensor Location**. These sensors are the reference points used to enter the aerodynamic maps (defined in the Aerodynamic property file) and carry out the values of the front and rear downforce respectively.

An auxiliary front sensor (**Front Aux Downforce Sensor**) can be used; this sensor is expected to be on the right half side of the car; a check is performed on its location at run time in order to assign the front ride height and auxiliary front ride height output channels to the proper sensor.

When the Front Aux Downforce Sensor is active the front ride height values used in aerodynamic maps will be the average of the two channels.



Aero forces can also be scaled, shifted and include a smoothing time to their application by using modifiers tab in the property editor:

- **smoothing time**

represents the amount of time used to raise the force from 0 to the target value. Such parameter is used to avoid a too sharp transient behavior when you pass from static analysis to dynamic analysis.

- **shift**

value used as offset for the related force component.

- **scale**

multiplying factor of the force component.

Zero values in scale fields are used to deactivate the specific components. The activity of the modifiers depends on the aerodynamic model adopted in property file. The full set of modifiers is available for [user aerodynamic model](#).

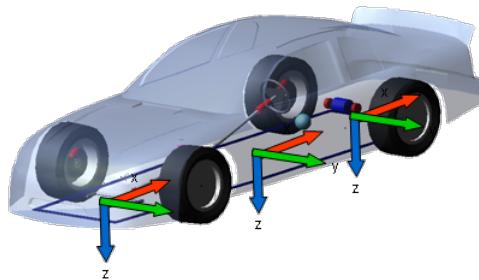
Properties			
	Smoothing Time	Shift	Scale
Front Downforce	0.0	0.0	1.0
Rear Downforce	0.0	0.0	1.0
Center Downforce	0.0	0.0	1.0
Front Dragforce	0.0	0.0	0.0
Center Dragforce	0.5	0.0	1.0
Rear Dragforce	0.0	0.0	0.0
Front Sideforce	0.0	0.0	1.0
Rear Sideforce	0.0	0.0	1.0
Center Sideforce	0.0	0.0	1.0
Roll Moment	0.5	0.0	1.0
Pitch Moment	0.5	0.0	1.0
Yaw Moment	0.0	0.0	1.0

VI-CarRealTime implements two different aerodynamic equations:

- [Standard Aerodynamic](#)
- [Advanced Aerodynamic](#)

The aerodynamic parameters and maps are characterized into an [AER](#) property file.

The output forces produced by the aerodynamic routines are consistent with the reference system shown in the following picture:



ρ	air density
A	vehicle effective area
V	vehicle longitudinal speed
Frh	front ride height
Rrh	rear ride height
Dh	drag (arm) height, it's the distance between the drag force actuator point and the road plane
Sfh	side force front (arm) height, it's the distance between the front side force actuator point and the road plane
Srh	side force rear (arm) height, it's the distance between the rear side force actuator point and the road plane

VI-CarRealTime

SSA	vehicle side slip angle
PA	vehicle pitch angle
RA	vehicle roll angle
STA	vehicle steering angle

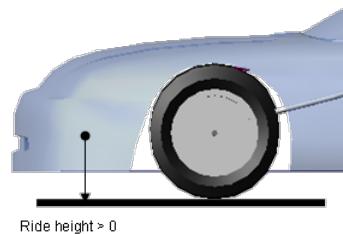
The Standard Aerodynamic consists in 3DOF aerodynamics forces:

- [front downforce](#)
- [rear downforce](#)
- [drag force](#)

The formulation implemented includes multiple maps of coefficients to generate aerodynamic forces that depends on:

- **front ride height**
runtime location of Front Sensor Location in Properties panel of Aerodynamic Forces section.
- **rear ride height**
runtime location of Rear Sensor Location in Properties panel of Aerodynamic Forces section.

The input signals passed to the aerodynamic routines are consistent with the VI-CarRealTime standard signals. Here the reference system used for ride height input signal:



Ride Height

Calculated as:

$$F_z = k \cdot \frac{\rho}{k_p} \cdot (F_z^{\text{table}} + k_{\text{shift}}^f) \cdot \left(\frac{V}{V_{ref}} \right)^2$$

where:

- k = front downforce Scale factor taken from Modifiers block of Aerodynamic Force panel
 k_p = reference density scaling factor, `REFERENCE_DENSITY` parameter in aer property file
 F_z^{table} = force value interpolated from `[FRONT_DOWNFORCE]` block of aer property file
 k_{shift}^f = front downforce Shift factor taken from Modifiers block of Aerodynamic Force panel
 V = actual vehicle longitudinal speed
 V_{ref} = reference speed at which the maps have been calculated; `REFERENCE_VELOCITY` parameter in aer property file

Calculated as:

$$F_z = k \cdot \frac{\rho}{k_\rho} \cdot (F_z^{table} + k_{shift}^r) \cdot \left(\frac{V}{V_{ref}} \right)^2$$

where:

- k = rear downforce Scale factor taken from Modifiers block of Aerodynamic Force panel
- k_ρ = reference density scaling factor, `REFERENCE_DENSITY` parameter in aer property file
- F_z^{table} = rear value interpolated from `[REAR_DOWNFORCE]` block of aer property file
- k_{shift}^r = rear downforce Shift factor taken from Modifiers block of Aerodynamic Force panel
- V = actual vehicle longitudinal speed
- V_{ref} = reference speed at which the maps have been calculated; `REFERENCE_VELOCITY` parameter in aer property file

Calculated as:

$$F_x = k \cdot \frac{\rho}{k_\rho} \cdot (F_d^{table} + k_{shift}^d) \cdot \left(\frac{V}{V_{ref}} \right)^2$$

where:

- k = center drag force Scale factor taken from Modifiers block of Aerodynamic Force panel
- k_ρ = reference density scaling factor, `REFERENCE_DENSITY` parameter in aer property file
- F_d^{table} = drag force value interpolated from `[DRAG]` block of aer property file
- k_{shift}^d = center drag force Shift factor taken from Modifiers block of Aerodynamic Force panel
- V = actual vehicle longitudinal speed
- V_{ref} = reference speed at which the maps have been calculated; `REFERENCE_VELOCITY` parameter in aer property file

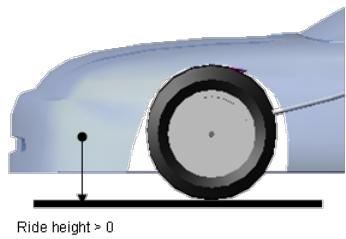
The Advanced Aerodynamic consists in 6DOF aerodynamics forces:

- [a single body longitudinal \(drag\) force](#)
- [front and rear lateral forces](#)
- [front and rear vertical forces](#)
- [a single body rolling torque](#)
- [a single body pitch torque](#) (depending on drag body force)

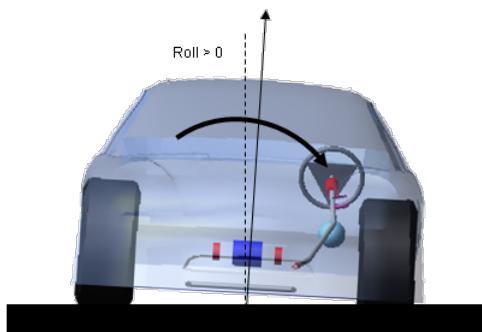
The formulation implemented includes multiple maps of coefficients and modifiers to generate aerodynamic forces that depends on:

- front ride height
- rear ride height
- side slip
- roll
- pitch
- steer

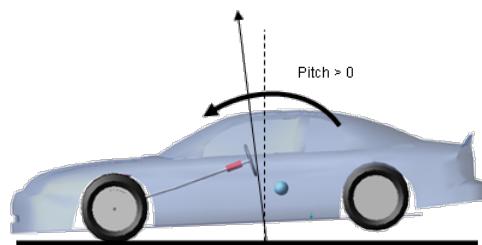
The input signals passed to the aerodynamic routines are consistent with the VI-CarRealTime standard signals. In detail the reference system for each input signal is the following:



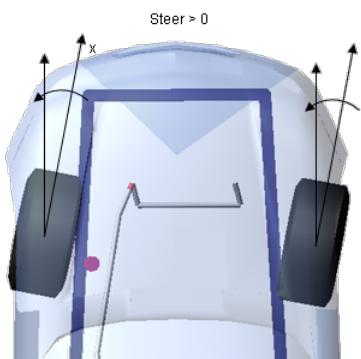
Ride Height



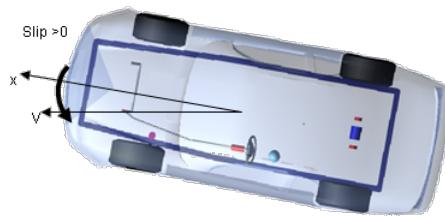
Roll (front view)



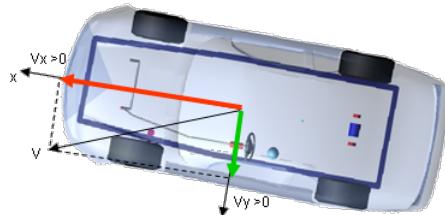
Pitch



Steer (ISO)



Slip (ISO)



Longitudinal/Lateral velocities (ISO)

Please note that the conventions defined above applies to both built in and user defined aerodynamic formulation based on the VI-grade aeroforce interface.

VI-CarRealTime solver will provide appropriate input to the computation routines and will apply resulting forces in the defined reference system. The user is responsible for defining in the correct reference system the coefficients of the [AER](#) property file maps.

Moments are consistent with right hand rotation along the defined axes.

Calculated as:

$$F_{zF} = \frac{1}{2} \cdot \rho \cdot A \cdot C_z \cdot V^2 \cdot Cz_{bal} \cdot Cz_a \cdot Cz_f$$

$$F_{zR} = \frac{1}{2} \cdot \rho \cdot A \cdot C_z \cdot V^2 \cdot (1 - Cz_{bal}) \cdot Cz_a \cdot Cz_r$$

where:

Cz	= Czb (Frh, RRh) + Czoff
Czb	= Cz body component (3d spline)
Czbal	= front-rear balance (3d spline)
Czoff	= Cz offset (+)
Cza	= additional vehicle effect = Czs (SSA) * Czp (PA) * Czr (RA) * Czst (STA)
Czs	= Cz (additional vehicle effect) side slip angle component (2D spline)
Czp	= Cz ((additional vehicle effect) pitch angle component (2D spline))
Czr	= Cz ((additional vehicle effect) roll angle component (2D spline))
Czst	= Cz ((additional vehicle effect) steering angle component (2D spline))
Czf	= front effect = Czfss (SSA) * Czfp (PA) * Czfr (RA) * Czfst (STA)
Czfss	= Cz (front effect) side slip angle component (2D spline)
Czfp	= Cz (front effect) pitch angle component (2D spline)
Czfr	= Cz (front effect) roll angle component (2D spline)
Czfst	= Cz (front effect) steering angle component (2D spline)
Czr	= rear effect = Czrss (SSA) * Czrp (PA) * Czrr (RA) * Czrst (STA)
Czrss	= Cz (rear effect) side slip angle component (2D spline)
Czrp	= Cz (rear effect) pitch angle component (2D spline)
Czrr	= Cz (rear effect) roll angle component (2D spline)
Czrst	= Cz (rear effect) steering angle component (2D spline)

Note: V value can be altered if the [WIND effect](#) is included in the aerodynamic file depending on the wind speed and direction.

(+) Cz offset formulation:

$V > OVlim \Rightarrow Czoff = OCz * |V - OVlim| / OVref$

$V \leq OVlim \Rightarrow Czoff = 0.0$

where:

OVlim = offset velocity activation limit (see **VELOCITY_THRESHOLD** under **OFFSET_PARAMETERS** data) <27.777 m/s>

OVref = offset reference velocity (see **REFERENCE_VELOCITY_OF_OFFSET** under **OFFSET_PARAMETERS** data) <13.888 m/s>

$OCz = Cz \text{ offset factor (see } CZ \text{ under OFFSET_PARAMETERS data) } <0.0>$

Please note that Cz offset formulation is rarely used (Cz offset is zero by default)

Calculated as:

$$F_x = \frac{1}{2} \cdot \rho \cdot A \cdot C_x \cdot V^2 \cdot Cx_a$$

where:

- $Cx = Cxb (\text{Frh, RRh}) + Cxf (\text{Frh, RRh}) + Cxr (\text{Frh, RRh}) + Cxoff$
- $Cxa = \text{additional vehicle effect} = Cxss (\text{SSA}) * Cxp (\text{PA}) * Cxr (\text{RA}) * Cxst (\text{STA})$
- $Cxb = Cx \text{ body component (3d spline)}$
- $Cxf = Cx \text{ front component (3d spline)}$
- $Cxr = Cx \text{ rear component (3d spline)}$
- $Cxoff = Cx \text{ offset (+)}$
- $Cxss = Cx \text{ side slip angle component (2D spline)}$
- $Cxp = Cx \text{ pitch angle component (2D spline)}$
- $Cxr = Cx \text{ roll angle component (2D spline)}$
- $Cxst = Cx \text{ steering angle component (2D spline)}$

Note: V value can be altered if the [WIND effect](#) is included in the aerodynamic file depending on the wind speed and direction.

(+) Cx offset formulation:

$V > OVlim \Rightarrow Cxoff = (OCxb + OCxf + OCxr) * |V - OVlim| / OVref$

$V \leq OVlim \Rightarrow Cxoff = 0.0$

where:

- $OVlim = \text{offset velocity activation limit (see } VELOCITY_THRESHOLD \text{ under OFFSET_PARAMETERS data) } <27.777 \text{ m/s}>$
- $OVref = \text{offset reference velocity (see } REFERENCE_VELOCITY_OF_OFFSET \text{ under OFFSET_PARAMETERS data) } <13.888 \text{ m/s}>$
- $OCxb = Cx \text{ offset body factor (see } CX_BODY \text{ under OFFSET_PARAMETERS data) } <0.0>$
- $OCxf = Cx \text{ offset front factor (see } CX_FRONT \text{ under OFFSET_PARAMETERS data) } <0.0>$
- $OCxr = Cx \text{ offset rear factor (see } CX_REAR \text{ under OFFSET_PARAMETERS data) } <0.0>$

Please note that Cx offset formulation is rarely used (Cx offset is zero by default)

Calculated as:

$$F_{yF} = \frac{1}{2} \cdot \rho \cdot A \cdot C_y \cdot V^2 \cdot Cy_{bal} \cdot Cy_a \cdot Cy_f$$

$$F_{yR} = \frac{1}{2} \cdot \rho \cdot A \cdot C_y \cdot V^2 \cdot (1 - Cy_{bal}) \cdot Cy_a \cdot Cy_r$$

where:

C_y	= Cy_b (Frh, RRh)
Cy_b	= Cy body component (3d spline)
Cy_{bal}	= front-rear balance (3d spline)
Cy_a	= additional vehicle effect = Cy_{ss} (SSA) * Cy_p (PA) * Cy_r (RA) * Cy_{st} (STA)
Cy_{ss}	= Cy (additional vehicle effect) side slip angle component (2D spline)
Cy_p	= Cy (additional vehicle effect) pitch angle component (2D spline)
Cy_r	= Cy (additional vehicle effect) roll angle component (2D spline)
Cy_{st}	= Cy (additional vehicle effect) steering angle component (2D spline)
Cy_f	= front effect = Cy_{fss} (SSA) * Cy_{fp} (PA) * Cy_{fr} (RA) * Cy_{fst} (STA)
Cy_{fss}	= Cy (front effect) side slip angle component (2D spline)
Cy_{fp}	= Cy (front effect) pitch angle component (2D spline)
Cy_{fr}	= Cy (front effect) roll angle component (2D spline)
Cy_{fst}	= Cy (front effect) steering angle component (2D spline)
Cy_r	= rear effect = Cy_{rss} (SSA) * Cy_{rp} (PA) * Cy_{rr} (RA) * Cy_{rst} (STA)
Cy_{rss}	= Cy (rear effect) side slip angle component (2D spline)
Cy_{rp}	= Cy (rear effect) pitch angle component (2D spline)
Cy_{rr}	= Cy (rear effect) roll angle component (2D spline)
Cy_{rst}	= Cy (rear effect) steering angle component (2D spline)

Note: V value can be altered if the [WIND effect](#) is included in the aerodynamic file depending on the wind speed and direction.

Calculated as:

$$M_x = \frac{1}{2} \cdot \rho \cdot A \cdot C_{mx} \cdot V^2 \cdot C_{mxa} \cdot C_{mxr} \cdot C_{mxr} + M_x F_y$$

$$M_x F_y = -(F_{yF} \cdot S_{fh} + F_{yR} \cdot S_{rh})$$

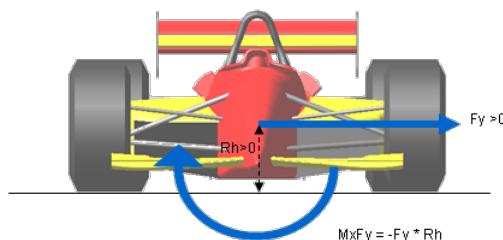
where:

C_{mx}	= C_{mx} (Frh, RRh) = C_{mx} body component (3d spline)
C_{mx} a	= additional vehicle effect = C_{mxss} (SSA) * C_{mxp} (PA) * C_{mxr} (RA) * C_{mxst} (STA)
C_{mxss}	= C_{mx} (additional vehicle effect) side slip angle component (2D spline)

Cmxp	=	Cmx (additional vehicle effect) pitch angle component (2D spline)
Cmxr	=	Cmx (additional vehicle effect) roll angle component (2D spline)
Cmxst	=	Cmx (additional vehicle effect) steering angle component (2D spline)
Cmxf	=	front effect = Cmxfs (SSA) * Cmxfp (PA) * Cmxfr (RA) * Cmxfst (STA)
Cmxfs	=	Cmx (front effect) side slip angle component (2D spline)
Cmxfp	=	Cmx (front effect) pitch angle component (2D spline)
Cmxfr	=	Cmx (front effect) roll angle component (2D spline)
Cmxfst	=	Cmx (front effect) steering angle component (2D spline)
Cmxr	=	rear effect = Cmxrss (SSA) * Cmxrp (PA) * Cmxrr (RA) * Cmxrst (STA)
Cmxrss	=	Cmx (rear effect) side slip angle component (2D spline)
Cmxrp	=	Cmx (rear effect) pitch angle component (2D spline)
Cmxrr	=	Cmx (rear effect) roll angle component (2D spline)
Cmxrst	=	Cmx (rear effect) steering angle component (2D spline)
FYf * Sfh	=	is the torque component due to the front side force arm
FYr * Srh	=	is the torque component due to the rear side force arm

Note: V value can be altered if the [WIND effect](#) is included in the aerodynamic file depending on the wind speed and direction.

Please note that switching off the body roll moment does not remove the **MxFy** contribution



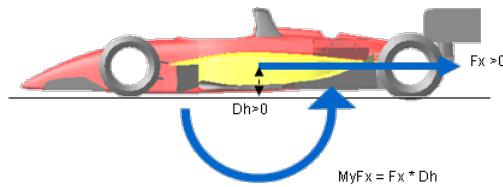
Calculated as the drag force multiplied the drag arm height:

$$M_y = F_x \cdot D_h$$

It's needed to compensate the distance between the force application point and the road plane.

The pitch torque represents a correction moment which is applied to the sprung mass.

In general the aerodynamic longitudinal forces are measured at road level, so in the aerodynamic property file the splines define the amount of longitudinal force (measured at ground level). Since the real drag force is applied at Drag Force Location point, if you apply only the Fx force you would have also a My moment due to the vertical distance (arm) between Drag Force Location point and Aerodynamic Measurement point. Such My moment must be subtracted in order to have a drag force effect which reflects the actual measures.



Body Setup Data

Rigid parts can be added to chassis, their behaviour can vary according the vehicle configuration (**passenger** or **racing**):

Header	Sprung Mass	Vehicle Graphics	Sensor Point	ADAMS/Car CG Point	Aerodynamic Forces	Body Setup Data	Cross Weight
Body Mass Setup Configuration: <input type="button" value="racing"/> <input type="button" value="passenger"/>						<input type="button" value="racing"/>	
X	Z	Setup Mass	Running Mass	Setup	Running		

Racing Configuration:

In motorsport applications it is a common practise to have setup conditions different from running conditions.

Body setup data Property Editor can be used to define masses and locations for driver, fuel and ballasts in setup and driving conditions.

Locations are expressed in [Global Reference](#) frame.

	X	Y	Z	Setup Mass	Running Mass	Setup	Running
Driver	-914.4	0.0	369.4		74.84274105	<input type="checkbox"/>	<input checked="" type="checkbox"/> Active
Fuel	-1417.3	0.0	350.86	46.0	46.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Active
Ballast 1	-601.6	300.0	64.598		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active
Ballast 2	-601.6	-300.0	64.598		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active
Ballast 3	-1101.6	200.0	64.598		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active
Ballast 4	-1101.6	-200.0	64.598		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active
Ballast 5	-2931.6	539.94	236.12		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active
Ballast 6	-2931.6	-539.94	236.12		1.0	<input type="checkbox"/>	<input type="checkbox"/> Active

Passenger Configuration:

In common applications it is practise to test vehicle performance under different load condition.

Body setup data Property Editor can be used to define masses and locations for driver, passenger, fuel and luggage. Moreover in case of tow-vehicle destination, user can define mass and location for hitch part.

For J2807 events user can also specify max displacement offset w.r.t. design position for luggage part and hitch. Such offset will be used during setup stage in order to achieve event weight targets.

Locations are expressed in [Global Reference](#) frame.

Body Mass Setup Configuration: passenger								
	X	Y	Z	Mass	Activity	max Offset X	max Offset Y	max Offset Z
Driver	-1442.213	458.467	1054.695	68.0	<input checked="" type="checkbox"/> Active			
Passenger 1	-1442.213	-459.533	1054.695	68.0	<input checked="" type="checkbox"/> Active			
Passenger 2	-1425.213	-0.533	1087.695	0.0	<input type="checkbox"/> Active			
Passenger 3	-2467.213	464.467	1076.695	68.0	<input type="checkbox"/> Active			
Passenger 4	-2467.213	-465.533	1076.695	68.0	<input type="checkbox"/> Active			
Passenger 5	-2467.213	-0.533	1076.695	68.0	<input type="checkbox"/> Active			
Fuel	0.0	0.0	0.0	0.0	<input type="checkbox"/> Active			
Luggage	-3685.01	0.0	848.785	35.0	<input checked="" type="checkbox"/> Active	1500.0	0.0	0.0
Hitch	-4568.952	0.0	250.0	0.0	<input checked="" type="checkbox"/> Active	0.0	0.0	150.0

Cross Weight Adjustment

VI-CarRealTime features the capability of adjusting suspension and vehicle characteristics.

This practise, common in racing applications, is implemented in VI-CarRealTime by modifying the values that the adjustable quantities assume after static equilibrium of the vehicle and it is achieved using different methods depending on the vehicle and suspension architecture.

In body subsystem it is possible to activate a Property Editor for cross weight, which is defined as difference of L/R wheel loads after static equilibrium.

There are three methods available:

- **Front Force**
CW = FrontRight_TireFz - FrontLeft_TireFz.
- **Rear Force**
CW = RearRight_TireFz - RearLeft_TireFz.
- **Ratio**
CW = (FrontLeft_TireFz + RearRight_TireFz) /Sum (TireFz).
- **Cross Ratio**
CW = (FrontRight_TireFz + RearLeft_TireFz) /Sum (TireFz).

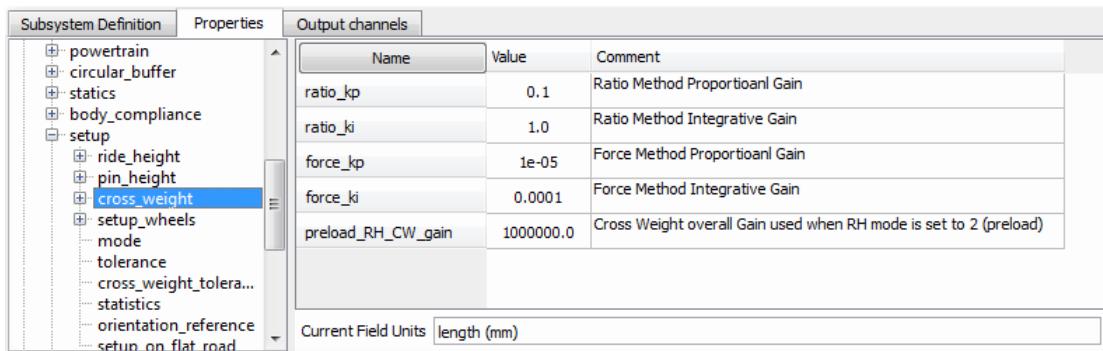
Sensor Point	ADAMS/Car CG Point	Aerodynamic Forces	Body Setup Data	Cross Weight Adjustment	Auxiliary Spline Data	User Sensor	◀	▶
Cross Weight Method <input type="radio"/> Front Force <input type="radio"/> Rear Force <input type="radio"/> Ratio <input checked="" type="radio"/> Cross Ratio <input checked="" type="checkbox"/> Active Cross Weight: CW=(FR_TirFZ+RL_TirFZ)/Sum(TirFZ) <input type="text" value="0.5"/>								

Cross Weight adjustment properties are created automatically when exporting a vehicle model form VI-Automotive.

VI-CarRealTime

In VI-CarRealTime static equilibrium is performed by means of a dynamic analysis which is continuing until the kinetic energy of the system falls below a given threshold. The Cross Weight adjustment is achieved in VI-CarRealTime by modifying the pushrod length or spring preload during "static analysis" phase via PI controller.

The gains of the controller can be set in the system file via the [Property Editor](#). Different values are set depending on the cross weight setup method.



Ride Height Maps

In some specific events available in VI-CarRealTime such as the **QuasiStatic** and **MaxPerformance**, a performance predictor algorithm called **VI-SpeedGen** it is used.

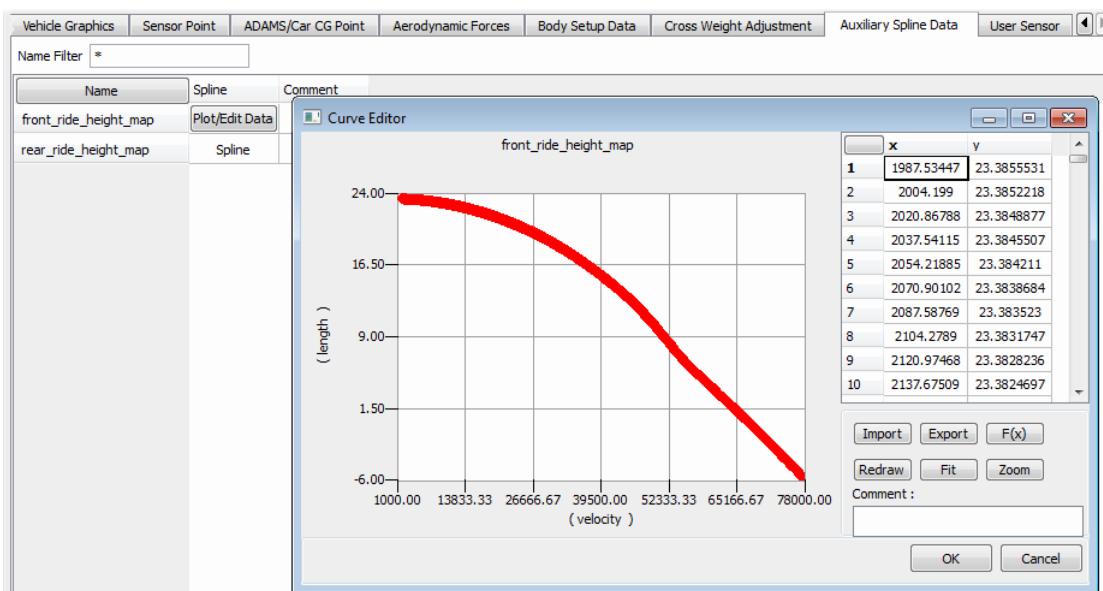
VI-SpeedGen has an internal simplified vehicle model which needs to estimate the dependency of ride heights on vehicle velocity.

This information can be stored in the body subsystem file and modified using a specific Property Editor.

Please refer to the [event](#) documentation to learn how to generate ride height maps.

By default, the following two splines are used to characterize ride heights:

- **front_ride_height_map**
front ride height mapped as a function of vehicle longitudinal velocity.
- **rear_ride_height_map**
rear ride height mapped as a function of vehicle longitudinal velocity.



As an alternative, ride height maps can be mapped as a function of the front and rear axle load. To activate ride height computation as a function of axle load [experimental_mode](#) parameter must be set to 1 (its default value is 0, that is ride height maps mapped as a function of longitudinal velocity).

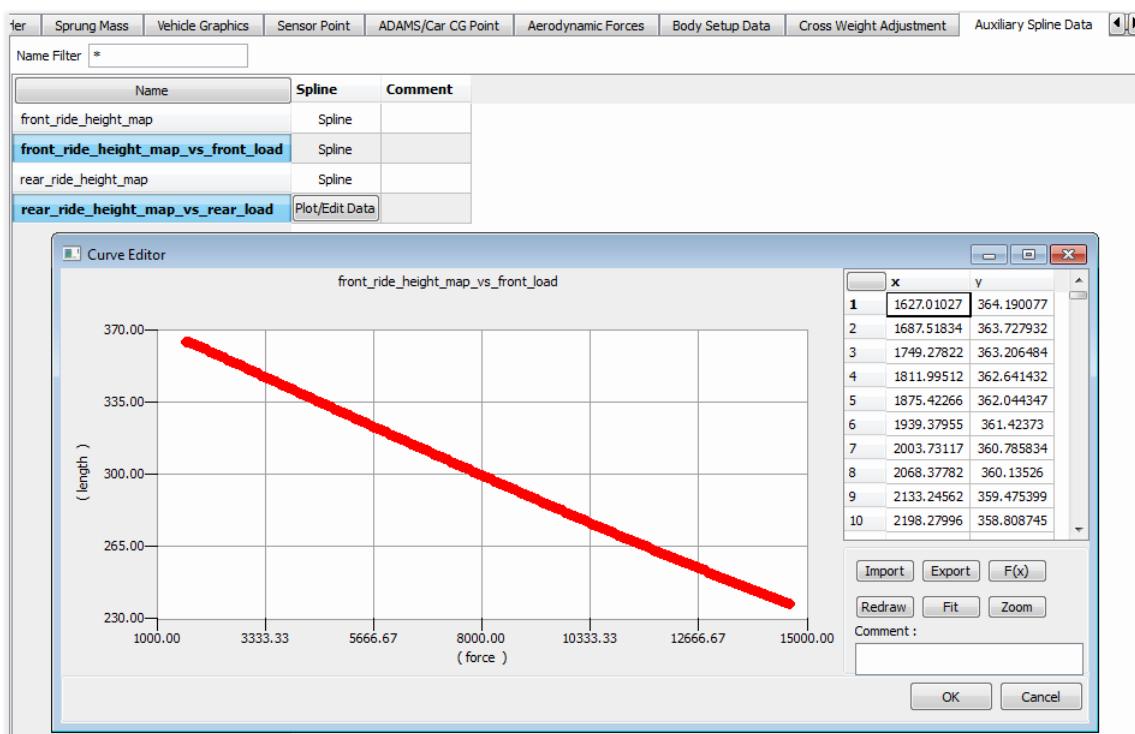
In such case, the following two splines will be used to characterize ride heights:

- **front_ride_height_map_vs_front_load**

front ride height mapped as a function of total load on the front axle

- **rear_ride_height_map_vs_rear_load**

rear ride height mapped as a function of total load on the rear axle



User Sensor

VI-CarRealTime includes the capability of adding user sensors to the model.

The type of supported sensors are:

- **Lap Sensor**

the sensor gives to the user the capability of measuring the time taken to complete a full track course. Lap sensor is unique in VI-CarRealTime model; it cannot be created by the user and track path must be selected in [Event Editor](#) in order to activate the sensor.

- **Beacon Sensor**

the sensor gives to the user the capability of measuring the time taken to complete a track course sector. Beacon sensors can be multiple and can be optionally added by the user in the table view editor.

- **Displacement Sensor**

the sensor gives the ability of measuring relative translational displacements among parts belonging to the vehicle model.

- **Velocity Sensor**

the sensor gives the ability of measuring relative translational velocities among parts belonging to the vehicle model.

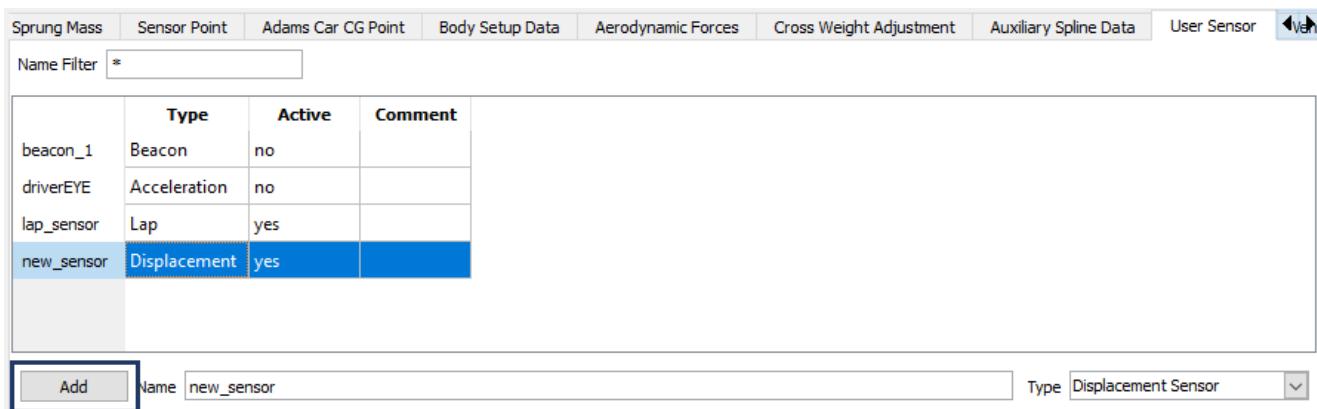
- **[Acceleration Sensor](#)**

the sensor gives the ability of measuring relative translational accelerations among parts belonging to the vehicle model.

- **Acceleration Sensor (with G)**

same concept as Acceleration Sensor but also the gravity force is taken into account.

To add a new sensor, select the **Type**, enter a **Name** and press Add button:



Once the sensor has been created, it will be added in the sensor list.

Right-clicking on the sensor item, it is possible to:

- **Rename**

it allows the user to change the sensor name. The option is not available for lap and beacon sensors.

- **Delete**

remove the sensor. The option is not available for lap and beacon sensors.

For each of the user sensors the user can specify the name of the result set and of the component in which specific results can will be stored.

Note: When using a VI-CarRealTime model featuring user sensors in [MATLAB](#) Simulink the sensor channels can be automatically added to the output bus. The feature is active when the communication mode is set to input file.

For lap and beacon sensors the user can specify the result set and component for output channels. Output data are:

- **Trigger Path S**

the parameter is used to define beacon sensor position along the track. (available only for beacon sensor)

- **Time Output Channel**

total lap time (lap sensor) or time from finish line to beacon (beacon sensor).

- **Lap Output Channel**

lap count for the given sensor.

- **Sector Time Output Channel**

time taken to go from preceding to current sensor.

- **Progressive Time Output Channel**

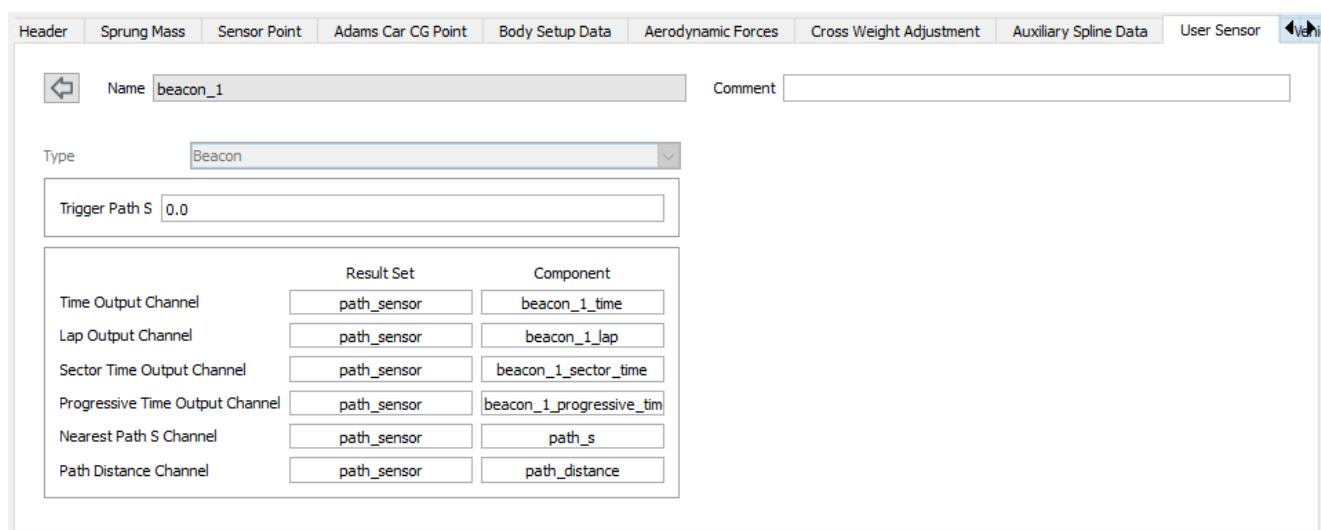
current progressive time of the sensor.

- **Nearest Path S Channel**

returns the nearest arc length value of the lap sensor path with respect to the current position of the driver reference frame.

- **Path Distance Channel**

distance of the driver sensor reference from the reference trajectory.

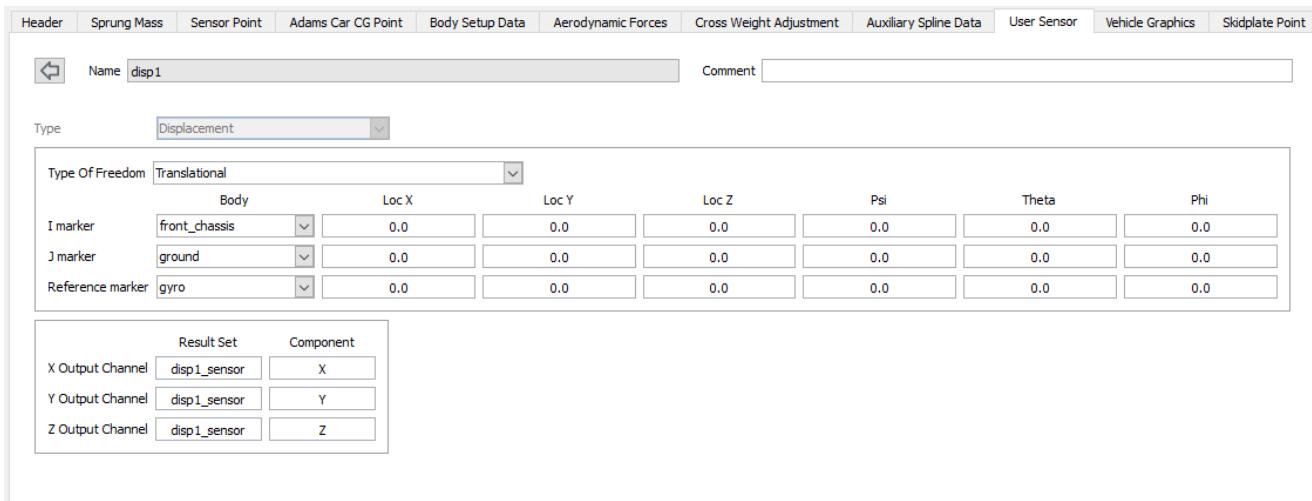


For creating custom displacement measures it is possible to use the displacement sensor; it relies on three reference markers (I, J, Reference). For each of the three markers it is possible to specify the part of the model to which it belongs, the initial location and orientation at design time and it returns the components of the translational displacement vector from J marker to I marker as expressed in the Reference Marker coordinate system.

The location and orientation must be entered with respect to [Vehicle Reference System](#).

For displacement sensor markers (I,J,Reference) the following parameters can be set:

- **Type Of Freedom**
specifies the type of measures to be computed for the sensor. Available choices are:
 - [Translational](#)
 - [Rotational](#)
- **Body**
it specifies the part to which the marker is belonging; available options are: *front_chassis*, *rear_chassis*, *front_left_suspension*, *front_right_suspension*, *rear_left_suspension*, *rear_right_suspension*, *engine*, *gyro*, *ground* .
- **Loc X, Loc Y, Loc Z**
the parameters specify the components of the marker location in [Vehicle Reference System](#) at design time.
- **Psi, Theta, Phi**
the parameters specify the components of the marker orientation expressed as Euler angles (Body 3-1-3 rotation).



Translational Displacement

The translational displacement sensor measures the displacement of the I-Marker with respect to J-Marker. The resulting vector is projected along the X,Y,Z axis of the Reference Marker.

$$D_X = (\vec{R_I} - \vec{R_J}) \cdot \vec{x_R}$$

$$D_Y = (\vec{R_I} - \vec{R_J}) \cdot \vec{y_R}$$

$$D_Z = (\vec{R_I} - \vec{R_J}) \cdot \vec{z_R}$$

where:

R_I is the displacement of the I Marker in the global reference system.

R_J is the displacement of the J Marker in the global reference system.

x_R is the unit vector along the x-axis of the Reference Marker.

y_R is the unit vector along the y-axis of the Reference Marker.

z_R is the unit vector along the z-axis of the Reference Marker.

Rotational Displacement

The rotational displacement sensor measures the angles of the I Marker with respect to the J Marker.

Three different methods are available:

- **AX AY AZ**

the sensor outputs the AX, AY, AZ output channels, computed according to the formula described below.

Type Of Freedom	Rotational						
Method	AX_AY_AZ						
	Body	Loc X	Loc Y	Loc Z	Psi	Theta	Phi
I marker	front_chassis	0.0	0.0	0.0	0.0	0.0	0.0
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0
Result Set		Component					
AX Output Channel	disp1_sensor	AX					
AY Output Channel	disp1_sensor	AY					
AZ Output Channel	disp1_sensor	AZ					

$$A_X = \tan^{-1}\left(\frac{y_I \cdot z_J}{y_I \cdot y_J}\right)$$

$$A_Y = \tan^{-1}\left(\frac{z_I \cdot x_J}{z_I \cdot z_J}\right)$$

$$A_Z = \tan^{-1}\left(\frac{x_I \cdot y_J}{x_I \cdot x_J}\right)$$

where:

x_I is the x-axis of the I Marker.

y_I is the y-axis of the I Marker.

z_I is the z-axis of the I Marker.

x_J is the x-axis of the J Marker.

y_J is the y-axis of the J Marker.

z_J is the z-axis of the J Marker.

• represents the dot product.

• Euler Angles

the sensor outputs the Euler Angles PSI, THETA and PHI (Body 3-1-3 rotation)

Type Of Freedom	Rotational						
Method	Euler_Angles						
	Body	Loc X	Loc Y	Loc Z	Psi	Theta	Phi
I marker	front_chassis	0.0	0.0	0.0	0.0	0.0	0.0
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0
Result Set		Component					
PSI Output Channel	disp1_sensor	PSI					
THETA Output Channel	disp1_sensor	THETA					
PHI Output Channel	disp1_sensor	PHI					

• Yaw Pitch Roll

the sensor outputs the YAW, PITCH and ROLL angles (Body 3-2-1 rotation)

Type Of Freedom	Rotational																	
Method	Yaw_Pitch_Roll																	
I marker	front_chassis	Loc X	Loc Y	Loc Z	Psi	Theta	Phi											
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0											
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0											
<table border="1"> <tr> <td>Result Set</td> <td>Component</td> </tr> <tr> <td>YAW Output Channel</td> <td>disp1_sensor</td> <td>YAW</td> </tr> <tr> <td>PITCH Output Channel</td> <td>disp1_sensor</td> <td>PITCH</td> </tr> <tr> <td>ROLL Output Channel</td> <td>disp1_sensor</td> <td>ROLL</td> </tr> </table>								Result Set	Component	YAW Output Channel	disp1_sensor	YAW	PITCH Output Channel	disp1_sensor	PITCH	ROLL Output Channel	disp1_sensor	ROLL
Result Set	Component																	
YAW Output Channel	disp1_sensor	YAW																
PITCH Output Channel	disp1_sensor	PITCH																
ROLL Output Channel	disp1_sensor	ROLL																

The velocity sensor can be used for velocity measurements and it also relies on three markers (I, J, Reference). For each of the three markers it is possible to specify the part of the model to which it belongs, the initial location and orientation at design time and it returns the components of the translational velocity vector from J marker to I marker as expressed in the Reference Marker coordinate system.

The location and orientation must be entered with respect to [Vehicle Reference System](#).

For velocity sensor markers (I,J,Reference) the following parameters can be set:

- Type Of Freedom**

specifies the type of measures to be computed for the sensor. Available choices are:

- [Translational](#)
- [Rotational](#)

- Body**

it specifies the part to which the marker is belonging; available options are: *front_chassis* , *rear_chassis* , *front_left_suspension* , *front_right_suspension* , *rear_left_suspension* , *rear_right_suspension* , *engine* , *gyro* , *ground* ;

- Loc X, Loc Y, Loc Z**

the parameters specify the components of the marker location in [Vehicle Reference System](#) at design time.

- Psi, Theta, Phi**

the parameters specify the components of the marker orientation expressed as Euler angles (*Body* 3-1-3 rotation).

Header	Sprung Mass	Sensor Point	Adams Car CG Point	Body Setup Data	Aerodynamic Forces	Cross Weight Adjustment	Auxiliary Spline Data	User Sensor	Vehicle Graphics	Skidplate Point											
Name	vel1	Comment																			
Type	Velocity																				
Type Of Freedom	Translational	Loc X	Loc Y	Loc Z	Psi	Theta	Phi														
I marker	front_chassis	0.0	0.0	0.0	0.0	0.0	0.0														
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0														
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0														
<table border="1"> <tr> <td>Result Set</td> <td>Component</td> </tr> <tr> <td>X Output Channel</td> <td>vel1_sensor</td> <td>VX</td> </tr> <tr> <td>Y Output Channel</td> <td>vel1_sensor</td> <td>YY</td> </tr> <tr> <td>Z Output Channel</td> <td>vel1_sensor</td> <td>VZ</td> </tr> </table>											Result Set	Component	X Output Channel	vel1_sensor	VX	Y Output Channel	vel1_sensor	YY	Z Output Channel	vel1_sensor	VZ
Result Set	Component																				
X Output Channel	vel1_sensor	VX																			
Y Output Channel	vel1_sensor	YY																			
Z Output Channel	vel1_sensor	VZ																			

Translational Velocity

The translational velocity sensor measures the difference between the velocity vector of the I Marker and the velocity vector of the J Marker. The first derivative of the vectors are computed in the global reference system. The resulting vector is projected along the X,Y,Z axis of the Reference Marker.

$$v_x = \left(\frac{d}{dt} \vec{R}_I - \frac{d}{dt} \vec{R}_J \right) \cdot \vec{x}_R$$

$$v_y = \left(\frac{d}{dt} \vec{R}_I - \frac{d}{dt} \vec{R}_J \right) \cdot \vec{y}_R$$

$$v_z = \left(\frac{d}{dt} \vec{R}_I - \frac{d}{dt} \vec{R}_J \right) \cdot \vec{z}_R$$

where:

R_I is the displacement of the I-Marker in the global reference system.

R_J is the displacement of the J-Marker in the global reference system.

x_R is the unit vector along the x-axis of the Reference Marker.

y_R is the unit vector along the y-axis of the Reference Marker.

z_R is the unit vector along the z-axis of the Reference Marker.

Rotational Velocity

The rotational velocity sensor measures the difference between the angular velocity vector of the part defined by the I Marker and the angular velocity vector of the part defined by the J Marker. The resulting vector is projected along the X,Y,Z axis of the Reference Marker.

Type Of Freedom	Rotational						
I marker	front_chassis	Loc X 0.0	Loc Y 0.0	Loc Z 0.0	Psi 0.0	Theta 0.0	Phi 0.0
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0

Result Set	Component	
WX Output Channel	vel1_sensor	WX
WY Output Channel	vel1_sensor	WY
WZ Output Channel	vel1_sensor	WZ

$$w_x = (\omega_I - \omega_J) \cdot x_R$$

$$w_y = (\omega_I - \omega_J) \cdot y_R$$

$$w_z = (\omega_I - \omega_J) \cdot z_R$$

where:

ω_I is the angular velocity of the part defined by the I Marker in the global reference system.

ω_J is the angular velocity of the part defined by the J Marker in the global reference system.

x_R is the unit vector along the x-axis of the Reference Marker.

y_R is the unit vector along the y-axis of the Reference Marker.

z_R is the unit vector along the z-axis of the Reference Marker.

User acceleration measurements can be computed in VI-CarRealTime using the acceleration (or acceleration with gravity) sensor, which is defined through I, J and Reference markers.

The sensor returns the components of the translational acceleration vector from J marker to I marker as expressed in the Reference Marker coordinate system.

For the acceleration sensor markers (I,J,Reference) the following parameters can be set:

- **Type Of Freedom**

specifies the type of measures to be computed for the sensor. Available choices are:

- [Translational](#)
- [Rotational](#)

- **Body**

it specifies the part to which the marker is belonging; available options are: `front_chassis` , `rear_chassis` , `front_left_suspension` , `front_right_suspension` , `rear_left_suspension` , `rear_right_suspension` , `engine` , `gyro` , `ground` ;

- **Loc X, Loc Y, Loc Z**

the parameters specify the components of the marker location in [Vehicle Reference System](#) at design time.

- **Psi, Theta, Phi**

the parameters specify the components of the marker orientation expressed as Euler angles (*Body 3-1-3 rotation*).

Header	Sprung Mass	Sensor Point	Adams Car CG Point	Body Setup Data	Aerodynamic Forces	Cross Weight Adjustment	Auxiliary Spline Data	User Sensor	Vehicle Graphics	Skidplate Point			
<input type="button" value="New"/> Name	acc1							Comment					
Type	Acceleration												
Type Of Freedom	Translational												
I marker	front_chassis	Loc X	0.0	Loc Y	0.0	Loc Z	0.0	Psi	0.0	Theta	0.0	Phi	0.0
J marker	ground		0.0		0.0		0.0		0.0		0.0		
Reference marker	gyro		0.0		0.0		0.0		0.0		0.0		
X Output Channel	acc1_sensor	Result Set	ACCX	Component									
Y Output Channel	acc1_sensor		ACCY										
Z Output Channel	acc1_sensor		ACCCZ										

Translational Acceleration

The translational acceleration sensor measures the difference between the acceleration vector of the I Marker and the acceleration vector of the J Marker. The second derivative of the vectors are computed in the global reference system. The resulting vector is projected along the X,Y,Z axis of the Reference Marker.

$$A_x = \left(\frac{d^2}{dt^2} \vec{R}_I - \frac{d^2}{dt^2} \vec{R}_J \right) \cdot \vec{x}_R$$

$$A_y = \left(\frac{d^2}{dt^2} \vec{R}_I - \frac{d^2}{dt^2} \vec{R}_J \right) \cdot \vec{y}_R$$

$$A_z = \left(\frac{d^2}{dt^2} \vec{R}_I - \frac{d^2}{dt^2} \vec{R}_J \right) \cdot \vec{z}_R$$

where:

R_I is the displacement of the I-Marker in the global reference system.

R_J is the displacement of the J-Marker in the global reference system.

x_R is the unit vector along the x-axis of the Reference Marker.

y_R is the unit vector along the y-axis of the Reference Marker.

z_R is the unit vector along the z-axis of the Reference Marker.

When an acceleration sensor with gravity is used, also the gravity field is considered in the computed acceleration between I Marker and J Marker.

The following conventions applies:

Vehicle at a standstill:

- Az = 0
- Az (with gravity) = +g

Free falling vehicle:

- Az = -g
- Az (with gravity) = 0

Rotational Acceleration

The rotational acceleration sensor measures the difference between the derivative of the angular velocity vector (ω_I) of the part defined by the I Marker and the derivative of the angular velocity vector (ω_J) of the part defined by the J Marker. The derivative of the angular velocity vectors are computed in global reference system. The resulting vector is projected along the X,Y,Z axis of the Reference Marker.

Type Of Freedom	Rotational						
	Body	Loc X	Loc Y	Loc Z	Psi	Theta	Phi
I marker	front_chassis	0.0	0.0	0.0	0.0	0.0	0.0
J marker	ground	0.0	0.0	0.0	0.0	0.0	0.0
Reference marker	gyro	0.0	0.0	0.0	0.0	0.0	0.0

	Result Set	Component
WDTX Output Channel	acc1_sensor	WDTX
WDTY Output Channel	acc1_sensor	WDTY
WDTZ Output Channel	acc1_sensor	WDTZ

$$WDT_X = \left(\frac{d}{dt} \omega_I - \frac{d}{dt} \omega_J \right) \cdot x_R$$

$$WDT_Y = \left(\frac{d}{dt} \omega_I - \frac{d}{dt} \omega_J \right) \cdot y_R$$

$$WDT_Z = \left(\frac{d}{dt} \omega_I - \frac{d}{dt} \omega_J \right) \cdot z_R$$

where:

ω_I is the angular velocity of the part defined by the I Marker in the global reference system.

ω_J is the angular velocity of the part defined by the J Marker in the global reference system.

x_R is the unit vector along the x-axis of the Reference Marker.

y_R is the unit vector along the y-axis of the Reference Marker.

z_R is the unit vector along the z-axis of the Reference Marker.

Rigid Part

VI-CarRealTime supports the possibility of including in the body subsystem model additional parts, which can be separated from the main chassis mass and connected to it through bushing elements.

The feature is optional and is activated for the models generated through the Adams Car plugin when the body template includes a group named [front_body_parts](#) and/or a group named [rear_body_parts](#). Each group must contain the list of the parts forming respectively the front rigid part and the rear rigid part.

Chassis Rigid Part panel embeds the following sub-panels:

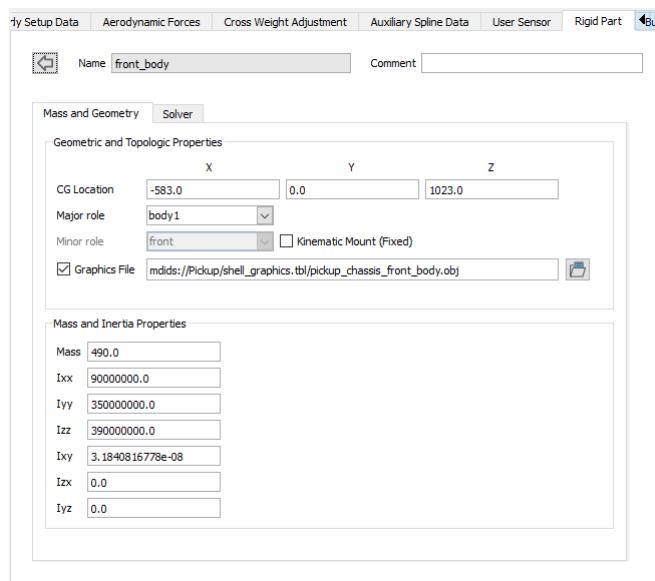
- [Mass and Geometry](#)
- [Solver](#)

Note: an example of body subsystem (*RT_pickup_body.xml*) including multiple rigid parts is shipped with *pickup* database.

The following additional outputs are available when rigid parts are defined in body subsystem:

- [Body Part Acceleration](#)
- [Body Part Displacement](#)
- [Body Part Velocity](#)

Rigid chassis part data are input through the widget described in the snapshot below and features the following parameters:



Geometric and Topological Properties

- **CG Location**

X,Y,Z Part CG coordinates expressed in [Global Reference](#) frame.

- **Major Role**

Attribute which identifies the rigid part. Available choices are:

- body1
- body2
- body3

The attribute is referenced in [bushing](#) I/J Body field.

- **Minor Role**

Attribute which specifies whether the rigid part is a front or a rear part. The option menu is active only when Kinematic Mount (Fixed) toggle is checked; otherwise, the part connection with other parts is defined in the Bushings panel.

- **Kinematic Mount (Fixed)**

If deactivated the body part will be considered mounted to the main chassis with compliance attachments (using [bushing](#)); if activated the body part will be mounted to the main chassis with kinematic constraint (fixed joint).

- **Graphics File**

When the toggle is checked, the field allows the user to set the wavefront graphics file of the rigid part.

Mass and Inertia Properties

Mass

Mass of the rigid part.

I_{xx}

I_{xx} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yy}

I_{yy} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{zz}

I_{zz} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{xy}

I_{xy} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

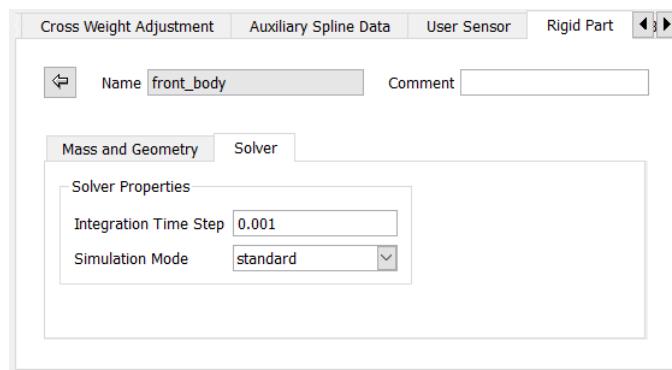
I_{xz}

I_{xz} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yz}

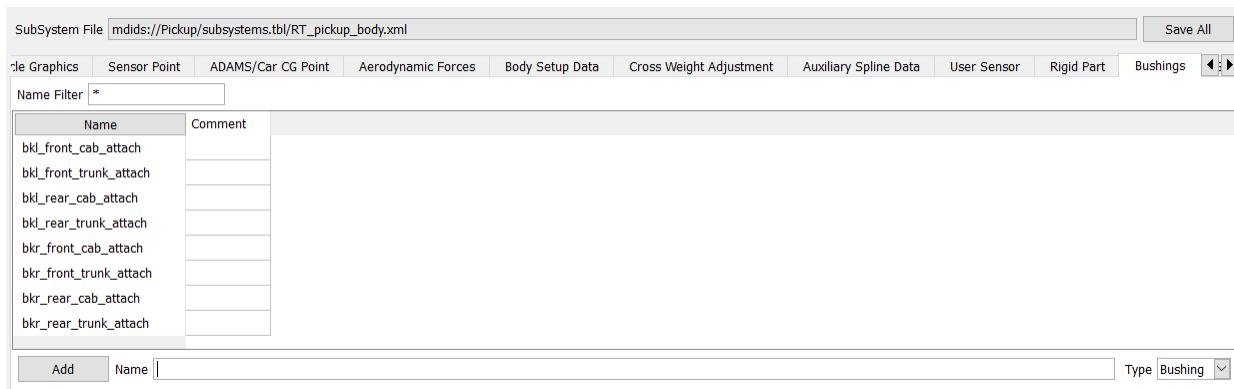
I_{yz} moment of inertia of the rigid part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

Each chassis rigid part is integrated with its own Integration Time Step, independently with respect to the [Integration Time Step](#) set in the [Test Mode](#).

**Bushings**

For body subsystem including at least a [Rigid Part](#), the connectivity to the main chassis is realized either through kinematic mount (rigid joint) or through bushing elements.

When at least one bushing instance exists in the model, additional bushings can be created in VI-CarRealTime by simply entering a name for the new bushing and then selecting the **Add** button.



Two different types of bushing are available:

- [Standard Bushing](#)
- [Frequency Bushing](#)
- [VI-MxMount Bushing](#)

The following parameters are common for both bushing types:



The **Active** toggle button allows the user to activate/deactivate the bushing.

Topological Properties

- **Bushing Type**

Allows the user to select the type of the bushing. Available choices are: *standard* , *frequency* .

- **I Body**

Specify the first body where the bushing is attached to.

- **J Body**

Specify the second body where the bushing is attached to.

Geometric Properties

- **Location**

X,Y,Z locations, expressed in [Vehicle Reference System](#) at design time.

- **Orientation**

Psi, Theta, Phi angles, expressed in [Vehicle Reference System](#) at design time.

Stiffness Properties

- **Elastic Force**

Use spline to define characteristic curves (bushing force vs translational deformation).

- **Elastic Torque**

Use spline to define characteristic curves (bushing torque vs rotational deformation).

Damping Properties

- **Translational Damping**

Constant values defining the three translational damping coefficients. The coefficients multiply the relative translational velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

- **Rotational Damping**

Constant values defining the three rotational damping coefficients. The coefficients multiply the relative angular velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

Preload

- **Translational Preload**

Initial force applied to the bushing in design conditions.

- **Rotational Preload**

Initial torque applied to the bushing in design conditions.

Offset

- **Translational Offset**

Initial translational deformation applied to the bushing in design conditions.

- **Rotational Offset**

Initial angular deformation applied to the bushing in design conditions.

Scaling Factors

- **Translational Scalings**

Constant values defining the three translational scaling factors which multiply the bushing Elastic Force characteristic.

- **Rotational Scalings**

Constant values defining the three rotational scaling factors which multiply the bushing Elastic Torque characteristic.

When standard bushing type is selected, the following additional panels appear:

Element Properties		Along X	Along Y	Along z	About X	About Y	About z
Elastic Properties	Plot/Edit Data						
Damping Properties	10.0	10.0	10.0	11.0	11.0	11.0	
Preload	0.0	0.0	0.0	0.0	0.0	0.0	
Offset	0.0	0.0	0.0	0.0	0.0	0.0	

Scaling Factors		Along X	Along Y	Along z	About X	About Y	About z
Force	1.0	1.0	1.0	1.0	1.0	1.0	

Element Properties

- **Elastic Properties (Spline)**

Use spline to define characteristic curves (bushing force vs translational deformation & bushing torque vs rotational deformation)

- **Damping Properties**

Constant values defining the translational and rotational damping coefficients. The coefficients multiply the relative translational and angular velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

- **Preload**

Initial forces and torques applied to the bushing in design conditions.

- **Offset**

Initial translational and angular deformations applied to the bushing in design conditions.

Scaling Factors

- **Force**

Constant values defining the translational and rotational scaling factors which multiply the bushing Elastic characteristic.

Damping Properties

- **Translational Damping**

Constant translational damping value (force/velocity units).

- **Rotational Damping**

Constant translational damping value (torque/angular velocity units).

The standard bushing allows to define a force connection between two parts ([I Body](#) and [J Body](#)) in the location defined by the user in [Geometric Properties](#) tab.

For each degree of freedom a force (torque), function of relative displacement (rotation) and velocity (angular velocity), will be applied between the two parts.

The elastic contribute of the force (torque) is mapped as a function of the relative displacement (rotation) in the Elastic Properties section.

The damping contribute is defined by means of a constant value in Damping Properties section.

When frequency bushing type is selected, the following additional panel appears:

Geometric Properties						
Location	X	Y	Z	Psi	Theta	Phi
	-360.0	0.0	412.0	-180.0	0.0	0.0
Element Properties						
Elastic Properties	Along X	Along Y	Along z	About X	About Y	About z
Plot/Edit Data	Plot/Edit Data	Plot/Edit Data	Plot/Edit Data	Plot/Edit Data	Plot/Edit Data	Plot/Edit Data
Loss Angle Properties	0.0	0.0	0.0	0.0	0.0	0.0
Preload	0.0	0.0	0.0	0.0	0.0	0.0
Offset	0.0	0.0	0.0	0.0	0.0	0.0
Scaling Factors						
Force	Along X	Along Y	Along z	About X	About Y	About z
1.0	1.0	1.0	1.0	1.0	1.0	1.0

Element Properties

- **Elastic Properties (Spline)**

Use spline to define characteristic curves (bushing force vs translational deformation & bushing torque vs rotational deformation)

- **Loss Angles**

Constant loss angle for the translational and rotational degrees of freedom

- **Preload**

Initial forces and torques applied to the bushing in design conditions.

- **Offset**

Initial translational and angular deformations applied to the bushing in design conditions.

Scaling Factors

- **Force**

Constant values defining the translational and rotational scaling factors which multiply the bushing Elastic characteristic.

Damping Properties

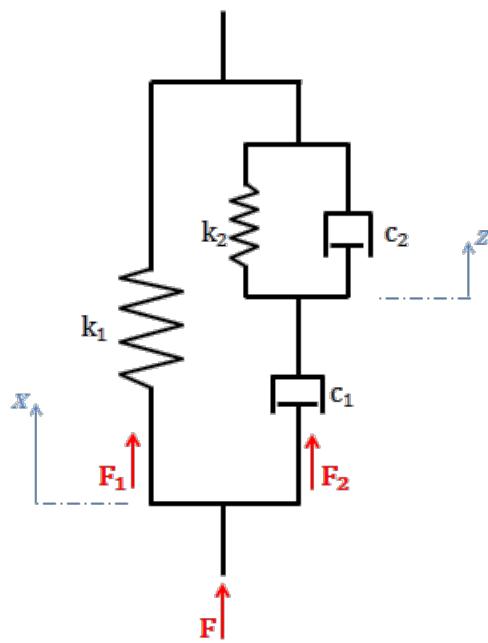
- **Translational Damping**

Constant translational damping value (force/velocity units).

- **Rotational Damping**

Constant translational damping value (torque/angular velocity units).

The frequency bushing relies on the following model:



The force F of the frequency bushing is computed as the sum between F_1 and F_2 , where:

$$F_1 = k_1 \cdot x$$

$$F_2 = c_1 \cdot (\dot{x} - \dot{z}) = c_2 \cdot \dot{z} + k_2 \cdot z$$

The total force is:

$$F = F_1 + F_2 = k_1 \cdot x + c_1 \cdot (\dot{x} - \dot{z})$$

Defining the following auxiliary parameters:

$$\alpha = \frac{k_2}{k_1}$$

$$\beta = \frac{c_2}{c_1}$$

$$\gamma = \frac{c_1}{k_1}$$

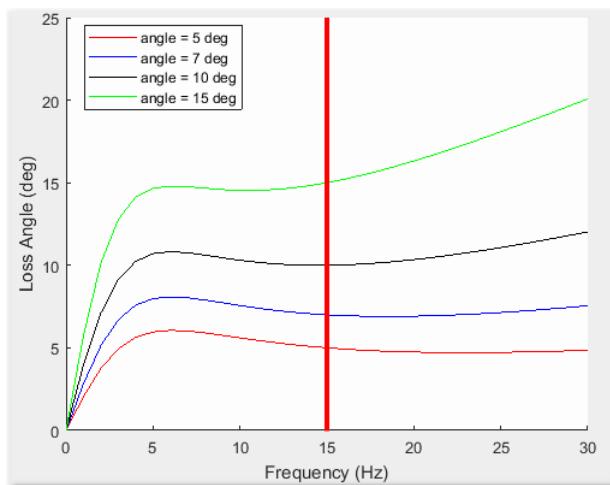
and expressing z as a function of x and \dot{z} we have:

$$F = k_1 \cdot [x + \gamma \cdot (\dot{x} - \dot{z})]$$

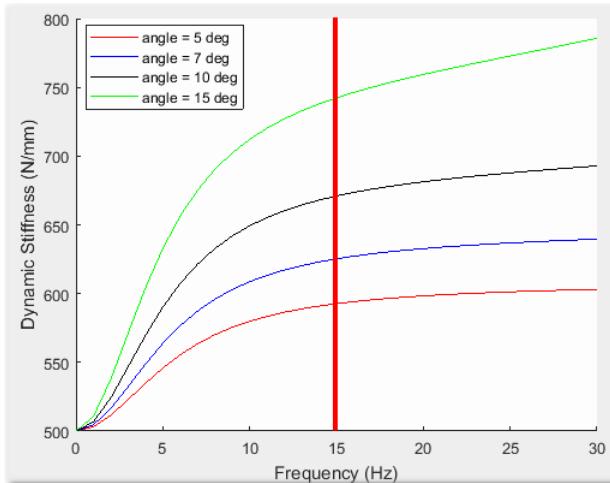
$$\dot{z} = \frac{1}{1 + \beta} \cdot \left(\dot{x} - \frac{\alpha}{\gamma} \cdot z \right)$$

k_1 coefficient is computed by interpolating the spline data from the related Stiffness Properties table.

The Fourier Transform of the F is computed and the phase vs. frequency curve is used to scale the coefficients α, β and γ in order to obtain the target loss angle value at 15 Hz:



The dynamic stiffness varies accordingly with the loss angle:



Once α, β and γ coefficients have been computed it is possible to compute the frequency bushing force.

VI-CarRealTime

When the `vi_mount` option is selected as *Bushing Type* then the following panel is shown.

Topological Properties

Bushing Type	vi_mount
I Body	body1
J Body	front_chassis

VI-mount Property File

Property File	mdids://carrealtimeshared/bushings.tbl/hydromount_example.ubf	<input type="button" value="..."/>	<input type="button" value="Edit"/>
---------------	---	------------------------------------	-------------------------------------

Geometric Properties

Location	X	Y	Z	Psi	Theta	Phi
	117.0	400.0	723.0	-180.0	0.0	0.0

Element Properties

	Along X	Along Y	Along z	About X	About Y	About z
Force Model	elastomer	elastomer	hydromount	elastomer	elastomer	elastomer
Preload	0.0	0.0	0.0	0.0	0.0	0.0
Offset	0.0	0.0	0.0	0.0	0.0	0.0

Scaling Factors

	Along X	Along Y	Along z	About X	About Y	About z
Force	1.0	1.0	1.0	1.0	1.0	1.0
Displacement	1.0	1.0	1.0	1.0	1.0	1.0
Rubber Stiffness	1.0	1.0	1.0	1.0	1.0	1.0
Chamber Stiffness	1.0	1.0	1.0	1.0	1.0	1.0
Resonance Frequency	1.0	1.0	1.0	1.0	1.0	1.0

The following additional panels contain the specific VI-MxMount parameters:

VI-mount Property File

- **Property file**
Property file path for the VI-MxMount bushing (.ubf file extension).

Element Properties**• Force model**

Model type for each direction (*elastomer /hydrobushing /hydromount /none*). The type is available for the selection if found in the property file for the given direction. Select *none* to get a null force.

• Force preload

Output force/torque offset.

• Displacement offset

Input displacement/rotation offset.

Scaling Factors**• Force**

Output force/torque scaling.

• Displacement

Input displacement/rotation scaling.

• Rubber stiffness

Scaling for rubber stiffness .

• Chamber stiffness

Scaling for chamber stiffness (enabled for *hydrobushing /hydromount*) .

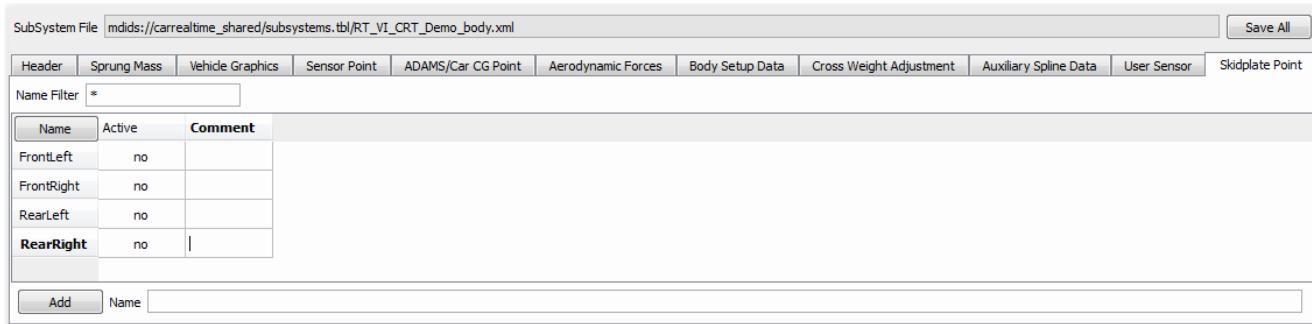
- **Resonance frequency**

Scaling for resonance frequency (enabled for *hydribushing / hydromount*) .

For more information about the elastomer model, the hydribushing/hydromount types and the property file format, refer to the [UBF file documentation](#).

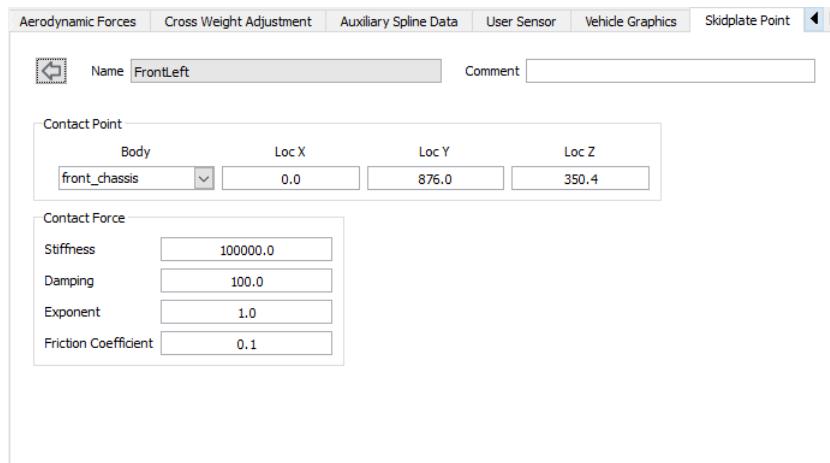
Skidplate Point

The Skidplate is a feature designed to simulate the impact of the chassis underbody with the track surface. The model is based on force elements that could be placed wherever the user requires by using contact point references. The action part for these forces is the chassis part. When the ride height of one of these references drop to 0 a contact force arises to prevent the ground compenetration.



The **Active** box allows the user to activate/deactivate each contact point.

By using the **Add** button it is possible to Add further contact points to compute the skidplate forces between ground and chassis part.



About the *Contact Point* , the following parameters can be set:

- **Body**

action part on which the contact force acts (front_chassis or rear_chassis). The two parts are the same chassis part when the [chassis compliance](#) is not active.

- **Loc X, Loc Y, Loc Z**

X,Y,Z location of the contact point sensor in [vehicle reference frame](#).

About the *Contact Force* , the following parameters can be set:

- **Stiffness**

stiffness k of the contact force. Note that the stiffness units are Force/(Length e).

- **Damping**

damping c of the contact force.

- **Exponent**

exponent e of the contact force.

- **Friction Coefficient**

friction coefficient α of the contact force.

When the distance between the contact point and the road becomes less than 0, the contact force is applied in the Body part.

Such force is composed by a F_z component which is oriented as the road normal in the actual contact point location and by a F_p component which lies in the road plane and has a direction given by the actual velocity of the contact point sensor projected in such plane.

$$F_z = -k \cdot \delta^e + \text{step}(-\delta, 0, \varepsilon, 0, 1) \cdot c \cdot \frac{d\delta}{dt}$$

$$F_p = \alpha \cdot F_z$$

where:

δ is the penetration depth of the contact point sensor inside the road.

ε is the penetration depth value (5 mm) beyond which the damping component of the force is 100% applied (between 0 and ε the damping force rises with a step function).

For each active contact point sensor, [9 output](#) are automatically computed.

Powertrain Subsystem

The powertrain subsystem includes data for the engine, clutch, transmission and differential. It also includes information for gear shifting strategies.

[Driveline](#)

[Engine Part](#)

[Bushings](#)

Driveline

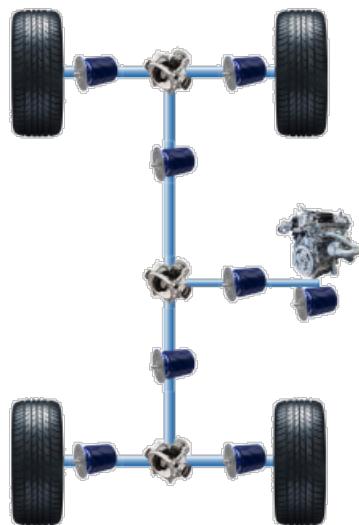
Starting with version 18, the whole driveline has been completely redesigned in order to be compliant with vehicles state of the art.

The flexibility and modularity of the new driveline allows the user to easily configure powertrain subsystem in terms of:

- main engine (internal combustion or electric)
- 8 different motors (internal combustion or electric)
- battery
- gearbox (automatic/manual)
- clutch (standard, dual)
- torque converter
- differentials (FWD, RWD, AWD)

The new driveline provides a panel ([Driveline Layout](#)) in which the user can set up all the components according to the driveline flavor he's going to model.

A dynamic image (updated on the fly) allows to visualize the actual driveline scheme:

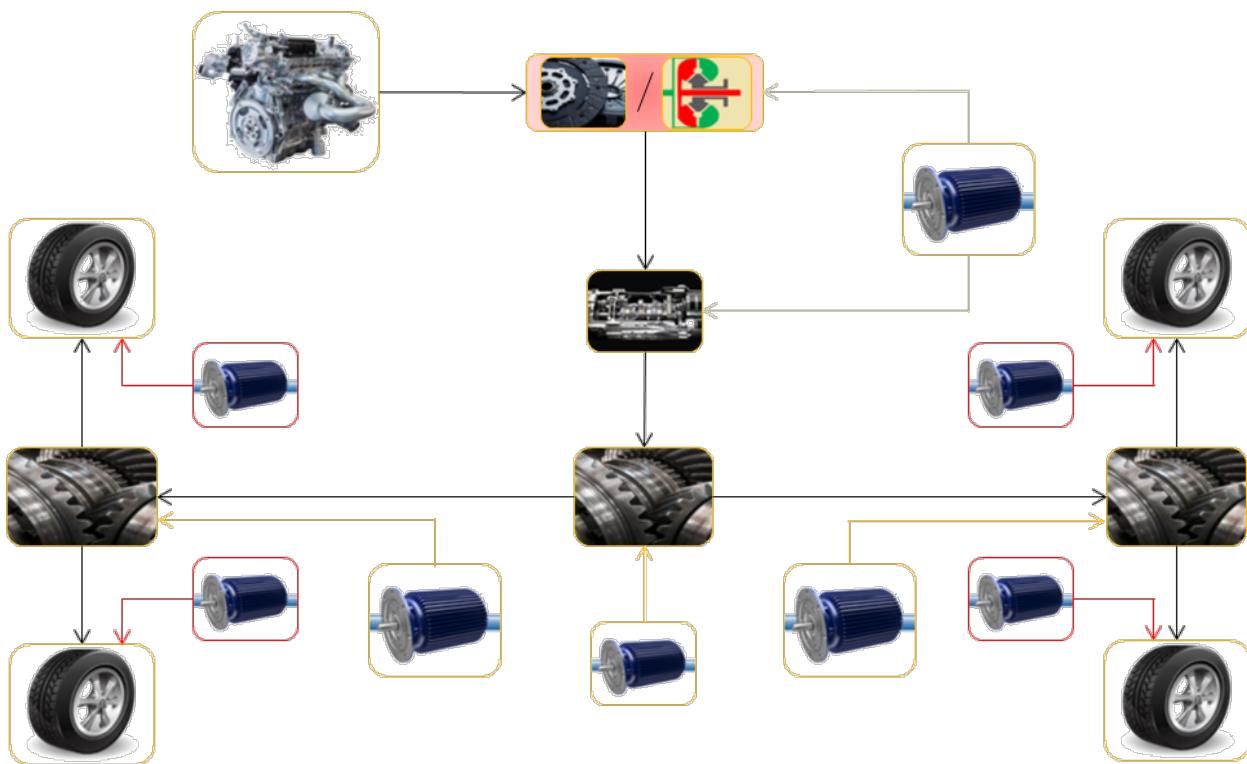


The new driveline provides an input/output interface for each of its components; the user is allowed to switch on/off each component and replace it with its own defined in an external environment (i.e. Simulink). For further details about how to replace internal driveline components with external ones by using Simulink please refer to [Implementing an Electric Vehicle](#).

Each driveline component can be activated/deactivated by using the related flag defined in [Subsystem Activation](#) Input block.

The complete list of inputs/outputs each component defines can be found in [Input Channels](#) and [Output Channels](#) chapters.

Here the diagram of the new driveline with all the supported components:

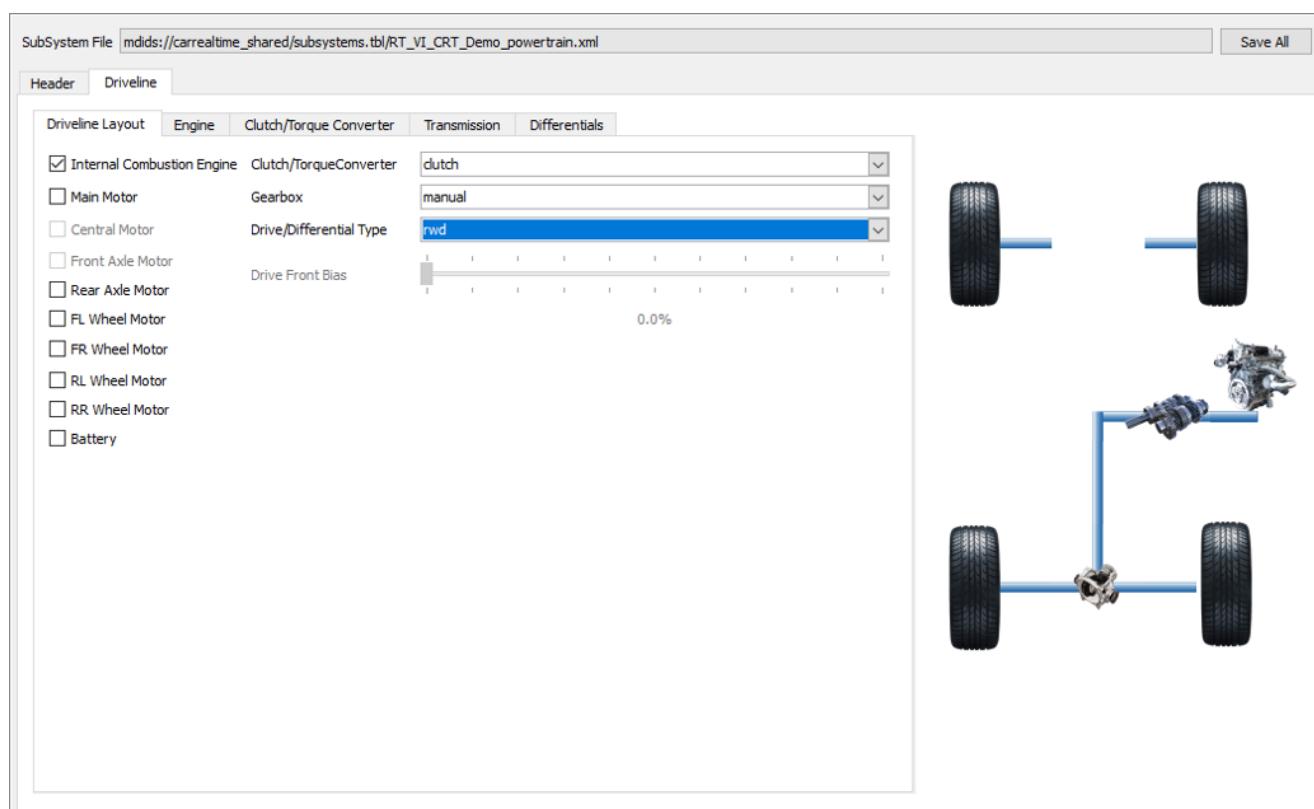


The driveline module is composed by the following sub-panels:

- [Driveline Layout](#)
- [Engine](#)
- [Motors](#)
- [Clutch/Torque Converter](#)
- [Transmission](#)
- [Differentials](#)
- [Battery](#)

The panel summarizes all driveline components that can be set. By acting on the toggle buttons on the left side of the panel, it is possible to activate/deactivate each motor; the driveline scheme image on the right side will be automatically updated depending on the flags set by the user.

Furthermore by clicking on a component in the image, the corresponding tab will be automatically selected and displayed.



Driveline layout panel allows to set the following parameters.

- **Internal Combustion Engine**
- **Main Motor**
- **Central Motor**
- **Front Axle Motor**
- **Rear Axle Motor**
- **FL Wheel Motor**
- **FR Wheel Motor**
- **RL Wheel Motor**
- **RR Wheel Motor**

The following option menus are available:

- **Clutch/TorqueConverter**
it allows to choose among:
 - [Clutch](#)
 - [Torque Converter](#)
 - *Inactive* : no clutch model is used and both torque and omega are transmitted, with unitary transmission ratio, from the engine to the transmission.
- **Gearbox**
it allows to choose among:
 - [Manual](#)
 - [Automatic](#)
 - *Inactive* : no gearbox model is used and both torque and omega are transmitted with unitary transmission ratio.
- **Drive/Differential Type**
it allows to choose among:

- [fwd](#): front wheel drive vehicle
- [rwd](#): rear wheel drive vehicle
- [awd](#): all wheel drive vehicle
- *no differentials* : all differentials are switched off, together with the related Engine/Motors.
- *dual motor awd*: it is composed by an independent front axle motor (entering the front differential) and an independent rear axle motor (entering the rear differential). Central differential is switched off.
- *fwd with rear diff* : it allows the user to define an hybrid layout, for example a front wheel drive ICE in combination with a Rear Axle Motor and a rear differential.
- *rwd with front diff* : it allows the user to define an hybrid layout, for example a rear wheel drive ICE in combination with a Front Axle Motor and a front differential.

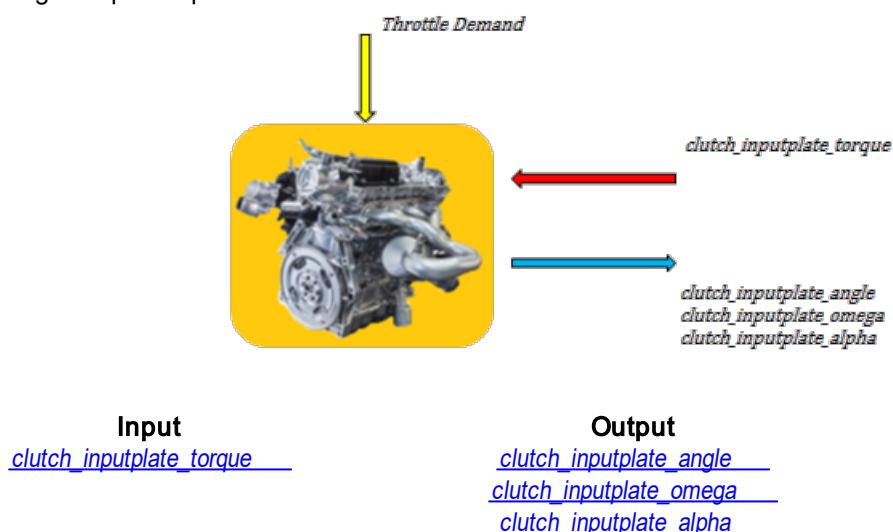
• **Drive Front Bias**

the slider is active only when Drive/Differential Type is set to **awd**. It defines how the input torque entering the central differential is split between front and rear differentials. It has to be set to 50% when the torque split is done using Limited Slip or Locked Differential.

Note: the dynamic driveline image is always updated according to the user selections on the panel, so that the actual driveline scheme adopted is always displayed.

Engine panel defines the parameters of the main engine of VI-CarRealTime model.

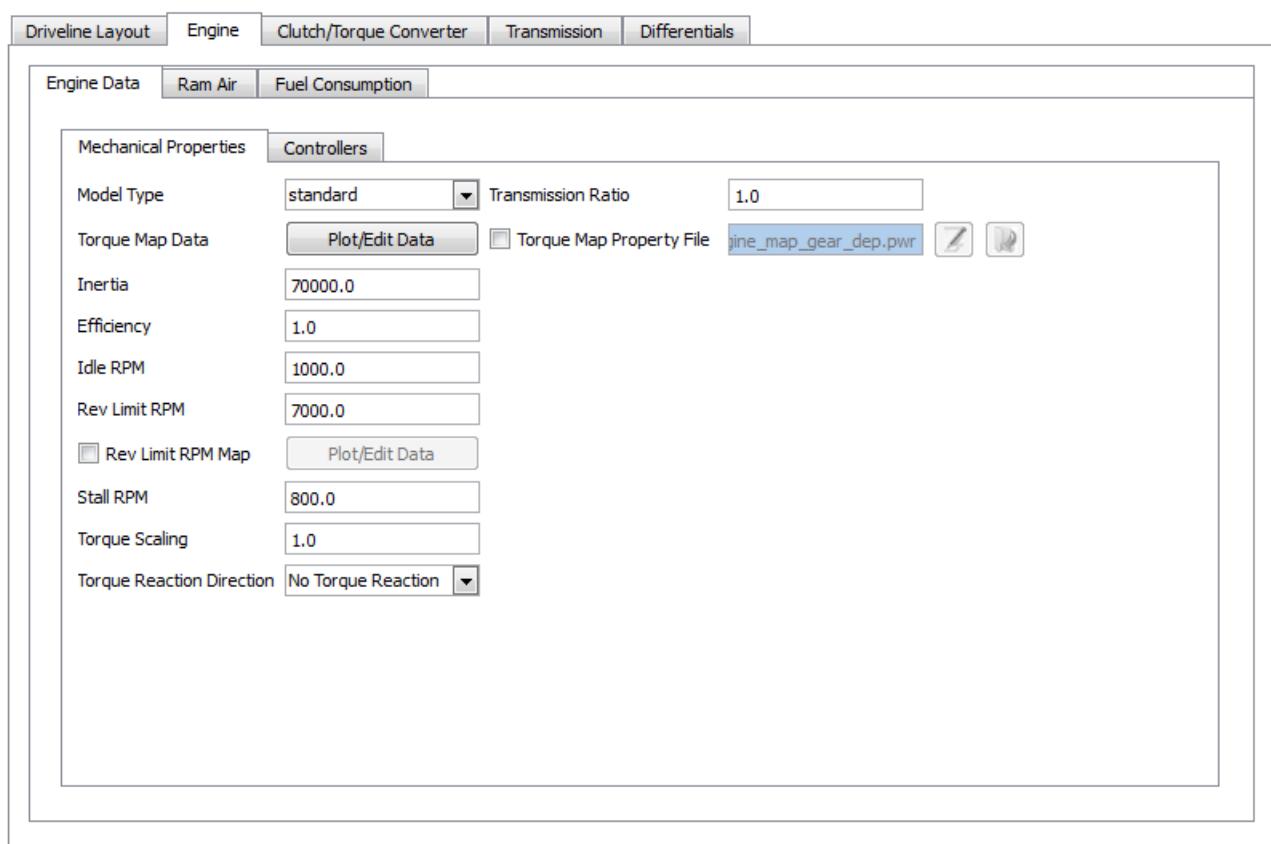
Here the engine input/output interface:



The following tabs are available:

- [Engine Data](#)
- [Ram Air](#)
- [Fuel Consumption](#)

Engine Data tab allows the user to set the main engine characteristics.

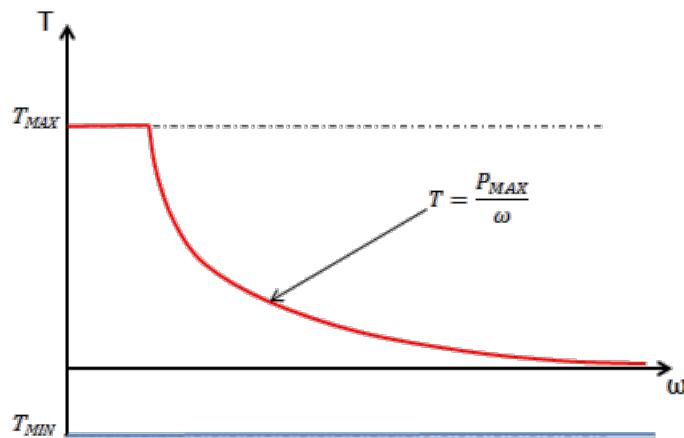


Two different Model Types are available:

- [Simple Engine](#)
- [Standard Engine](#)

Simple Engine

A simple engine is modeled as follows:



At full throttle (100%), the engine provides a torque equal to T_{MAX} until the product between T and ω is less than P_{MAX} ; then it provides a torque equals to the ratio of P_{MAX} and ω .

During coasting (throttle 0%), it generates a constant T_{MIN} .

Mechanical Properties	Controllers		
Model Type	simple	Transmission Ratio	1.0
Inertia	70000.0	Simple Engine Minimum Torque	0.0
Efficiency	1.0	Simple Engine Maximum Torque	1.0
Idle RPM	1000.0	Simple Engine Maximum Power	1.0
Rev Limit RPM	7000.0	Simple Engine Tau	0.0
<input type="checkbox"/> Rev Limit RPM Map	Plot/Edit Data		
Stall RPM	800.0		
Torque Scaling	1.0		
Torque Reaction Direction	No Torque Reaction	<input type="button" value="▼"/>	

The following parameters are peculiar to the simple engine:

- **Simple Engine Minimum Torque**

It defines the minimum torque value of the simple engine (T_{MIN}).

- **Simple Engine Maximum Torque**

It defines the maximum torque value of the simple engine (T_{MAX}).

- **Simple Engine Maximum Power**

It defines the maximum power value of the simple engine (P_{MAX}).

- **Simple Engine Tau**

Simple engine time constant, used to model a first order lag in simple engine model.

Standard Engine

The following parameters are peculiar to the standard engine:

- **Torque Map Data**

A 3D spline giving the torque produced by the engine and having as first independent variable (X) the engine RPM and as second independent variable (Z) the throttle demand (0-1).

$$T_{engine} = f(RPM_{engine}, Throttle)$$

- **Torque Map Property File**

when the related toggle button is activated, it is possible to select a *.pwr file in which the powertrain map is defined. Multiple maps pwr file is supported, meaning that it's possible to define a powertrain map for each gear. For all the details, please refer to [PWR File Format](#) section.

Note: a gear dependent pwr property file is shipped with carrealtime_shared database:
`mdids://carrealtime_shared/powertrains.tbl/VI_engine_map_gear_dep.pwr`

The following parameters refers both to simple and standard Engine:

- **Inertia**

inertia of the engine rotating parts.

- **Efficiency**

It defines the efficiency of the engine rotating parts.

- **Stall RPM**

It is the engine rpm speed at which the driver disengages the clutch.

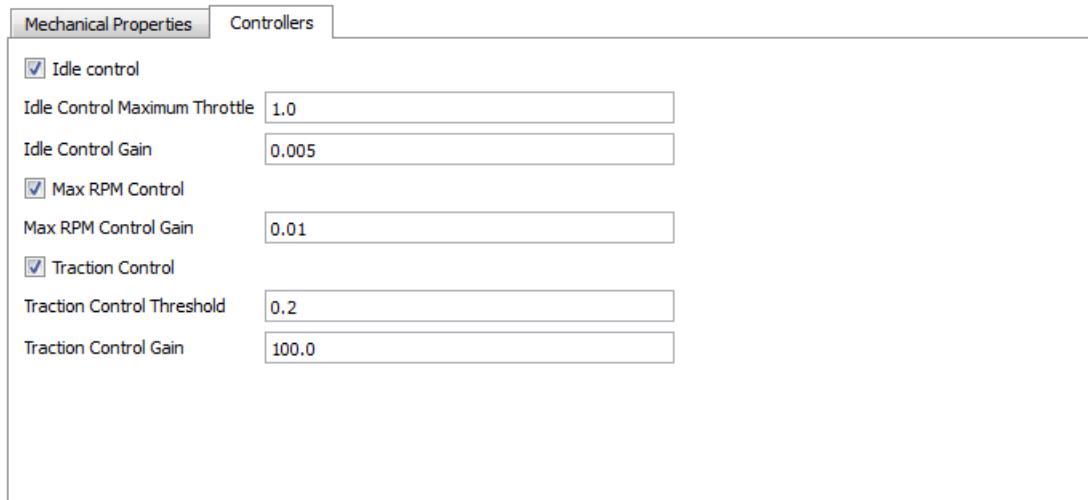
- **Idle RPM**

Idle rotational speed of the engine.

- **Rev Limit RPM**
Maximum rotational speed of the engine.
- **Rev Limit RPM Map**
It activates a map which allows to define an engine revolute limit for each gear.
- **Torque scaling**
Engine torque scaling factor.
- **Transmission Ratio**
it is the transmission ratio between the engine crankshaft and the clutch input plate.
- **Torque Reaction Direction**
It allows to set the direction of the reaction torque that acts on the chassis. Available choices are:
 - No Torque Reaction : motor reaction torque is null
 - X+: positive X direction in [vehicle reference frame](#)
 - X-: negative X direction in [vehicle reference frame](#)
 - Y+: positive Y direction in [vehicle reference frame](#)
 - Y-: negative Y direction in [vehicle reference frame](#)

Controllers

Controllers panel defines the following parameters, which refer both to simple and standard Engine:



- **Idle Control**
When the toggle button is checked, a controller works in order to keep the engine rotational speed at Idle Rpm, when engine rpm falls below such limit.
- **Idle Control Maximum Throttle**
It defines the maximum throttle demand the controller can use to keep the engine at Idle Rpm.
- **Idle Control Gain**
Proportional gain parameter which is used by the controller to keep the engine at Idle Rpm.
- **Max RPM Control**
When the toggle button is checked, a controller works in order to avoid the engine to overpass the Rev Limit Rpm.
- **Max RPM Control Gain**
Proportional gain parameter which is used by the controller to keep the engine below Rev Limit Rpm.
- **Traction Control**
Toggle button which activates the internal traction control system.
- **Traction Control Threshold**

It defines the longitudinal slip threshold at which the TCS starts to work.

- **Traction Control Gain**

It defines the proportional gain of the traction control system.

The Ram Air Tab in Powertrain Property Editor is used to set the following data:

- **Reference Air Density**

air density with respect to the engine map was measured. Unit: kg/mm³.

- **Ambient Pressure**

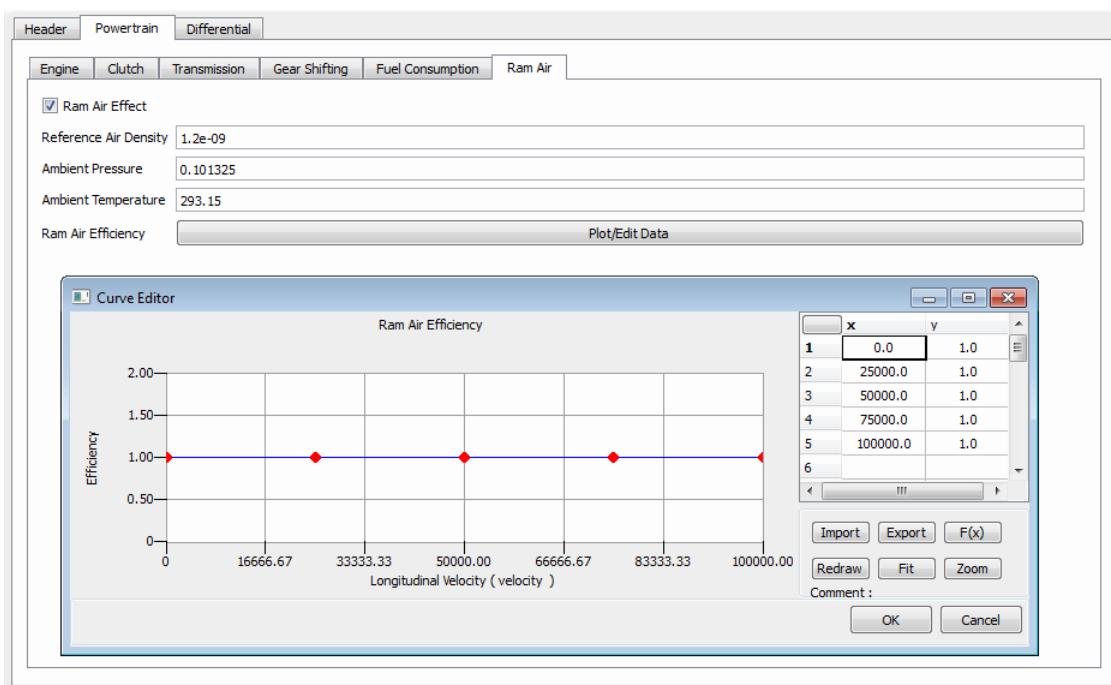
pressure unit: newton/mm².

- **Ambient Temperature**

temperature unit: Kelvin.

- **Ram Air Efficiency**

A 2D spline giving the efficiency (ϵ) vs. longitudinal velocity of the vehicle.



Previous data will be used to define a scaling coefficient for engine torque representing the ram air effect. This resultant scaling coefficient α is so computed:

$$\alpha = \frac{\rho_{\text{ram}}}{\rho_{\text{ref}}}$$

Where ρ_{ref} is the reference air density ρ_{ram} and is the air density in the ram obtained from the ideal gas law:

$$\rho_{\text{ram}} = \frac{P_{\text{ram}}}{R \cdot T_{\text{env}}}$$

Where the air pressure in the ram P_{ram} is so computed:

$$P_{\text{ram}} = P_{\text{env}} + 0.5 \cdot \rho_{\text{env}} \cdot \epsilon(\text{vel}) \cdot \text{vel}^2$$

With ρ_{env} the ambient density computed using again the ideal gas law:

$$\rho_{env} = \frac{P_{env}}{R \cdot T_{env}}$$

The Fuel Consumption Tab in Powertrain Property Editor is used to estimate the how much fuel is consumed for a certain analysis; both the instantaneous consumption and the overall consumption are computed and are available among [VI-CarRealTime output](#).

VI-CarRealTime allows the following computation method for the fuel consumption:

- [Brake Specific Fuel Consumption map](#);

Brake Specific Fuel Consumption map

The method computes the *Brake Specific Fuel Consumption* (BSFC) for each time step.

BSFC is a measure of the fuel efficiency of an engine. It represents the rate of fuel consumption divided by the power produced. It is the power-specific fuel consumption and for this reason this parameter allows the fuel efficiency of different reciprocating engines to be directly compared.

$$\text{BSFC} = r / P \quad [\text{g/kWh}]$$

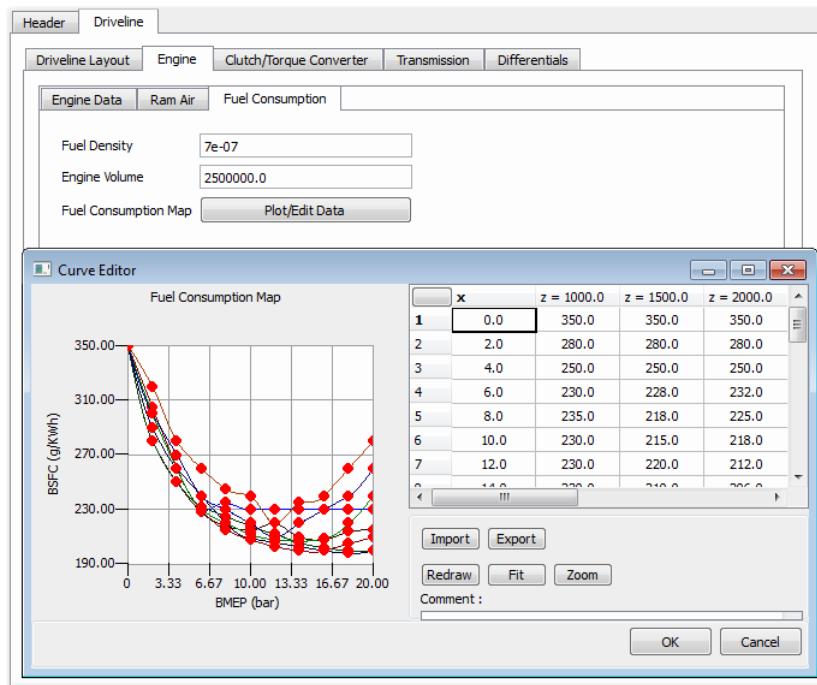
where:

- **r** is the fuel consumption rate in grams per second (g/s);
- **P** is the power produced in kiloWatts where $P = 0.001 \cdot T \cdot \omega$;
- **T** is the engine torque in newton meters (N·m);
- **ω** is the engine speed in radians per second (rad/s).

The resulting units of BSFC are grams per kiloWatt per hour (g/kWh).

Note: The units of the input variables for the BSFC map are fixed: *bar* for BMEP (Brake Mean Effective Pressure) and *round per minute* for engine RPM.

The following picture shows the panel for BSFC map computation together with an explanation of its fields:



- **Fuel Density**

Density of the fuel in current unit system [mass/length³].

- **Engine Volume**

Displacement of the engine in current unit system [length³].

- **Fuel Consumption Map**

A 3D spline giving the BSFC (g/kWh) and having as first independent variable (X) the BMEP (Brake Mean Effective Pressure) and as second independent variable (Z) the engine RPM: $BSFC = f(BMEP, RPM)$.

During the simulation, the BMEP value used to enter the Fuel Consumption Map is computed as:

$$BMEP = \frac{T_{eng} \cdot 2\pi \cdot 2}{V \cdot 10^5}$$

where:

- T_{eng} is the engine torque.
- V is the engine displacement.
- 10^5 is the unit conversion factor from meter (*send_svm.xml file is always in MKS) to bar.
- 2 is the number of revolution per power stroke of a 4-stroke engine.

Additional Motors, other than the main [Engine](#), can be activated:

- **Main**

motor located right before the gearbox or right before the clutch, depending on [Torque Output To](#) option menu choice.

- **Central**

motor located right before the central differential.

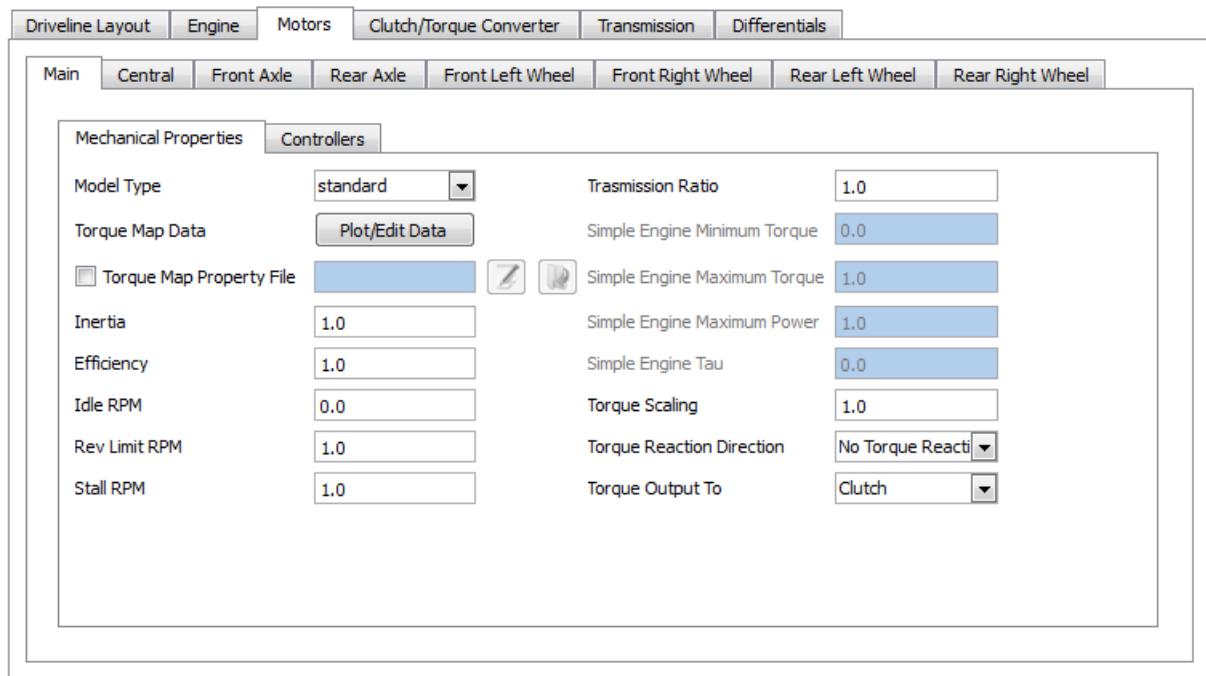
- **Front Axle**

motor located right before the front differential.

- **Rear Axle**

motor located right before the rear differential.

- **Front Left Wheel**
motor rigidly connected to the front left wheel.
- **Front Right Wheel**
motor rigidly connected to the front right wheel.
- **Rear Left Wheel**
motor rigidly connected to the rear left wheel.
- **Rear Right Wheel**
motor rigidly connected to the rear right wheel.



For each motor three different Model Types are available:

- [Simple Engine](#)
- [Standard Engine](#)
- [Electric Engine](#)

Simple Engine

The parameters are the same as per the main Engine, so please refer to [Simple Engine Data](#) topic for the details.

Standard Engine

The parameters are the same as per the main Engine, so please refer to [Standard Engine Data](#) topic for the details.

Electric Engine

The following parameters are peculiar to the electric engine:

- **Motor Torque Map Data**

A 3D spline giving the *positive* torque produced by the engine and having as first independent variable (X) the engine RPM and as second independent variable (Z) the throttle demand (0-1). It is used when Torque Map Property File toggle button is unchecked.

$$T_{\text{engine}} = f(\text{RPM}_{\text{engine}}, \text{Throttle})$$

- **Torque Map Property File**

When the related toggle button is activated, it is possible to select a *.pwr file in which Motor Torque Map, Braking Torque Map and Coasting Torque Map are defined. For all the details, please refer to [PWR File Format](#) section.

Note: an electric motor pwr property file is shipped with carrealtime_shared database: `mdids://carrealtime_shared/powertrains.tbl/VI_engine_electric_map.pwr`

- **Braking Torque Map Data**

A 3D spline giving the torque produced by the engine and having as first independent variable (X) the engine RPM and as second independent variable (Z) the brake demand (0-1). It's used only during the regenerative phase in conjunction with the regenerative braking factor.

$$T_{\text{engine}} = f(\text{RPM}_{\text{engine}}, \text{Brake})$$

- **Coasting Torque Map Data**

A 2D spline giving the *negative* torque produced by the engine, mapped as a function of the engine RPM. This torque is applied only in coasting condition, so with no throttle and no brake applied by the driver.

- **Regenerative Braking Factor**

It's the percentage of the Braking Torque Map that it's recovered when the driver is braking. During the braking phase (driver brake demand bigger than 0) the Braking Torque Map spline is interpolated using the RPM as first independent variable (X) and the *brake demand* as second independent variable (Z); the torque obtained is multiplied by the regenerative braking factor and its sign is inverted (whenever positive). Such torque is the torque absorbed by the electric engine.

- **Regenerative Braking Map**

When the flag is checked, the Regenerative Braking Factor slider is inactive, and the braking factor must be mapped as a function of engine RPM and engine torque.

- **Longitudinal Velocity Threshold**

Below this speed (in [mm/s]) the electric motor will not deliver any negative torque, so coasting torque and braking torque map will be ignored.

- **Efficiency**

It defines the efficiency of the engine rotating parts.

- **Efficiency Map**

When the toggle button is checked, the Efficiency field is disabled and the efficiency must be mapped as a function of engine RPM (first independent variable) and engine torque (second independent variable).

The other parameters are common to both Standard and Simple Engine. Please refer to [Engine Data](#) topic for the details.

Note: an example of electric engine subsystem (*SedanCar_powertrain_MainElectric.xml*) with dissipative torque and regenerative factor is shipped with SedanCar.cdb.

The following parameter is used only by Main motor is:

- **Torque Output To**

It allows the user to choose where the Main motor torque acts. Available choices are:

- **Clutch:** Main motor output torque is transmitted right before the clutch
- **Gearbox:** Main motor output torque is transmitted right before the gearbox.

- **Stall RPM**

It is the engine rpm speed at which the driver disengages the clutch.

NOTE: Any motors you want to interact with the battery model must be set to "electric" Model Type.

The Battery panel defines the parameters of the battery.

Type	Lead-Acid
Voltage [V]	<input checked="" type="radio"/> Constant 400.0 <input type="radio"/> Function of SOC Plot/Edit Data
Polarization Constant [V / Ah]	0.0014563
Maximum Battery Capacity [Ah]	72.0
Response Time	10.0
Resistance [Ohm]	0.0125
State of Charge [%]	25.0
Exponential Voltage [V]	3.9883
Exponential Capacity [1/Ah]	1.6667
Max Peak Current [A]	1500.0

The following parameters must be set to characterize the standard clutch model:

- **Type**

Select battery type.

- **Voltage [V]**

Battery voltage.

- **PolarizationConstant [V / Ah]**
Battery polarization constant.
- **Maximum Battery Capacity [Ah]**
Battery maximum capacity.
- **Response Time**
Battery response time.
- **Resistance [Ohm]**
Battery resistance.
- **State of Charge [%]**
Battery initial state of charge.
- **Exponential Voltage [V]**
Battery exponential voltage.
- **Exponential Capacity [1/Ah]**
Battery exponential capacity.
- **Max Peak Current [A]**
Battery maximum peak current.

The following battery types are available:

- **Lead-Acid**

$$\text{Discharge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{Q-it} \cdot (it + i^*) + \text{Exp}(t)$$

$$\text{Charge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{it-0.1 \cdot Q} \cdot i^* - K \frac{Q}{Q-it} \cdot it + \text{Exp}(t)$$

- **Li-ion**

$$\text{Discharge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{Q-it} \cdot (it + i^*) + A \exp(-B \cdot it)$$

$$\text{Charge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{it-0.1 \cdot Q} \cdot i^* - K \frac{Q}{Q-it} \cdot it + A \exp(-B \cdot it)$$

- **Ni-Cd**

$$\text{Discharge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{Q-it} \cdot (it + i^*) + \text{Exp}(t)$$

$$\text{Charge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{|it|-0.1 \cdot Q} \cdot i^* - K \frac{Q}{Q-it} \cdot it + \text{Exp}(t)$$

- **Ni-Mh**

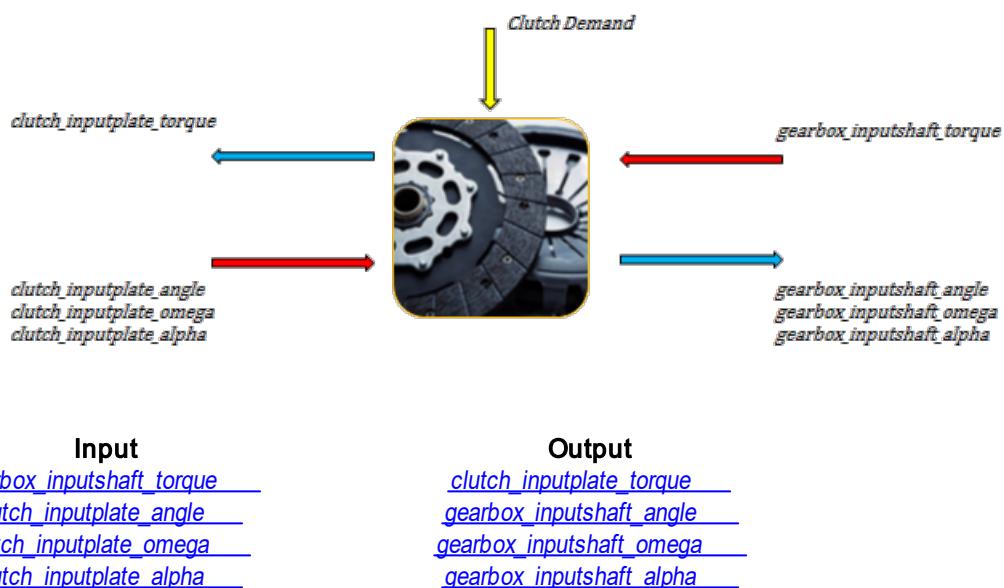
$$\text{Discharge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{Q-it} \cdot (it + i^*) + Exp(t)$$

$$\text{Charge : } V_{batt} = E_0 - R \cdot i - K \frac{Q}{|it|-0.1 \cdot Q} \cdot i^* - K \frac{Q}{Q-it} \cdot it + Exp(t)$$

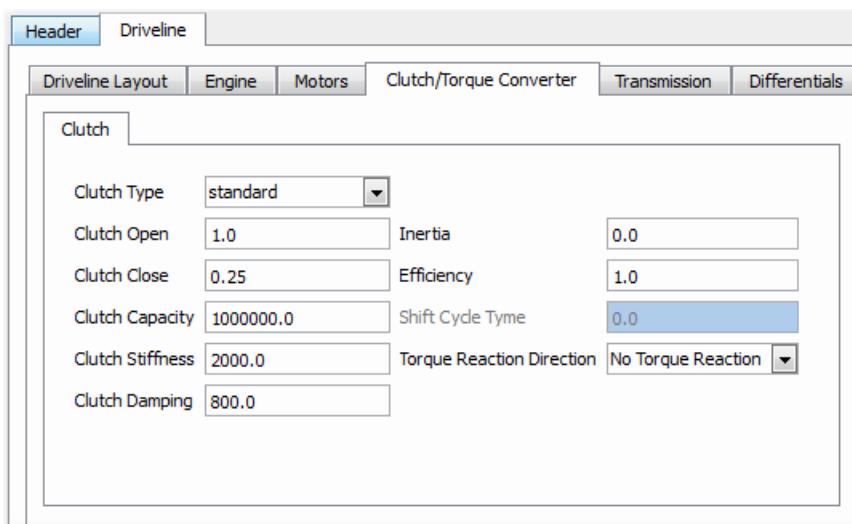
The following models are available to transmit the engine output shaft torque to the transmission:

- [Clutch](#)
- [Torque Converter](#)

Here the Clutch/Torque Converter input/output interface:



The Clutch Tab is used to set data for the internal clutch model.



The following clutch types are available:

- [Standard](#)
- [Dual](#)

Standard

The following parameters must be set to characterize the standard clutch model (please refer to [common](#) paragraph for the other parameters that are shared with Dual Clutch model):

- **Clutch Open**
clutch demand value at which the clutch is totally open (≤ 1.0).
- **Clutch Close**
clutch demand value at which the clutch is totally closed (≥ 0.0).
- **Clutch Capacity**
maximum torque the clutch is able to transmit.
- **Inertia**
inertia of the output plate of the clutch model (I).

The clutch torque expression is the following:

$$T = \frac{(k \cdot \theta + c \cdot \dot{\theta})}{\varepsilon} \cdot \text{step}(C_{\text{demand}}, C_{\text{open}}, 0, C_{\text{closed}}, 1)$$

where:

- θ is the clutch rotation
- C_{demand} is the driver clutch demand
- C_{open} is the clutch value at which the clutch is fully disengaged
- C_{closed} is the clutch value at which the clutch is fully engaged

The absolute value of the clutch torque is saturated by Clutch Capacity parameter.

Clutch slip is defined by the following differential equation:

$$\dot{\theta} = (\omega_{\text{eng}} - \omega_{\text{cl_out}}) \cdot \text{step}(C_{\text{demand}}, C_{\text{open}}, 0, C_{\text{closed}}, 1)$$

where:

- ω_{eng} is the engine rotational speed
- $\omega_{\text{cl_out}}$ is the clutch output plate rotational speed

Dual

The dual clutch model allows to smoothly shift gears. The gear shift is seamless and the time it takes to pass from one gear to another one can be set by the user.

The following parameters must be set to characterize the dual clutch model (please refer to [common](#) paragraph for the other parameters that are shared with Standard model):

- **Shift Cycle Time**
It is the time needed by the dual clutch model to smoothly shift from the actual gear ratio to the next one.

Common

The following parameters are *common* to both standard and dual clutch model:

- **Clutch Stiffness**
stiffness of the clutch model (k).

- **Clutch Damping**
damping of the clutch model (c).

- **Efficiency**

It is the clutch model efficiency (ε), defined as the ratio between clutch output plate torque and clutch input plate torque.

- **Inertia**

Inertia of the output plate of the clutch model (I).

- **Torque Reaction Direction**

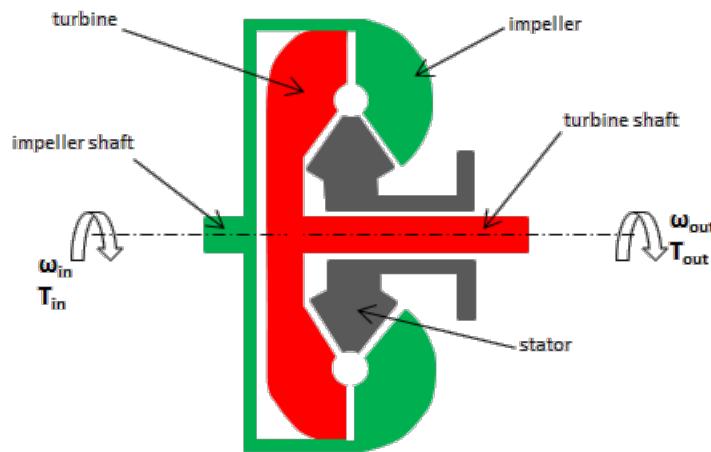
It allows to set the direction of the reaction torque that acts on the chassis. Available choices are:

- No Torque Reaction : clutch reaction torque is null
- X+: positive X direction in [vehicle reference frame](#)
- X-: negative X direction in [vehicle reference frame](#)
- Y+: positive Y direction in [vehicle reference frame](#)
- Y-: negative Y direction in [vehicle reference frame](#)

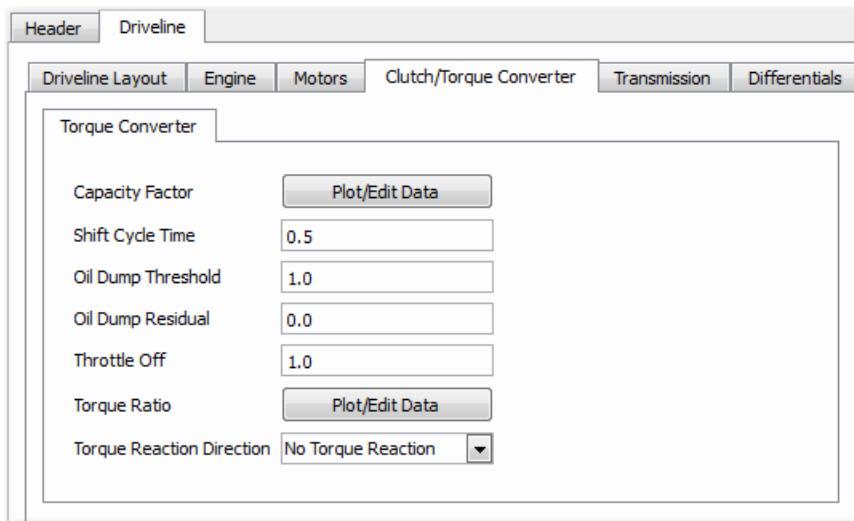
The Torque Converter is a fluid coupling composed by the following parts:

- Impeller
- Stator
- Turbine

The impeller (input shaft) is connected to the engine while the turbine (output shaft) is connected to the transmission.



The Torque Converter Tab is used to set the following data:



- **Capacity Factor**

It is a 2D Spline which characterizes the capacity factor as a function of speed ratio:

- **capacity factor (k)** is the torque converter input speed divided by the square root of the input torque
- **speed ratio (τ_ω)** is defined as output shaft omega divided by input shaft omega

- **Shift Cycle Time**

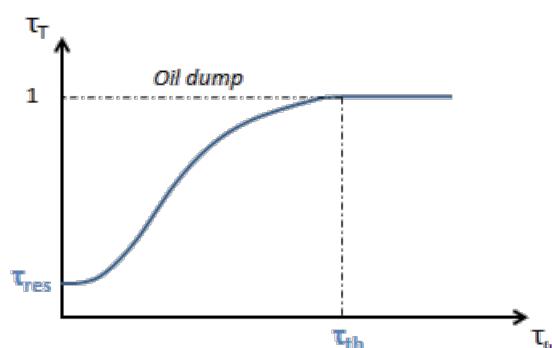
It's the time the torque converter takes to shift from one gear to the next one.

- **Oil Dump Threshold**

It is the speed ratio at which the torque attenuation begins. (τ_{th})

- **Oil Dump Residual**

It is the residual fraction of torque produced at zero speed ratio. (τ_{res})



- **Throttle Off**

It is the percentage of throttle at which the throttle is considered off, such allowing oil dump (th_{off}).

- **Torque Ratio**

It is a 2D Spline which characterizes the torque ratio as a function of speed ratio:

- **torque ratio (τ_T)** is the output shaft (turbine) torque divided by input shaft (impeller) torque.
- **speed ratio (τ_ω)** is defined as output shaft (turbine) omega divided by input shaft (impeller) omega

- **Torque Reaction Direction**

It allows to set the direction of the reaction torque that acts on the chassis. Available choices are:

- **No Torque Reaction** : torque converter reaction torque is null
- **X+:** positive X direction in [vehicle reference frame](#)
- **X-:** negative X direction in [vehicle reference frame](#)

- Y+: positive Y direction in [vehicle reference frame](#)
- Y-: negative Y direction in [vehicle reference frame](#)

Torque Converter output torque (T_{out}) is computed as:

$$T_{out} = \left(\frac{\omega_{in} \cdot \gamma}{k} \right)^2 \cdot \tau_T$$

where γ is defined as:

$$\gamma = \begin{cases} 1, & \Delta > 0 \\ step(\tau_\omega, 0, \tau_{res}, \tau_{th}, 1), & \Delta \leq 0 \end{cases}$$

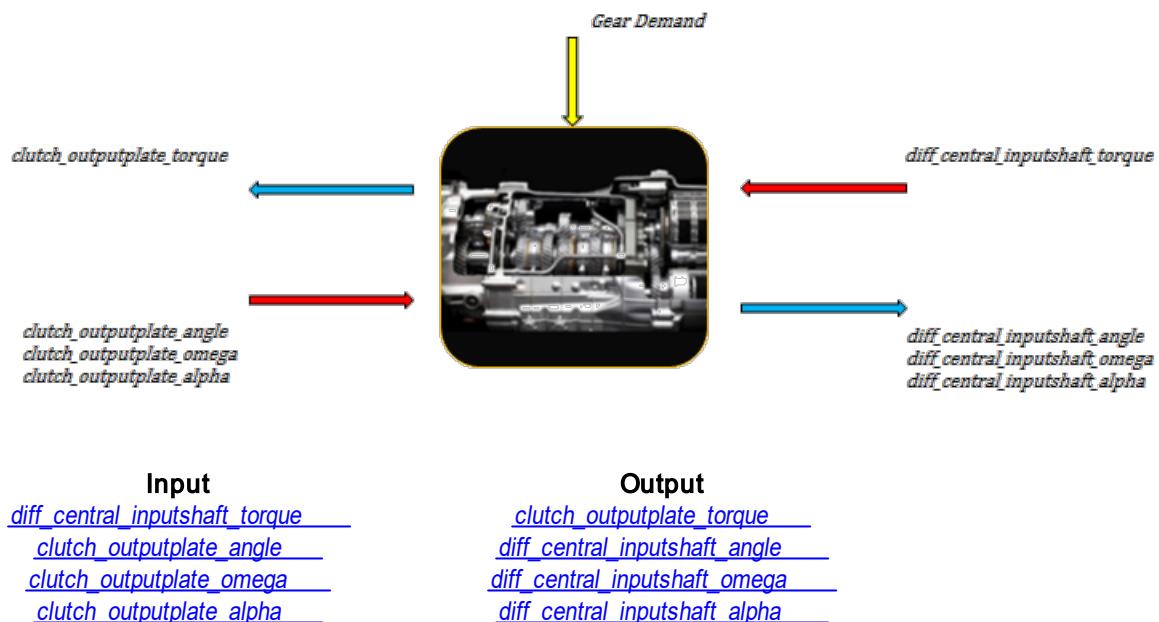
with:

$$\Delta = \%th - th_{off}$$

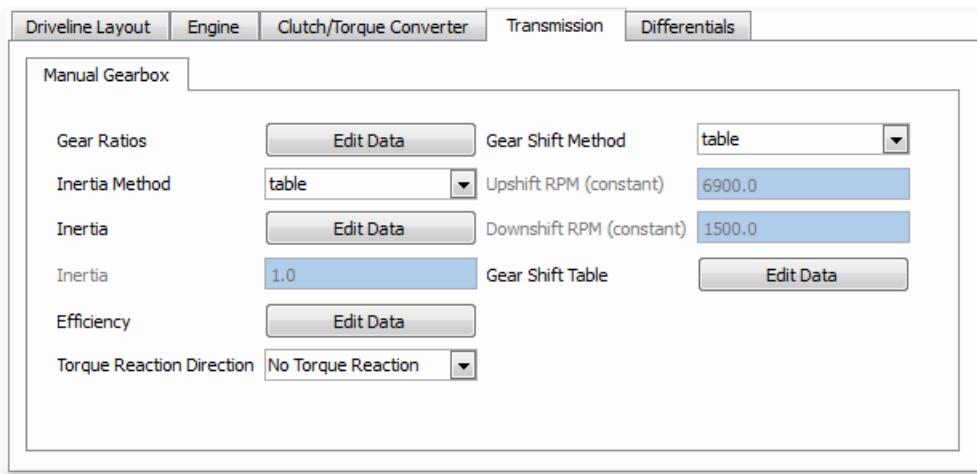
Transmission panel defines the following tabs:

- [Manual](#)
- [Automatic](#)

Here the transmission input/output interface:



Manual Tab is used to set the following data:



- **Gear Ratios**

It is a 2D spline describing gear ratios, and having as independent value the gear number.

- **Inertia Method**

It allows to select how to compute the inertia of the output shaft of the gearbox. Available choices are: *constant* and *table*.

- **Inertia**

It is the inertia of the output shaft of the gearbox. The field is active when Inertia Method option menu is set to *constant*.

- **Inertia (table)**

It allows the user to specify an output shaft inertia value for each gear. The table is active when Inertia Method option menu is set to *table*.

- **Efficiency**

It is a 2D spline describing the efficiency of the manual gearbox transmission as a function of gear number.

- **Gear Shift Method**

Option menu which is used to define the gear shifting strategy in simulations with a driver. The following methods are available to define the engine RPM defining gear upshift or downshift:

- **Constant**
- **Table**

- **Upshift RPM (constant)**

It defines the upshift RPM for each gear. Such field is enabled only when Gear Shift Method is set to Constant.

- **Downshift RPM (constant)**

It defines the downshift RPM for each gear. Such field is enabled only when Gear Shift Method is set to Constant.

- **Gear Shift Table**

3D spline describing gear shift RPM as a function of gear number (X) and gear shift "direction" (Z=-1 for downshift and Z=1 for upshift).

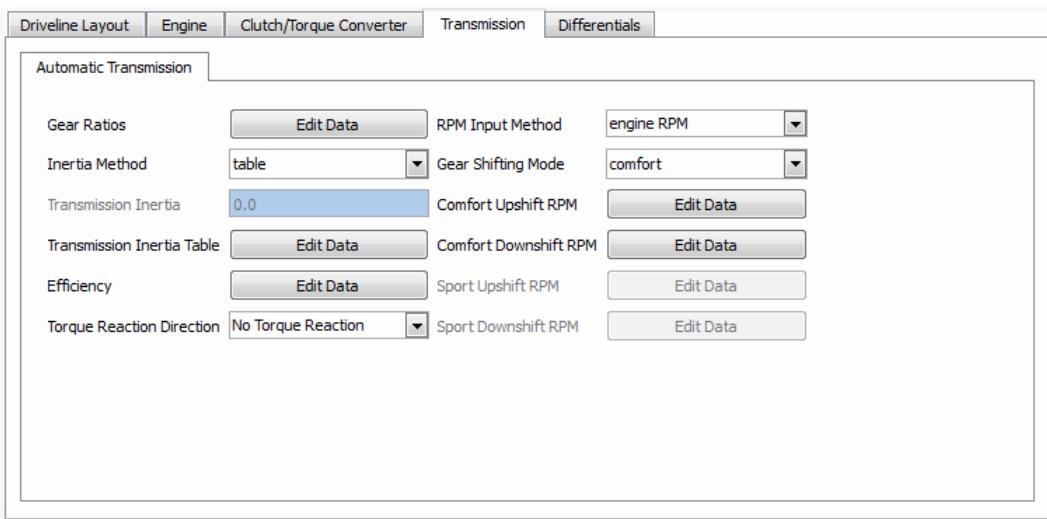
- **Torque Reaction Direction**

It allows to set the direction of the reaction torque that acts on the chassis. Available choices are:

- *No Torque Reaction* : transmission reaction torque is null
- *X+*: positive X direction in [vehicle reference frame](#)
- *X-*: negative X direction in [vehicle reference frame](#)
- *Y+*: positive Y direction in [vehicle reference frame](#)

- Y-: negative Y direction in [vehicle reference frame](#)

Automatic Tab is used to set the following data:



• Gear Ratios

It is a 2D spline describing automatic gearbox gear ratios, and having as independent value the gear number.

• Inertia Method

The option menu allows to select how to deal with the automatic transmission inertia. Available choices are: *constant* and *table*.

• Transmission Inertia

It is the inertia of the output shaft of the automatic gearbox. The field is active when Inertia Method option menu is set to *constant*.

• Transmission Inertia Table

It allows the user to specify a transmission inertia value for each gear. The table is active when Inertia Method option menu is set to *table*.

• Efficiency

It is a 2D spline describing the efficiency of the automatic gearbox transmission as a function of gear number.

• RPM Input Method

Option Menu used to identify the first independent variable of both Upshift and Downshift RPM 3D splines. Available choices are:

- engine RPM
- transmission input RPM
- transmission output RPM

• Gear Shifting Mode

The option menu allows to choose between two different maps of upshift and downshift rpm: **comfort** and **sport**.

Note: It is possible to modify runtime the gear shifting mode when running co-simulation by using [automatic_gearbox_shifting_mode](#) input value.

• Comfort Upshift RPM

It is a 3D spline defining the upshift RPM as a function of selected RPM Input Method and driver throttle demand. It is active when Gear Shifting Mode is set to *comfort*.

• Comfort Downshift RPM

It is a 3D spline defining the downshift RPM as a function of selected RPM Input Method and driver throttle demand. It is active when Gear Shifting Mode is set to *comfort*.

- **Sport Upshift RPM**

It is a 3D spline defining the upshift RPM as a function of selected RPM Input Method and driver throttle demand. It is active when Gear Shifting Mode is set to *sport*.

- **Sport Downshift RPM**

It is a 3D spline defining the downshift RPM as a function of selected RPM Input Method and driver throttle demand. It is active when Gear Shifting Mode is set to *sport*.

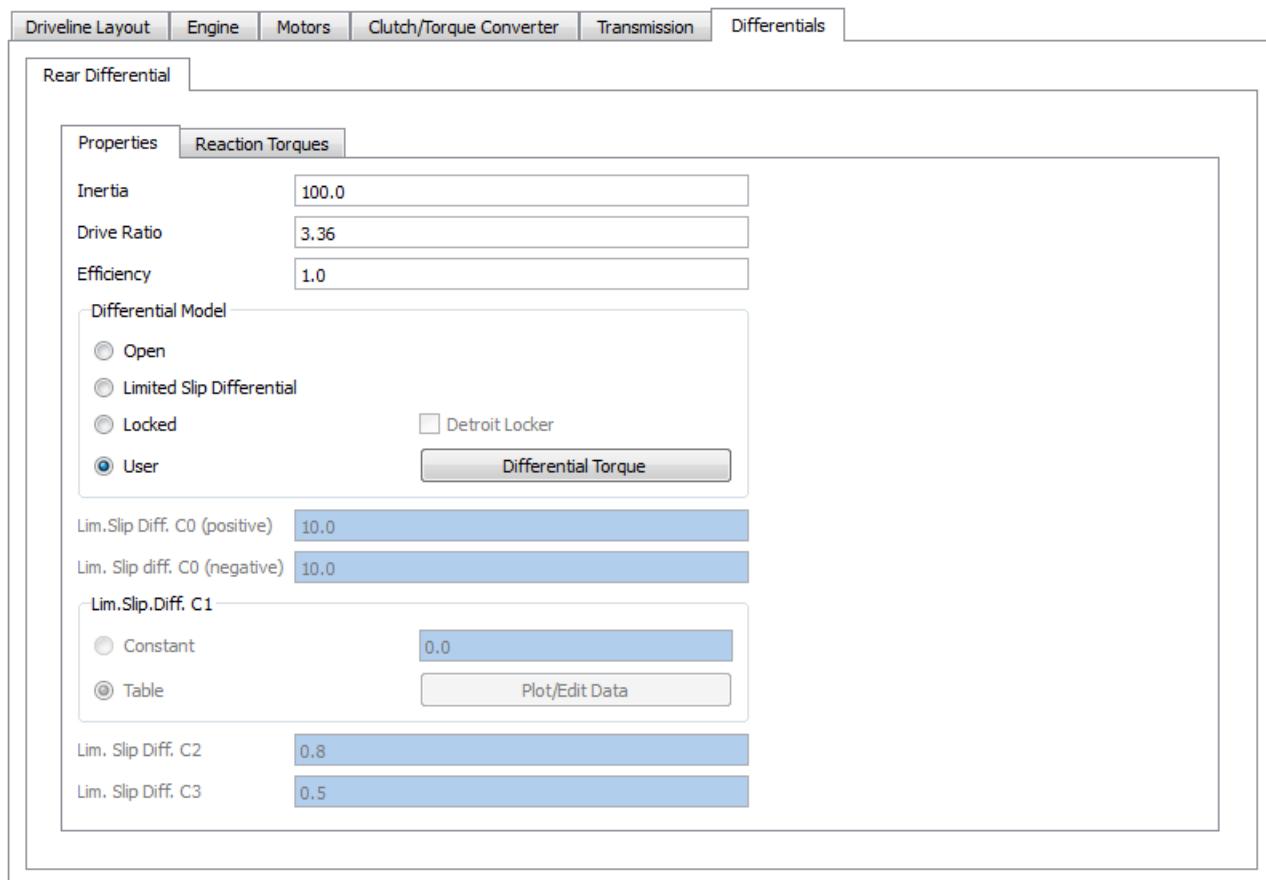
- **Torque Reaction Direction**

It allows to set the direction of the reaction torque that acts on the chassis. Available choices are:

- No Torque Reaction : transmission reaction torque is null
- X+: positive X direction in [vehicle reference frame](#)
- X-: negative X direction in [vehicle reference frame](#)
- Y+: positive Y direction in [vehicle reference frame](#)
- Y-: negative Y direction in [vehicle reference frame](#)

The Differentials panel defines the following tabs:

- [Front Differential](#)
- [Rear Differential](#)
- [Central Differential](#)



For each differential the following parameters can be set:

- [Properties](#)
- [Reaction Torques](#)

Properties

Properties panel allows to set the following data:

- **Inertia**
inertia of differential rotating parts.
- **Drive Ratio**
differential drive ratio.
- **Efficiency**
Efficiency of the differential. 0-1, where 1 means 100%.

Differential Model

- **Open**
output torques are the same.
- **Locked**
output angular velocities are the same. The condition of same output angular velocities for locked differential is obtained by a PID controller. The gains of the controller can be set in system parameters, under differential tree.

Subsystem Definition			Properties	Output channels
ABS				
TCS				
system_parameters				
trailer				
aerodynamics				
powertrain				
circular_buffer				
statics				
body_compliance				
setup				
differential				
anti_roll_bar				
auxiliary_vertical				
std_tire_ref				
dampers				
update_ride_height_maps				
g				
output_files				
simulink				
external_suspension				

Name	Value	Comment
locked_differential_kp	10000.0	Proportional Gain for locked differential controller
locked_differential_kd	100.0	Derivative Gain for locked differential controller
locked_differential_ki	10.0	Integral Gain for locked differential controller
user_differential_lp_filter	0.0	Activity flag for user differential low pass filter - cutoff freq 100Hz
LSD_omega_threshold	0.5	LSD angular velocity threshold [rad/s]

- **LSD**

The Limited Slip Differential is implemented as follows:

if ($|\delta T| < |c_0|$)

$$\delta T = T_{lock}$$

else

$$\delta T = \begin{cases} c_0 + \delta\omega_{STEP}(c_2|\delta\omega|^{c_3}) & , & c_1T - |c_0| \leq 0 \\ c_0 + \delta\omega_{STEP}(|c_1T| - |c_0| + c_2|\delta\omega|^{c_3}), & & c_1T - |c_0| > 0 \end{cases}$$

where:

T is the crown wheel torque

T_{lock} is the torque required to have $\delta\omega=0$

$\delta\omega$ is the wheel speed difference left – right

$\delta\omega_{STEP} = Step(\delta\omega, -\omega_{Threshold}, \omega_{Threshold}, -1, 1)$

When the wheel torque difference is lower than c_0 the differential behaves as locked (imposing $\delta\omega = 0$) having parameter c_0 the meaning of a differential preload. The preload can be set differently for traction (*LSC* c_0 positive) or braking (*LSD* c_0 negative).

As soon as the wheel torque difference becomes bigger than c_0 the following speed difference logic is used to 'point' T:

$$\begin{aligned}\delta\omega > 0 \quad T_{left} &= 1/2(T - \delta T) \\ T_{right} &= 1/2(T + \delta T)\end{aligned}$$

$$\begin{aligned}\delta\omega < 0 \quad T_{left} &= 1/2(T + \delta T) \\ T_{right} &= 1/2(T - \delta T)\end{aligned}$$

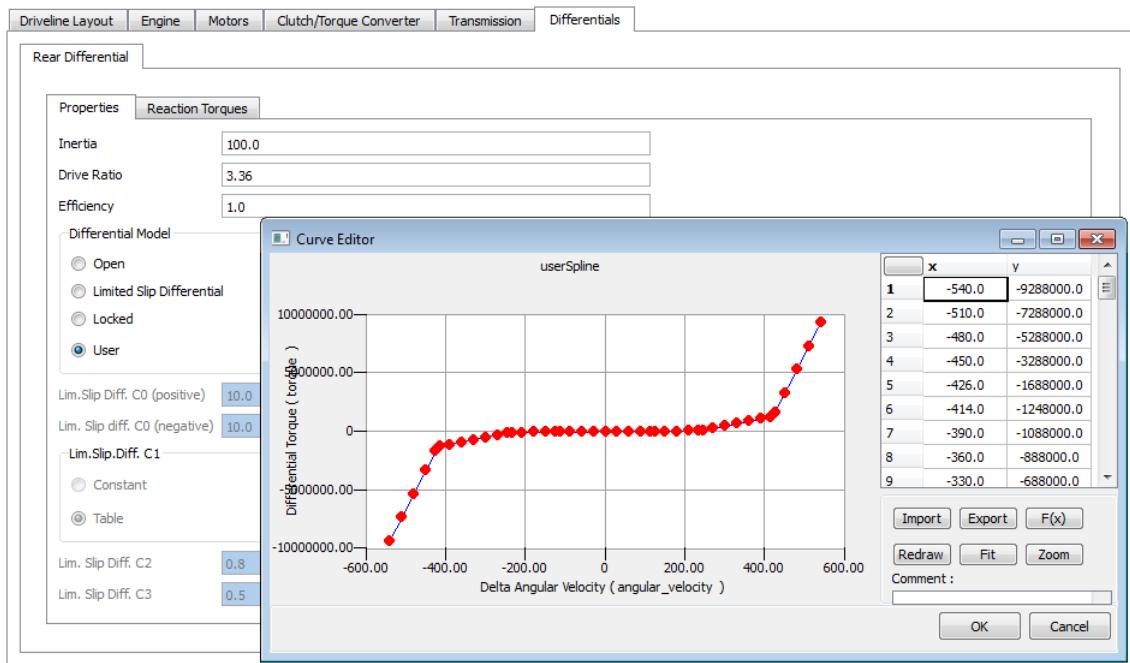
$$\begin{aligned}\delta\omega = 0 \quad T_{left} &= 1/2T \\ T_{right} &= 1/2T\end{aligned}$$

By choosing appropriate values for coefficients $c_0 \dots c_3$ the user can then model the torque split as desired. The implementation includes the possibility of having either a spline or a constant for C_1 , depending on differential input torque and constants C_0 , C_2 and C_3 . For C_0 there are two different constants, one for positive, the other for negative input torque.

For solver stability the System parameter tree also includes a tunable parameter for omega threshold used to invert the sign of output torque with continuity (by step function) when near zero crossing.

• User

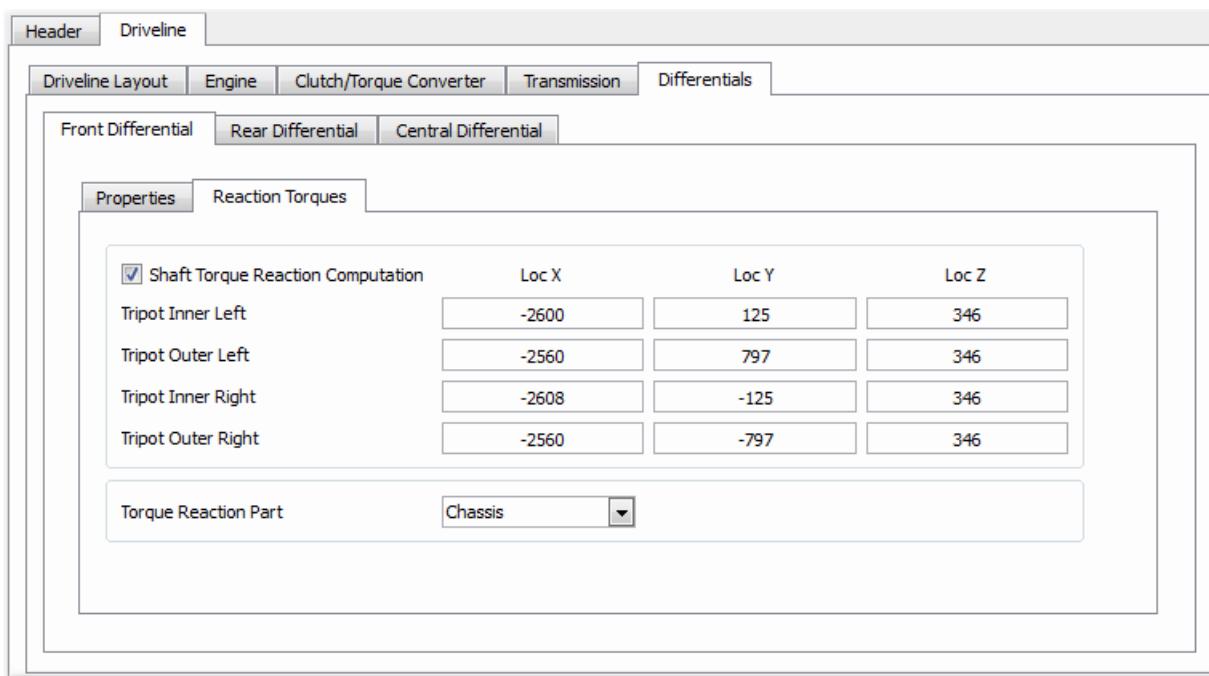
The user differential model is modeled through a lookup table. The spline will represent the differential torque w.r.t the delta angular wheel velocity.



Reaction Torques

When the driveshaft is not aligned with spindle axis, tripot joints generate reaction torque that should be applied to suspension part for the outer and to chassis part for the inner.

Reaction forces and torques on suspension part will add contributes to jacking force and to kingpin moment.



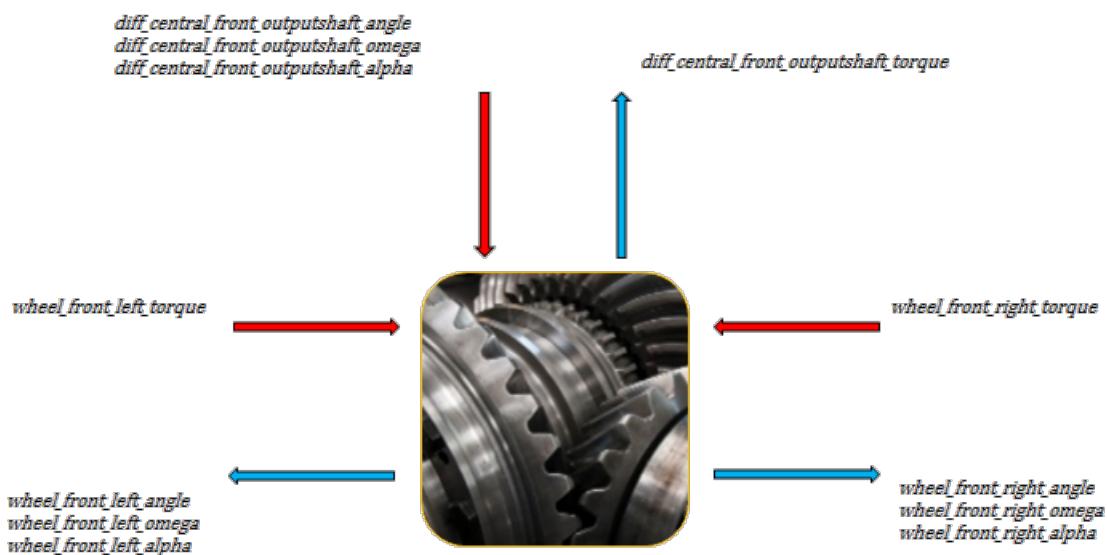
When **Shaft Torque Reaction Computation** toggle button is checked, such additional contributes will be computed and the user must enter the following coordinates:

- **Tripot Inner Left**
location of inner left tripot in [Vehicle Reference Frame](#).
- **Tripot Outer Left**
location of outer left tripot in [Vehicle Reference Frame](#).
- **Tripot Inner Right**
location of inner right tripot in [Vehicle Reference Frame](#).
- **Tripot Outer Right**
location of outer right tripot in [Vehicle Reference Frame](#).

VI-CarRealTime solver computes the differential reaction torque. It is the torque that puts the differential model in equilibrium depending on the another torques acting on it. The user is allowed to choose which is the part where the differential reaction torque acts:

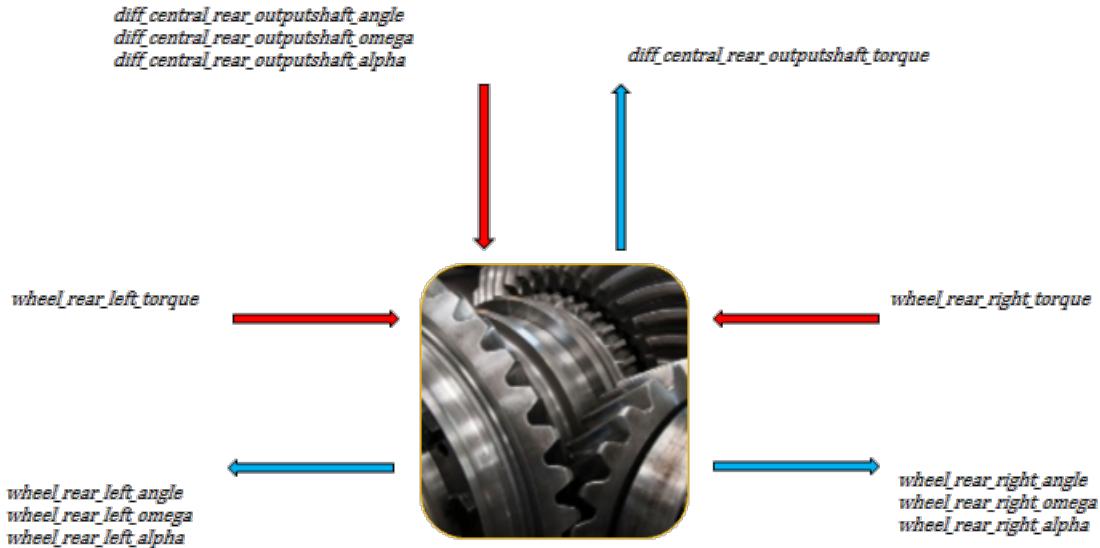
- **Torque Reaction Part**
Option menu which allows to select the part where the differential reaction torque acts, among:
 - No Reaction Part
 - Chassis
 - Engine
 - Suspension

Here the Front Differential input/output interface:



Input	Output
<u>wheel_front_right_torque</u>	<u>diff_central_front_outputshaft_torque</u>
<u>wheel_front_left_torque</u>	<u>wheel_front_left_angle</u>
<u>diff_central_front_outputshaft_angle</u>	<u>wheel_front_left_omega</u>
<u>diff_central_front_outputshaft_omega</u>	<u>wheel_front_left_alpha</u>
<u>diff_central_front_outputshaft_alpha</u>	<u>wheel_front_right_angle</u>
	<u>wheel_front_right_omega</u>
	<u>wheel_front_right_alpha</u>

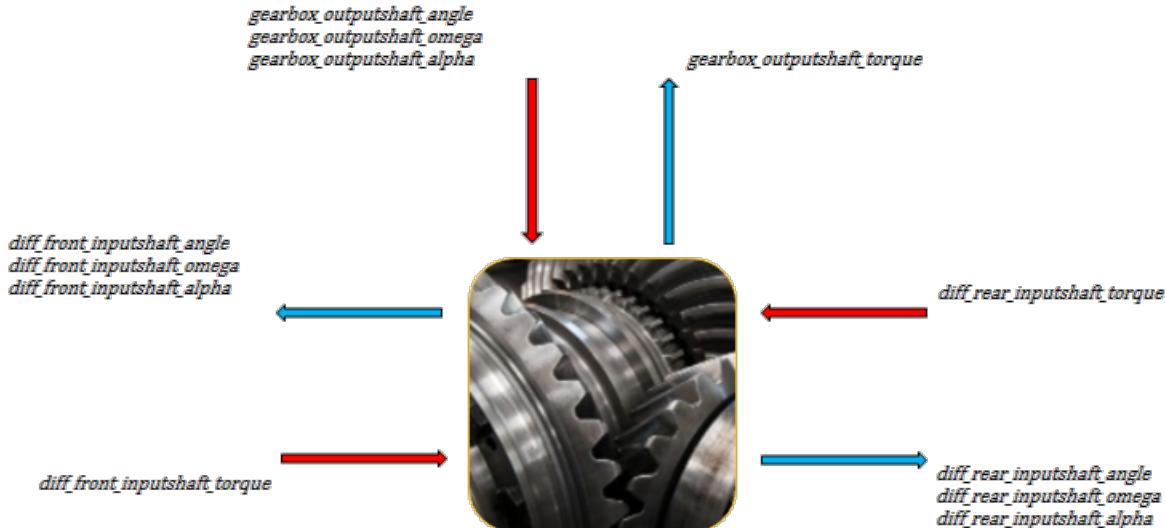
Here the Rear Differential input/output interface:



Input	Output
<u>wheel_rear_right_torque</u>	<u>diff_central_rear_outputshaft_torque</u>
<u>wheel_rear_left_torque</u>	<u>wheel_rear_left_angle</u>
<u>diff_central_rear_outputshaft_angle</u>	<u>wheel_rear_left_omega</u>
<u>diff_central_rear_outputshaft_omega</u>	<u>wheel_rear_left_alpha</u>

[diff_central_rear_outputshaft_alpha](#)[wheel_rear_right_angle](#)[wheel_rear_right_omega](#)[wheel_rear_right_alpha](#)

Here the Central Differential input/output interface:



Input

- [diff_front_inputshaft_torque](#)
- [diff_rear_inputshaft_torque](#)
- [gearbox_outputshaft_angle](#)
- [gearbox_outputshaft_omega](#)
- [gearbox_outputshaft_alpha](#)

Output

- [gearbox_outputshaft_torque](#)
- [diff_front_inputshaft_angle](#)
- [diff_front_inputshaft_omega](#)
- [diff_front_inputshaft_alpha](#)
- [diff_rear_inputshaft_angle](#)
- [diff_rear_inputshaft_omega](#)
- [diff_rear_inputshaft_alpha](#)

Engine Part

VI-CarRealTime supports the possibility of including in the vehicle model the engine part, which can be separated from the chassis sprung mass and connected to it through bushing elements.

The feature is optional and is activated for the models generated through the Adams Car plugin when the powertrain template includes a group named [powertrain_parts](#) containing the list of the parts which define the vehicle powertrain; engine torque will be then applied from engine part to wheels.

Moreover, it's possible to define additional rigid parts (VI-CarRealTime solver can manage up to 2 parts beside the main engine part). Such feature can be automatically managed by the Adams Car plugin when the powertrain template includes a group named [powertrain_rigid](#) containing the list of the parts which define the stabilizer bars of the main engine block.

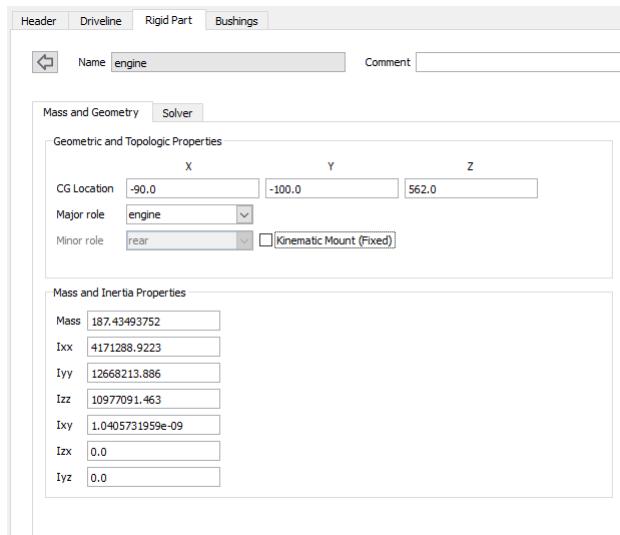
Powertrain Rigid Part panel embeds the following sub-panels:

- [Mass and Geometry](#)
- [Solver](#)

Note: an example of powertrain subsystem (*RT_VI_CRT_Demo_powertrain_w_engine.xml*) including the engine part is shipped with *carrealtime_shared* database.

Note: an example of powertrain subsystem (*RT_Crossover_powertrain.xml*) including both the engine part and a rod part is shipped with Crossover database.

Powertrain part data are input through the widget described in the snapshot below and features the following parameters:



Geometric and Topological Properties

- **CG Location**

X,Y,Z Engine Part CG coordinates expressed in [Global Reference](#) frame.

- **Major Role**

Attribute which identifies the rigid part. Available choices are:

- engine
- rod

The attribute is referenced in [bushing](#) I/J Body field.

- **Minor Role**

Attribute which specifies whether the rigid part is a front or a rear part. The option menu is active only when Kinematic Mount (Fixed) toggle is checked; otherwise, the part connection with other parts is defined in the Bushings panel.

Note: the attribute is always inactive for rod parts, since they are supposed to behave as the main engine part.

- **Kinematic Mount (Fixed)**

If deactivated the engine part will be considered mounted to the chassis/rod with compliance attachments (using [bushing](#)); if activated the engine part will be mounted to the chassis with kinematic constraint (fixed joint).

Note: the attribute is always inactive for rod parts, since they are supposed to behave as the main engine part.

Mass and Inertia Properties

Mass

Mass of engine part.

I_{xx}

I_{xx} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yy}

I_{yy} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{zz}

I_{zz} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

lxy

I_{xy} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

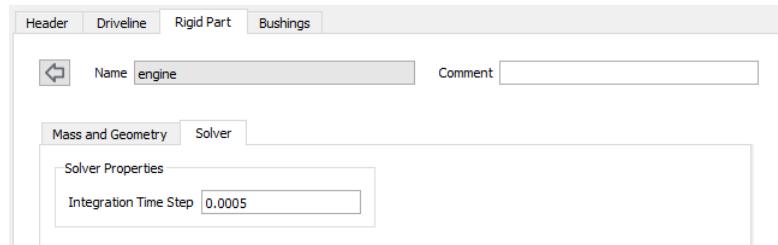
lxz

I_{xz} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

lyz

I_{yz} moment of inertia of engine part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

Powertrain part is integrated with its own Integration Time Step, independently with respect to the [Integration Time Step](#) set in the [Test Mode](#).



Bushings

For models including an engine part the connectivity to the chassis is realized either through kinematic mount (rigid joint) or through bushing elements.

When at least one bushing instance exists in the model, additional bushings can be created in VI-CarRealTime by simply entering a name for the new bushing and then selecting the **Add** button.



Three different types of bushing are available:

- [Standard Bushing](#)
- [Frequency Bushing](#)
- [VI-MxMount Bushing](#)

The following parameters are common for both bushing types:



The **Active** toggle button allows the user to activate/deactivate the bushing.

Topological Properties

- **Bushing Type**

Allows the user to select the type of the bushing. Available choices are: *standard* , *frequency* .

- **I Body**

Specify the first body where the bushing is attached to.

- **J Body**

Specify the second body where the bushing is attached to.

Geometric Properties

- **Location**

X,Y,Z locations, expressed in [Vehicle Reference System](#) at design time.

- **Orientation**

Psi, Theta, Phi angles, expressed in [Vehicle Reference System](#) at design time.

Stiffness Properties

- **Elastic Force**

Use spline to define characteristic curves (bushing force vs translational deformation).

- **Elastic Torque**

Use spline to define characteristic curves (bushing torque vs rotational deformation).

Damping Properties

- **Translational Damping**

Constant values defining the three translational damping coefficients. The coefficients multiply the relative translational velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

- **Rotational Damping**

Constant values defining the three rotational damping coefficients. The coefficients multiply the relative angular velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

Preload

- **Translational Preload**

Initial force applied to the bushing in design conditions.

- **Rotational Preload**

Initial torque applied to the bushing in design conditions.

Offset

- **Translational Offset**

Initial translational deformation applied to the bushing in design conditions.

- **Rotational Offset**

Initial angular deformation applied to the bushing in design conditions.

Scaling Factors

- **Translational Scalings**

Constant values defining the three translational scaling factors which multiply the bushing Elastic Force characteristic.

- **Rotational Scalings**

Constant values defining the three rotational scaling factors which multiply the bushing Elastic Torque characteristic.

When standard bushing type is selected, the following additional panels appear:

Element Properties		Along X	Along Y	Along z	About X	About Y	About z
Elastic Properties		Plot/Edit Data					
Damping Properties		10.0	10.0	10.0	11.0	11.0	11.0
Preload		0.0	0.0	0.0	0.0	0.0	0.0
Offset		0.0	0.0	0.0	0.0	0.0	0.0

Scaling Factors		Along X	Along Y	Along z	About X	About Y	About z
Force		1.0	1.0	1.0	1.0	1.0	1.0

Element Properties

- **Elastic Properties (Spline)**

Use spline to define characteristic curves (bushing force vs translational deformation & bushing torque vs rotational deformation)

- **Damping Properties**

Constant values defining the translational and rotational damping coefficients. The coefficients multiply the relative translational and angular velocity between the I marker (belonging to I Body) and the J marker (belonging to J Body).

- **Preload**

Initial forces and torques applied to the bushing in design conditions.

- **Offset**

Initial translational and angular deformations applied to the bushing in design conditions.

Scaling Factors

- **Force**

Constant values defining the translational and rotational scaling factors which multiply the bushing Elastic characteristic.

Damping Properties

- **Translational Damping**

Constant translational damping value (force/velocity units).

- **Rotational Damping**

Constant translational damping value (torque/angular velocity units).

The standard bushing allows to define a force connection between two parts ([I Body](#) and [J Body](#)) in the location defined by the user in [Geometric Properties](#) tab.

For each degree of freedom a force (torque), function of relative displacement (rotation) and velocity (angular velocity), will be applied between the two parts.

The elastic contribute of the force (torque) is mapped as a function of the relative displacement (rotation) in the Elastic Properties section.

The damping contribute is defined by means of a constant value in Damping Properties section.

When frequency bushing type is selected, the following additional panel appears:

Geometric Properties						
	X	Y	Z	Psi	Theta	Phi
Location	-360.0	0.0	412.0	-180.0	0.0	0.0
Element Properties						
	Along X	Along Y	Along z	About X	About Y	About z
Elastic Properties	Plot/Edit Data					
Loss Angle Properties	0.0	0.0	0.0	0.0	0.0	0.0
Preload	0.0	0.0	0.0	0.0	0.0	0.0
Offset	0.0	0.0	0.0	0.0	0.0	0.0
Scaling Factors						
	Along X	Along Y	Along z	About X	About Y	About z
Force	1.0	1.0	1.0	1.0	1.0	1.0

Element Properties

- **Elastic Properties (Spline)**

Use spline to define characteristic curves (bushing force vs translational deformation & bushing torque vs rotational deformation)

- **Loss Angles**

Constant loss angle for the translational and rotational degrees of freedom

- **Preload**

Initial forces and torques applied to the bushing in design conditions.

- **Offset**

Initial translational and angular deformations applied to the bushing in design conditions.

Scaling Factors

- **Force**

Constant values defining the translational and rotational scaling factors which multiply the bushing Elastic characteristic.

Damping Properties

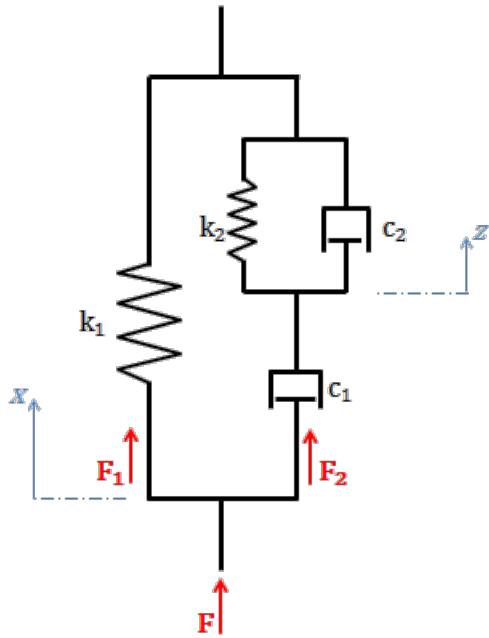
- **Translational Damping**

Constant translational damping value (force/velocity units).

- **Rotational Damping**

Constant translational damping value (torque/angular velocity units).

The frequency bushing relies on the following model:



The force F of the frequency bushing is computed as the sum between F_1 and F_2 , where:

$$F_1 = k_1 \cdot x$$

$$F_2 = c_1 \cdot (\dot{x} - \dot{z}) = c_2 \cdot \dot{z} + k_2 \cdot z$$

The total force is:

$$F = F_1 + F_2 = k_1 \cdot x + c_1 \cdot (\dot{x} - \dot{z})$$

Defining the following auxiliary parameters:

$$\alpha = \frac{k_2}{k_1}$$

$$\beta = \frac{c_2}{c_1}$$

$$\gamma = \frac{c_1}{k_1}$$

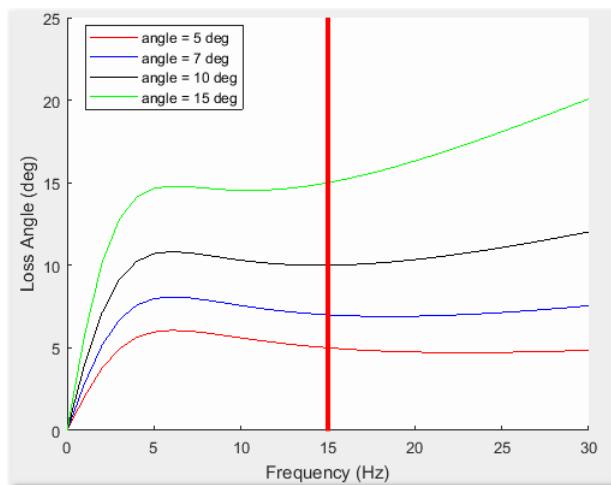
and expressing z as a function of x and \dot{z} we have:

$$F = k_1 \cdot [x + \gamma \cdot (\dot{x} - \dot{z})]$$

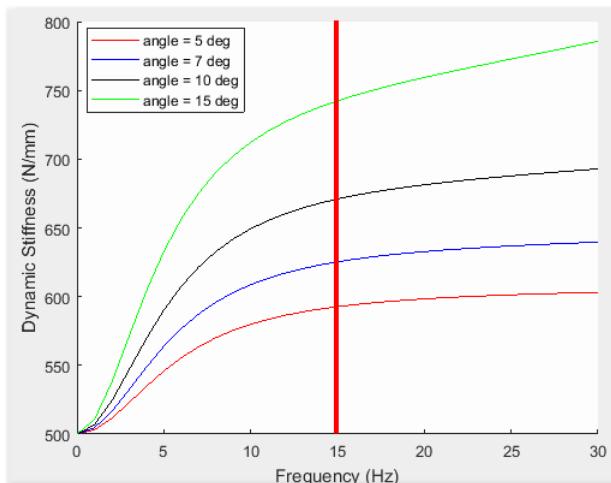
$$\dot{z} = \frac{1}{1 + \beta} \cdot \left(\dot{x} - \frac{\alpha}{\gamma} \cdot z \right)$$

k_1 coefficient is computed by interpolating the spline data from the related Stiffness Properties table.

The Fourier Transform of the F is computed and the phase vs. frequency curve is used to scale the coefficients α, β and γ in order to obtain the target loss angle value at 15 Hz:



The dynamic stiffness varies accordingly with the loss angle:



Once α, β and γ coefficients have been computed it is possible to compute the frequency bushing force.

When the `vi_mount` option is selected as *Bushing Type* then the following panel is shown.

Topological Properties

Bushing Type	vi_mount
I Body	body1
J Body	front_chassis

VI-mount Property File

Property File	mdids://carrealtimeshared/bushings.tbl/hydromount_example.ubf	<input type="button" value="File"/>	<input type="button" value="Edit"/>
---------------	---	-------------------------------------	-------------------------------------

Geometric Properties

Location	X	Y	Z	Psi	Theta	Phi
	117.0	400.0	723.0	-180.0	0.0	0.0

Element Properties

	Along X	Along Y	Along z	About X	About Y	About z
Force Model	elastomer	elastomer	hydromount	elastomer	elastomer	elastomer
Preload	0.0	0.0	0.0	0.0	0.0	0.0
Offset	0.0	0.0	0.0	0.0	0.0	0.0

Scaling Factors

	Along X	Along Y	Along z	About X	About Y	About z
Force	1.0	1.0	1.0	1.0	1.0	1.0
Displacement	1.0	1.0	1.0	1.0	1.0	1.0
Rubber Stiffness	1.0	1.0	1.0	1.0	1.0	1.0
Chamber Stiffness	1.0	1.0	1.0	1.0	1.0	1.0
Resonance Frequency	1.0	1.0	1.0	1.0	1.0	1.0

The following additional panels contain the specific VI-MxMount parameters:

VI-mount Property File

- **Property file**
Property file path for the VI-MxMount bushing (.ubf file extension).

Element Properties

- **Force model**

Model type for each direction (*elastomer /hydrobushing /hydromount /none*). The type is available for the selection if found in the property file for the given direction. Select *none* to get a null force.

- **Force preload**

Output force/torque offset.

- **Displacement offset**

Input displacement/rotation offset.

Scaling Factors

- **Force**

Output force/torque scaling.

- **Displacement**

Input displacement/rotation scaling.

- **Rubber stiffness**

Scaling for rubber stiffness .

- **Chamber stiffness**

Scaling for chamber stiffness (enabled for *hydrobushing /hydromount*) .

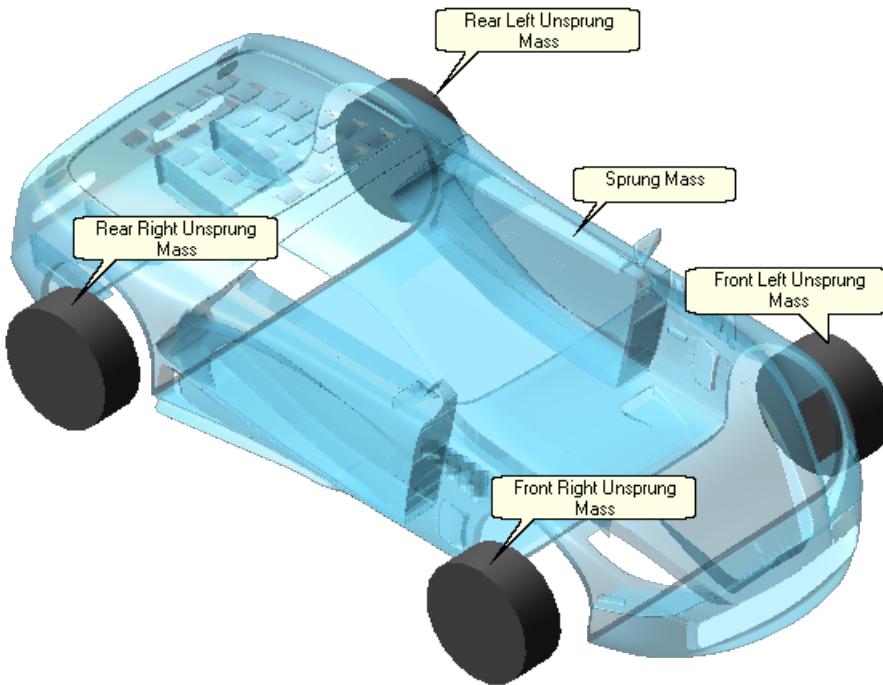
- **Resonance frequency**

Scaling for resonance frequency (enabled for *hydribushing / hydromount*) .

For more information about the elastomer model, the hydribushing/hydromount types and the property file format, refer to the [UBF file documentation](#).

Wheels Subsystem

The wheels subsystem properties collect mass, inertia tire property file information of unsprung mass pairs of VI-CarRealTime vehicle model.



Wheels subsystem Property Editor includes the following data:

- **Tire Property File**

property file for tire forces computation; supported format are: [VI_TIRE](#), MF_05, PAC2002, PAC_CT, FTire, MF-Tyre/MF-Swift, user defined tires.

- **Spin Inertia (one wheel)**

it sets the spin inertia of the parts rotating with the wheel like: rim, tire, brake disk, driveshaft

- **Ixx (hub carrier + wheel)**

Ixx moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.

- **Iyy (hub carrier + wheel)**

Iyy moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.

- **Izz (hub carrier + wheel)**

Izz moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.

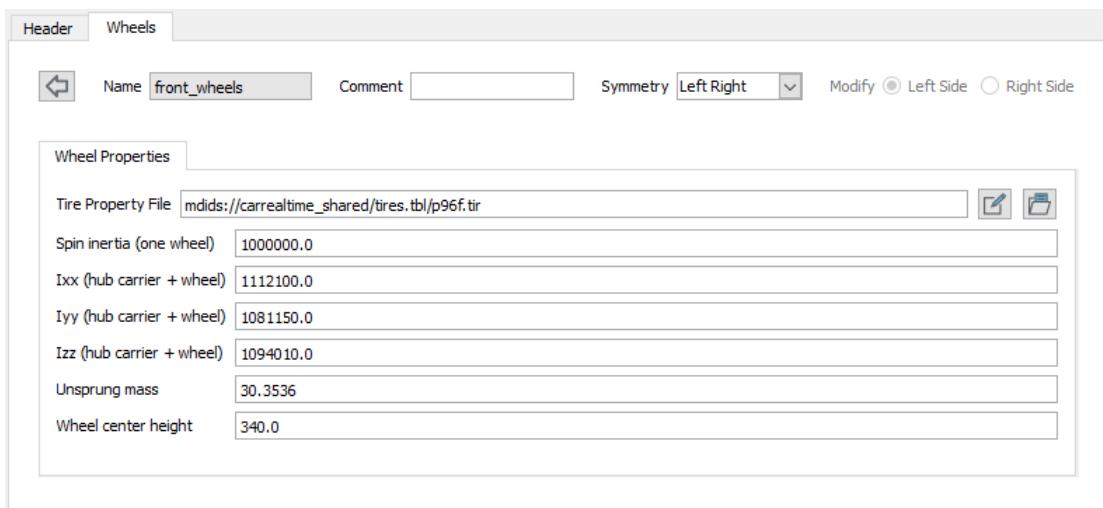
- **Unsprung Mass**

It includes the mass of a single wheel and the portion of suspension masses not belonging to sprung part.

- **Wheel Center Height**

It is the vertical position of wheel center expressed in [Global Reference](#) Frame.

Note: Cross moment of inertia of unsprung mass are neglected.



Rear Twin Wheels Subsystem

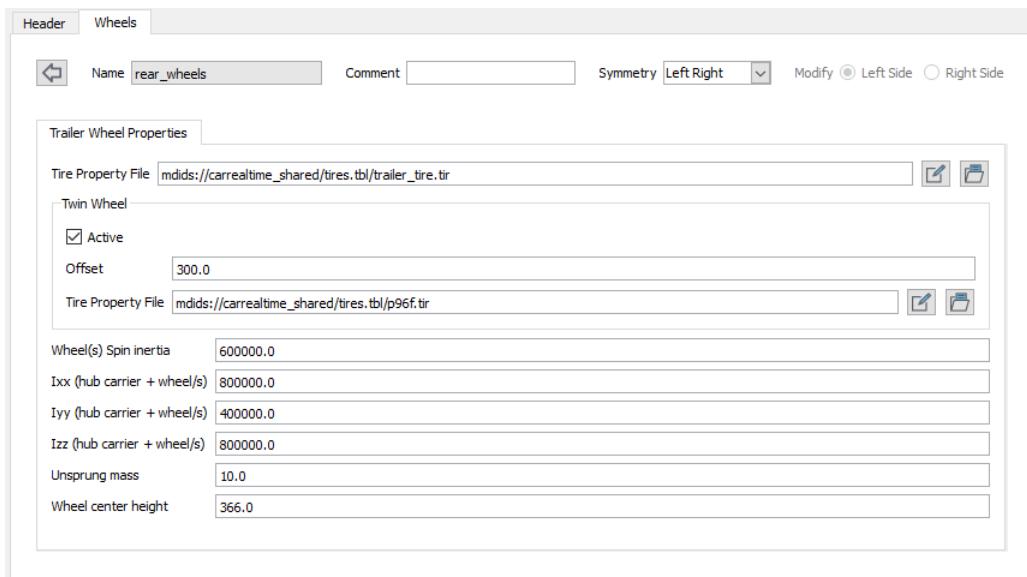
Only for rear wheels there is the possibility to model a twin tire wheel set.

The Rear Twin wheels subsystem properties collect mass, inertia tire property file information of unsprung mass pairs of VI-CarRealTime vehicle model.

Wheels subsystem Property Editor includes the following data:

- **Tire Property File**
property file for tire forces computation; supported format are: MF_05, PAC2002, PAC_CT, VI_TIRE.
- **Twin Wheel**
Check the Twin Wheel box to activate twin wheels and second wheel parameters:
 - **Offset**
set the distance from the first wheel to the second one (positive values).
 - **Tire Property File**
property file for tire forces computation; supported format are: MF_05, PAC2002, PAC_CT, VI_TIRE.
- **Spin Inertia (one wheel)**
it sets the spin inertia of the wheel.
- **Ixx (hub carrier + wheel)**
Ixx moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Iyy (hub carrier + wheel)**
Iyy moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Izz (hub carrier + wheel)**
Izz moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Unsprung Mass**: it includes the mass of a single wheel and the portion of suspension masses not belonging to sprung part.
- **Wheel Center Height**: it is the vertical position of wheel center expressed in [Global Reference Frame](#).

Note: Cross moment of inertia of unsprung mass are neglected.



Brakes Subsystem

The VI-CarRealTime vehicle model allows open or closed-loop brake torques to be specified at the wheels. The brake model in the vehicle model represents a four-wheel disk brake configuration.

All brake parameters associated with the brake on one wheel can be specified independently. In Adams Car, left-right symmetry is assumed.

The Low-speed brake model is quite robust near zero speed. This allows the vehicle to reasonably approximate coming to a dead stop.

The brake model includes the following parameters:

- **bias_front**
gives the portion of braking system pressure going to front wheels.
- **master_cylinder_pressure_gain**
it is defined as the maximum master cylinder pressure divided by maximum [brake demand](#).
- **mu** (front/rear)
friction coefficient
- **effective_piston_radius** (front/rear)
radius for applying friction force.
- **piston_area** (front/rear)
- **lockup_natural_frequency** (front/rear)
the brake model includes a 1DOF spring-damper system to model the wheel lockup; the parameter sets system natural frequency.
- **lockup_damping_ratio** (front/rear)
the parameter sets lockup model damping.
- **lockup_speed** (front/rear)
the parameter sets the lockup speed.

		Left Value	Right Value	Comment
bias_front	0.6			
master_cylinder_pressure_gain	0.1			
mu	0.4	0.4		
effective_piston_radius	177.0	177.0		
piston_area	2300.0	2300.0		
lockup_natural_frequency	10.0	10.0		
lockup_damping_ratio	1.0	1.0		
lockup_speed	139.0	139.0		
mu	0.4	0.4		
effective_piston_radius	167.0	167.0		
piston_area	2300.0	2300.0		
lockup_natural_frequency	10.0	10.0		
lockup_damping_ratio	1.0	1.0		
lockup_speed	139.0	139.0		

Brake System Model Description

For each unsprung mass a brake moment acts on the associated body W and is reacting on the associated body K as the vector:

$$\mathbf{M}_{\text{brk}} = M_{\text{brk}} \mathbf{w}_y$$

where its magnitude is the result of a call to the function LowSpeedBraking plus an external input:

$$M_{\text{brk}} = \text{LowSpeedBraking}(M_{\text{nom}}_{\text{brk}}, brk, brk_loc_state, brk_loc_rot, LkSpd_{\text{brk}} / H_{\text{wc}}, LkFreq_{\text{brk}}, LkRd_{\text{brk}}) + \text{INPUT}(brk_mmag_unsign)$$

The LowSpeedBraking function returns the argument $M_{\text{nom}}_{\text{brk}}$, if the speed is above a threshold set by the parameter $LkSpd_{\text{brk}}$. The parameter H_{wc} is the wheel-center height. The argument $M_{\text{nom}}_{\text{brk}}$ is the nominal brake torque defined below. For speeds less than $LkSpd_{\text{brk}}$, the brake model transitions to a proportional, rotational spring-damper that tends to lock the rotation of bodies W relative to bodies K. When this spring-damper is engaged the brake is considered to be "locked" and the value of the auxiliary-algebraic state brk_loc_state is 1. Otherwise the brake is not locked and the value of brk_loc_state is 0. When the brake is locked, the auxiliary-algebraic state brk_loc_rot stores the current value of the rotation of W relative to the value at which the brake became locked. The behavior of the spring-damper can be tuned by adjusting the parameters $LkFreq_{\text{brk}}$ and $LkRd_{\text{brk}}$, the locked natural frequency and damping ratio, respectively. The values of the spring and damping rates are adjusted, assuming a rotational inertia of I_w so that the wheel has a natural frequency and damping ratio equal to $LkFreq_{\text{brk}}$ and $LkRd_{\text{brk}}$.

The nominal brake torque is given by:

$$M_{\text{nom}}_{\text{brk}} = \text{Sign}(2 A_{\text{p}}_{\text{brk}} \mu_{\text{brk}} R_{\text{adpe}}_{\text{brk}} P_{\text{ch}}_{\text{brk}} + \text{INPUT}(brk_mmag_sign), -\omega_w)$$

where $A_{\text{p}}_{\text{brk}}$, μ_{brk} , and $R_{\text{adpe}}_{\text{brk}}$ are parameters for the brake piston area, friction coefficient, and effective piston radius. The variable $P_{\text{ch}}_{\text{brk}}$ is the chamber pressure of the brake, defined as

$$P_{\text{ch}}_{\text{brk}} = \text{INPUT}(brk_pres_gain)(P_{\text{in}}_{\text{brk}} + \text{INPUT}(brk_pres))$$

in which P_{ln_brk} is the brake line pressure. The line pressure is defined in terms a parameter for the front-brake bias, $BiasF_{brk}$. For brakes on the front suspension the line pressure is

$$P_{ln_brk} = BiasF_{brk} P_{mc_brk}$$

For brakes on the rear suspension the line pressure is

$$P_{ln_brk} = (1 - BiasF_{brk}) P_{mc_brk}$$

Both line pressure expressions are defined in terms of the master cylinder pressure, P_{mc_brk} , which is in turn defined as

$$P_{mc_brk} = R_{mc} Brk_{dm} + INPUT(brk_mast_cyl_pres)$$

where:

- R_{mc} is the master cylinder pressure gain
- Brk_{dm} is the driver [brake demand](#).

Note: the dimensions of R_{mc} are force/length².

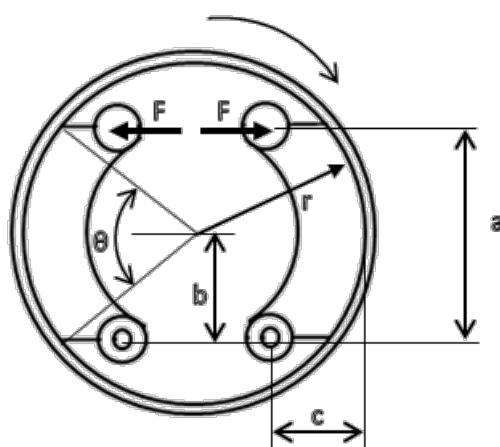
Note: There are several inputs associated with the brake system. Their purposes are summarized in the following table:

Input	User-model purpose
INPUT(brk_mast_cyl_pres)	Master-cylinder brake pressure.
INPUT(brk_pres)	Brake-chamber pressure.
INPUT(brk_pres_gain)	ABS on/off switch control without modeling pressure system. Note: this is the only input in the vehicle model that has a default value of 1. (All others default to zero.)
INPUT(brk_mmag_sign)	Nominal brake torque but retaining the vehicle model low-speed braking model.
INPUT(brk_mmag_unsign)	Complete brake system model.

Note: For further details about brake system, please refer to [About External Brake Model](#) topic.

Drum Brakes

As alternative to the disk based setup, VI-CarRealTime offers a drum brake model:



With respect to the standard brake parameters, the drum brake defines also the following additional parameters:

- **leading_shoes**

number of active leading shoes. The parameter can assume the following values:

1: there is one *leading* shoe and one *trailing* shoe; the total braking torque is computed as the sum between trailing torque (M_{TRAIL}) and leading torque (M_{LEAD})

2: there are two *leading* shoes; the total braking torque is the leading torque (M_{LEAD}) multiplied by 2.

- **drum_radius**

r parameter.

- **shoe_anchor_pin_to_cylinder**

a parameter.

- **shoe_anchor_pin_to_shaft**

b parameter.

- **shoe_anchor_pin_to_fulcrum**

c parameter.

- **shoe_engagement_angle**

θ parameter.

The screenshot shows the configuration interface for a VI-grade model. On the left, a tree view displays the following structure under 'brake_parameters': 'brake_system', 'front', and 'rear'. The 'front' node is currently selected and highlighted in blue. On the right, a table lists various parameters with their current values. The table has columns for 'Name', 'Left Value', 'Right Value', and 'Comment'. The 'Left Value' column contains the values from the configuration interface, while the 'Right Value' column is mostly empty except for 'lockup_damping_ratio' which has a value of 1.0. The 'Comment' column is also mostly empty.

Name	Left Value	Right Value	Comment
model	drum		
leading_shoes	2		
mu	0.35	0.35	
drum_radius	210.0	210.0	
piston_area	500.0	500.0	
shoe_anchor_pin_to_cylinder	280.0	280.0	
shoe_anchor_pin_to_shaft	140.0	140.0	
shoe_anchor_pin_to_fulcrum	150.0	150.0	
shoe_engagement_angle	120.0	120.0	
lockup_natural_frequency	10.0	10.0	
lockup_damping_ratio	1.0	1.0	
lockup_speed	139.0	139.0	

Name Type

Here the drum brake equations:

$$M_{LEAD} = \frac{k \cdot \mu \cdot F \cdot a \cdot r}{b - k \cdot \mu \cdot c}$$

$$M_{TRAIL} = \frac{k \cdot \mu \cdot F \cdot a \cdot r}{b + k \cdot \mu \cdot c}$$

with the drum Force (F) computed as:

$$F = p \cdot A$$

p is the drum brake pressure computed as:

$$p = \gamma \cdot k_{MCP} \cdot \%B$$

where:

VI-CarRealTime

- γ is the brake bias;
- k_{MCP} is the master cylinder pressure gain;
- $\%B$ is the driver brake demand percentage.

k coefficient is defined as:

$$k = \frac{4 \cdot \sin\theta}{2\theta + \sin 2\theta}$$

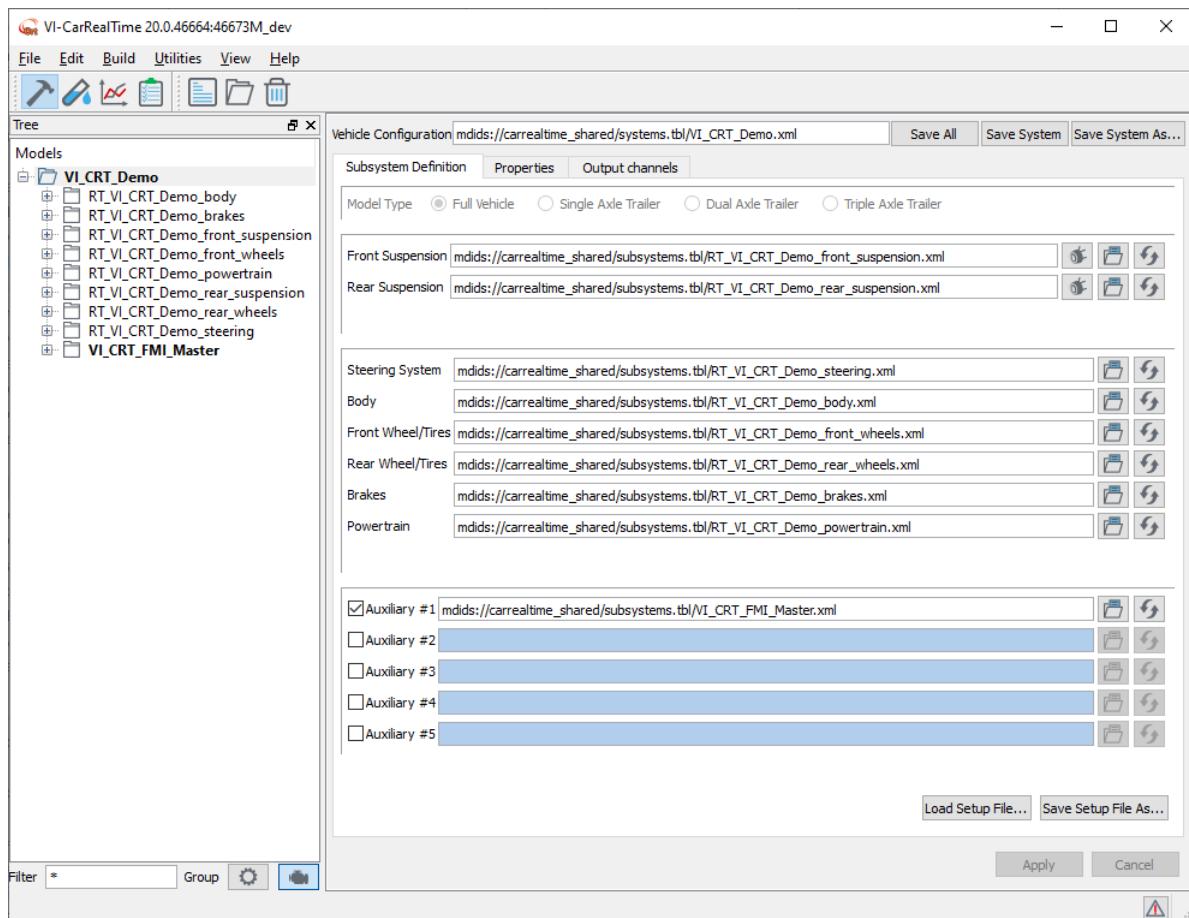
The following two subsystems are provided with carrealtime_shared database:

- *RT_VI_CRT_Demo_brakesFdrumRdrum.xml*
- *RT_VI_CRT_Demo_brakesFdiskRdrum.xml*

Auxiliary Subsystem

In VI-CarRealTime it is possible to add further subsystems in addition to the standard ones. Such additional subsystems are called **Auxiliary** Subsystems.

VI-CarRealTime GUI supports up to 5 Auxiliary subsystems.



The following examples of auxiliary subsystems are described:

- [Rear Steering Subsystem](#)

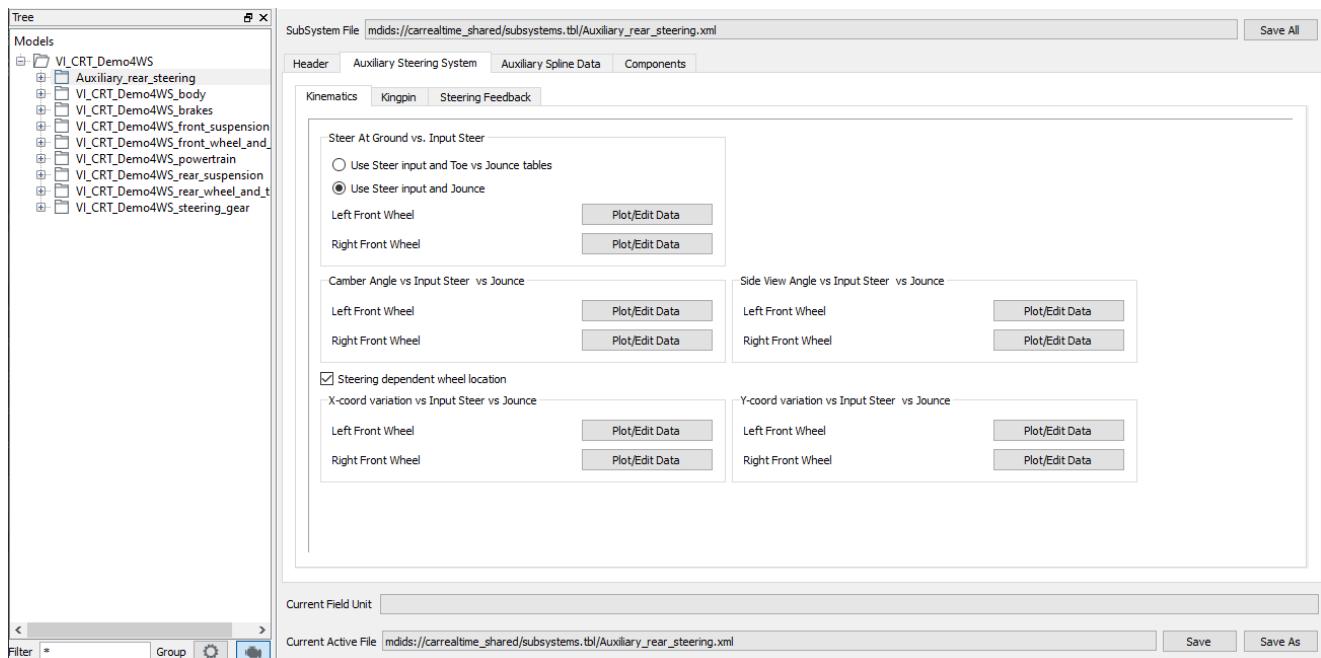
- [Adding external output to VI-CarRealTime](#)
- [Cab Subsystem](#)
- VI-TireScrub

Rear Steering Subsystem

VI-CarRealTime allows to add a rear steering subsystem as an auxiliary subsystem.

The rear steering subsystem models two (left and right) conceptual actuators which allow the rear wheels to steer, depending on the imposed actuators value.

The actuators law can be applied externally by using [Rear Steering Actuator Disp](#) inputs.



The auxiliary rear steering defines a subset of the features defined in the standard steering subsystem, so for all the details please refer to [Steering Subsystem](#) chapter.

The splines defined in the *Kinematics* section supersedes the related splines defined in the rear suspension subsystem according to the standard rules defined in the [Kinematics](#) topic of the standard [Steering Subsystem](#).

An event submitted with the auxiliary rear steering subsystem generates the [Rear Steering System](#) additional outputs.

Note: an example of rear steering auxiliary subsystem (*Auxiliary_rear_steering.xml*) is shipped with *carrealtime_shared* database.

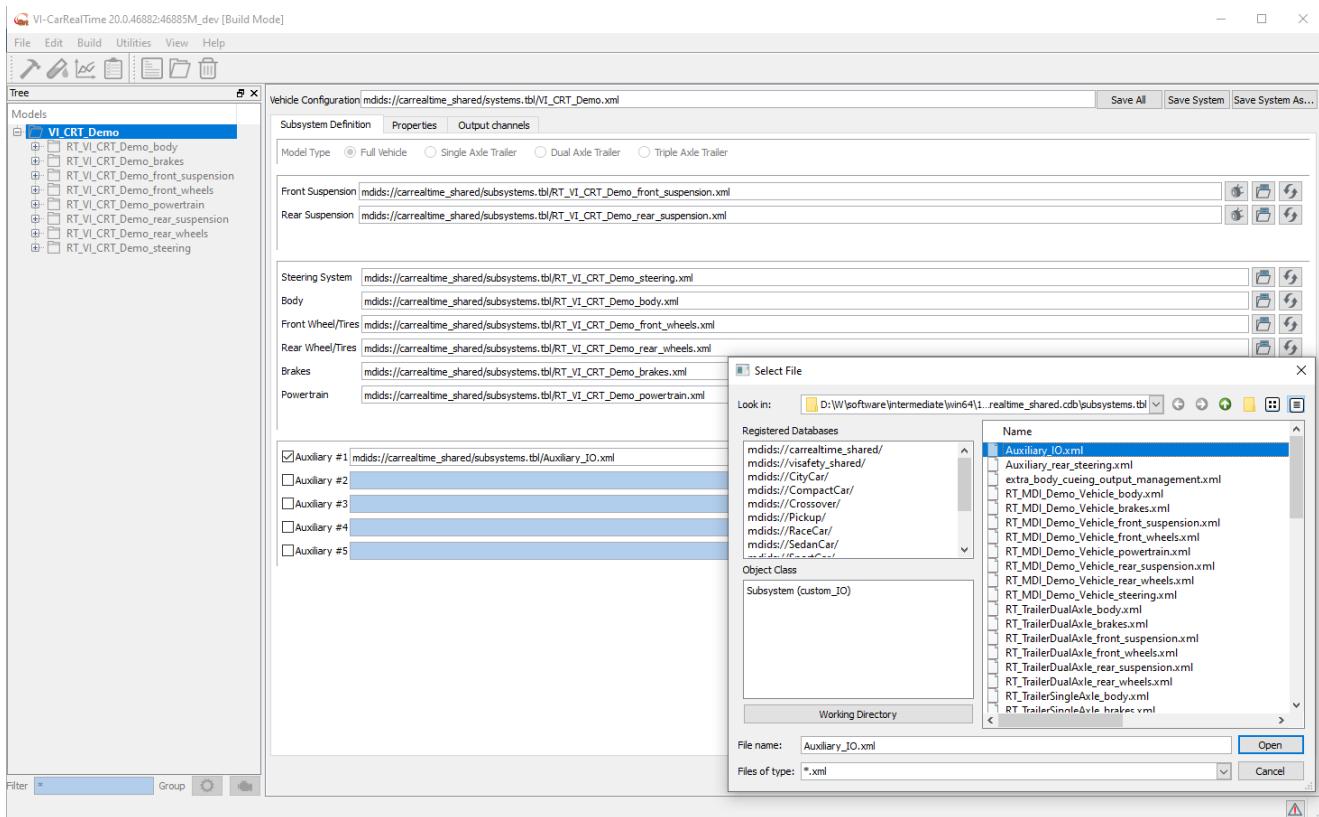
Adding external output to VI-CarRealTime

Adding external output to VI-CarRealTime output file

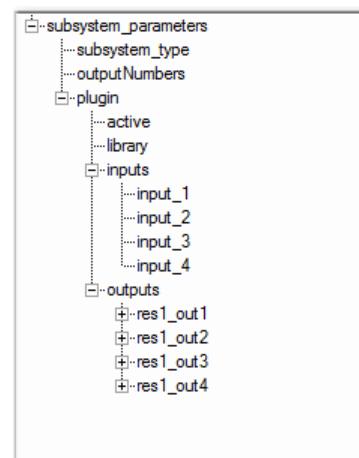
It is possible to add to standard VI-CarRealTime results file a custom set of external channels coming from an external application (MATLAB/Simulink or plugin dll):

In order to use this feature, user should instrument his vehicle model with a dedicated auxiliary subsystem (\$VI-CarRealTime\Installationdir\lacart\carrealtime_shared.cdb\subsystems.tb\Auxiliary_IO.xml).

VI-CarRealTime



Auxiliary_IO template subsystem includes 4 new input and 4 new output channels. User can modify channels names or numbers using VI-CarRealTime GUI or by editing manually the xml file:

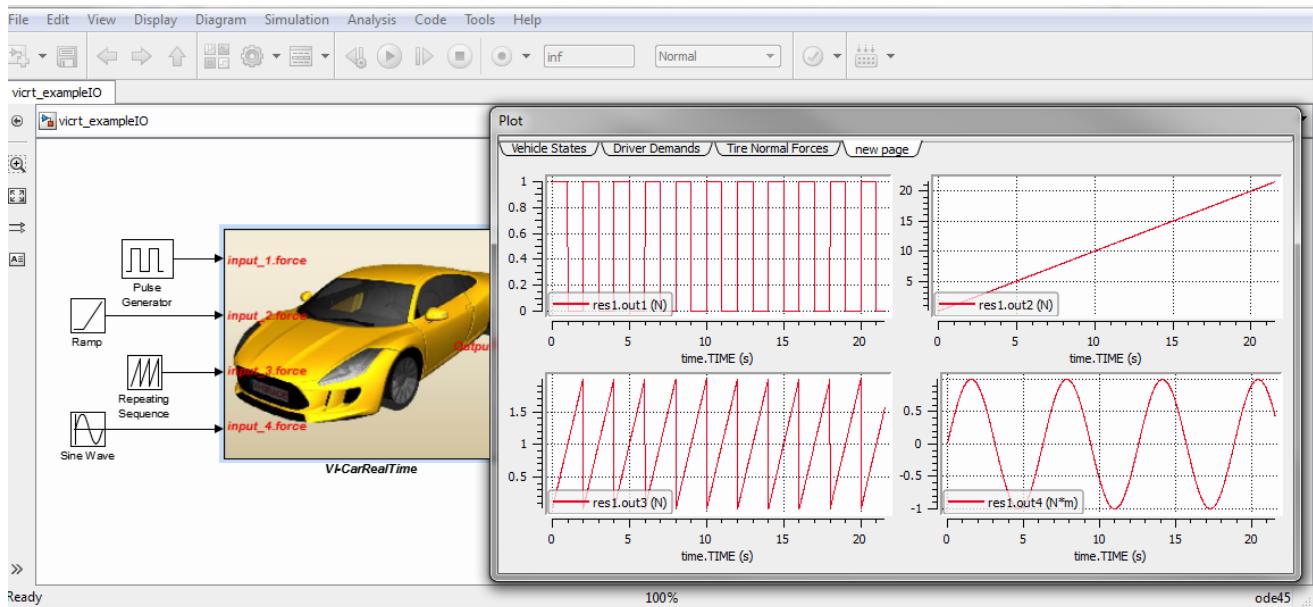


```

<ParameterTree name="inputs" active="true" userDefined="false">CR LF
  <StringParameter name="input_1" active="true" userDefined="false" value="force" />CR LF
  <StringParameter name="input_2" active="true" userDefined="false" value="force" />CR LF
  <StringParameter name="input_3" active="true" userDefined="false" value="force" />CR LF
  <StringParameter name="input_4" active="true" userDefined="false" value="force" />CR LF
</ParameterTree>CR LF
<ParameterTree name="outputs" active="true" userDefined="false">CR LF
  <ParameterTree name="res1_out1" active="true" userDefined="false">CR LF
    <StringParameter name="result_set_component" active="true" userDefined="false" value="res1.out1" />CR LF
    <StringParameter name="units" active="true" userDefined="false" value="newton" />CR LF
  </ParameterTree>CR LF
  <ParameterTree name="res1_out2" active="true" userDefined="false">CR LF
    <StringParameter name="result_set_component" active="true" userDefined="false" value="res1.out2" />CR LF
    <StringParameter name="units" active="true" userDefined="false" value="newton" />CR LF
  </ParameterTree>CR LF
  <ParameterTree name="res1_out3" active="true" userDefined="false">CR LF
    <StringParameter name="result_set_component" active="true" userDefined="false" value="res1.out3" />CR LF
    <StringParameter name="units" active="true" userDefined="false" value="newton" />CR LF
  </ParameterTree>CR LF
  <ParameterTree name="res1_out4" active="true" userDefined="false">CR LF
    <StringParameter name="result_set_component" active="true" userDefined="false" value="res1.out4" />CR LF
    <StringParameter name="units" active="true" userDefined="false" value="newton-meter" />CR LF
  </ParameterTree>CR LF
</ParameterTree>CR LF

```

On MATLAB/Simulink side new inputs will appear in the list of available model inputs, and the new outputs will be included on result file.



Cab Subsystem

Cab suspension is a dedicated solver-module inside VI-CarRealTime main solver. The whole cab's kinematic and dynamic behavior is included in an additional library.

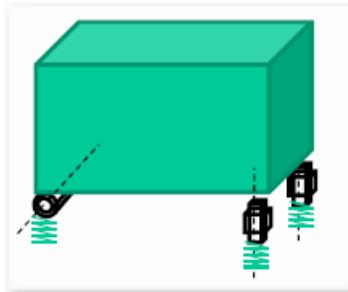
This plug-in approach gives advantages in terms of flexibility and maintenance; the library framework is indeed capable of including future suspension topologies and add more elastic connections between the cab and the basic VI-CarRealTime model.

In general terms, cab model contains no closed-kinematical loops nor non-holonomic constraints. This means that the bodies in the model can be (and are indeed) represented using a tree topology. It further means that the kinematics of each body are described in terms of state variables that define its configuration (position and orientation) and motion (translational and angular velocities), relative to other rigid bodies that are farther up the tree, ultimately terminating on a single root body. The root body of the tree, representing the rigid-body system, is the global frame. The body is a Newtonian or inertial frame, meaning that it does not accelerate in translation and does not rotate.

VI-CarRealTime

The library includes also a middle step Runge-Kutta integrator that is responsible of the integration of cab dynamic equations of motion and an interface part for the interaction of forces between cab solver and VI-CarRealTime standard solver. This interface part is responsible for leading the co-simulation between the two different solvers and maintain the congruency of whole truck dynamic system.

Current library version includes the equations of motion to capture the physics of the particular cab suspension topology shown in the following picture.



Such topology expects the cab part to be connected with respect to the rest of vehicle chassis by means of four elastic supports in order to describe a kinematic system having a cylindrical collar in the front cab part and two supports in the rear.

The cab rigid body represents the part that is supported by elastic bushings.

The body has six degrees of freedom (DOFs), so it can attain any position and/or any orientation in space according to the system of forces generated by elastic elements.

Moreover, a linear elastic beam element connects the cab part to the chassis.

Rigid Part

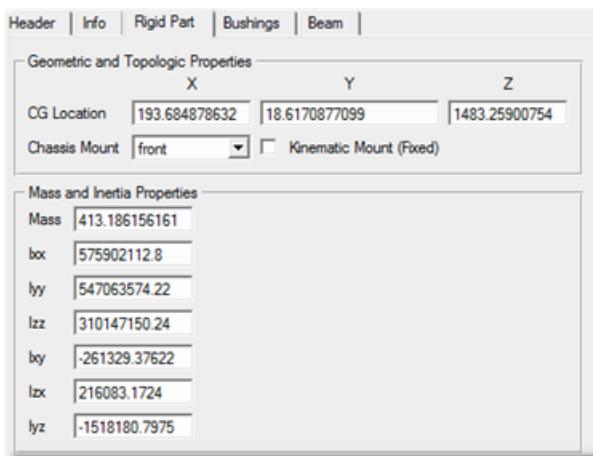
Cab rigid part is characterized by:

Mass and Inertia Properties

- *Mass*
mass of the cab.
- I_{xx} , I_{yy} , I_{zz} , I_{xy} , I_{zx} , I_{yz}
inertia moments of the cab with respect to a marker located in CG location and oriented as the cab reference frame.

Geometric and Topological Properties

- *CG Location*
X,Y,Z location of the cab part, defined with respect to standard VI-CarRealTime [reference frame](#).
- *Chassis Mount*
part the cab part is attached to.



When Kinematic Mount toggle button is unchecked, the connection between the cab part and the chassis part is made with bushings.

Bushings

Bushing supports are massless elements with non linear stiffness and damping properties. The generic bushing applies a force and a torque to two parts (cab and chassis). User specifies the location and orientation for force and/or torque application.

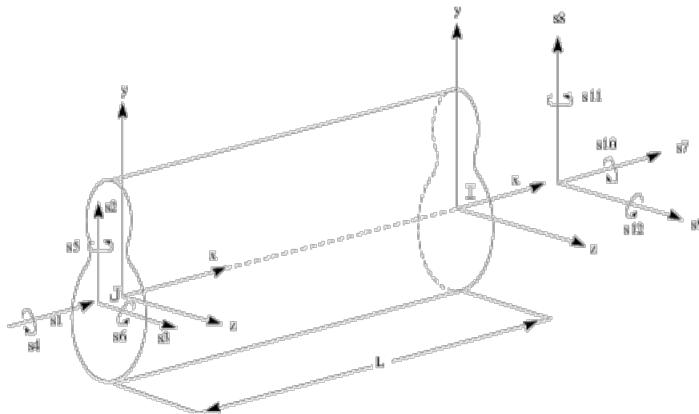
Each force consists of three components in the coordinate system of the application marker, one in the x-axis direction, one in the y-axis direction, and one in the z-axis direction. Likewise each torque consists of three components in the coordinate system of the marker: one about the x-axis, one about the y-axis, and one about the z-axis.

The force is dependent upon the relative displacement and the relative velocity of the two markers (the first on cab and the second on chassis). The torque is dependent upon the relative angle of rotation and the relative rotational velocity of the parts containing the specified markers.

$$\begin{bmatrix} F_x \\ F_y \\ F_z \\ T_x \\ T_y \\ T_z \end{bmatrix} = - \begin{bmatrix} K_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & K_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & K_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & K_{66} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ a \\ b \\ c \end{bmatrix} - \begin{bmatrix} C_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & C_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Beam

Beam connection is a massless elastic beam with a uniform cross section. The beam transmits forces and torques between two markers in accordance with either linear Timoshenko beam theory.



The beam will be oriented with respect to chassis part according Euler angles Psi, Theta, Phi and it will be placed according to its length and the location of its middle point.

Physical Properties			
Young Modulus	207000.0		
Shear Modulus	80232.55814		
Cross Section Area	572.555261117		
Length	918.633333332		
Section Ixx	52174.098169		
Section Iyy	26087.049085		
Section Izz	26087.049085		
Y Shear Ratio	1.0		
Z Shear Ratio	1.0		
Damping Ratio	0.001		

Geometric Properties			
	X	Y	Z
Location	836.346947286	386.441666666	1197.74
	Psi	Theta	Phi
Orientation	90.0	0.0	0.0

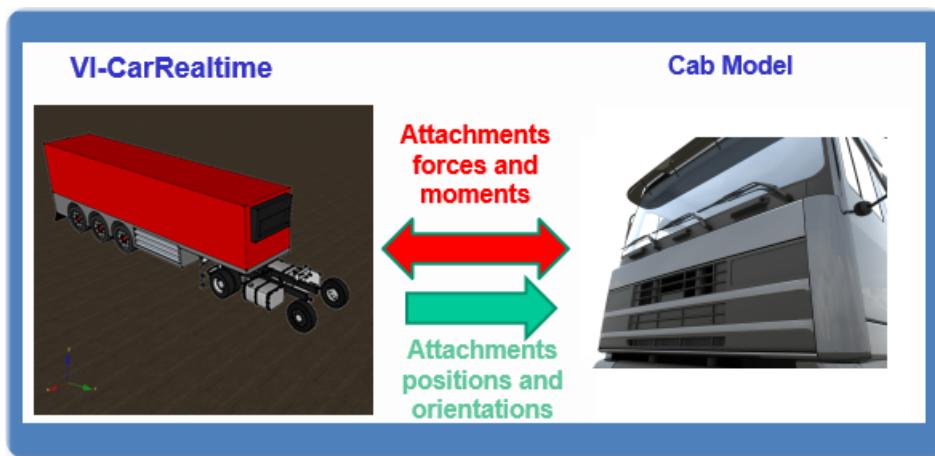
The x-axis of the marker defines the centroidal axis of the beam. The y-axis and z-axis of the marker are the principal axes of the cross section. They are perpendicular to the x-axis and to each other.

$$\begin{bmatrix} F_x \\ F_y \\ F_z \\ T_x \\ T_y \\ T_z \end{bmatrix} = - \begin{bmatrix} K_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{22} & 0 & 0 & 0 & K_{26} \\ 0 & 0 & K_{33} & 0 & K_{35} & 0 \\ 0 & 0 & 0 & K_{44} & 0 & 0 \\ 0 & 0 & 0 & K_{55} & 0 & 0 \\ 0 & K_{66} & 0 & 0 & 0 & K_{66} \end{bmatrix} \begin{bmatrix} x-L \\ y \\ z \\ d \\ b \\ c \end{bmatrix}$$

$$\begin{aligned} K_{11} &= EA/L \\ K_{22} &= 12E[Izz/JL^3(1+P_y)] \\ K_{26} &= -6E[Izz/JL^2(1+P_y)] \\ K_{33} &= 12E[Iyy/JL^3(1+P_z)] \\ K_{35} &= 6E[Iyy/JL^2(1+P_z)] \\ K_{44} &= GIx/L \\ K_{55} &= (4+P_z)E[Iyy/JL(1+P_z)] \\ K_{66} &= (4+P_y)E[Izz/JL(1+P_y)] \end{aligned}$$

where $P_y = 12E[Izz/ASY/(GAL^2)]$ and $P_z = 12E[Iyy/ASZ/(GAL^2)]$

As explained in the first part of the topic, VI-CarRealTime and the external suspension cab model will perform a co-simulation. More in details, during the simulation flow, VI-CarRealTime sends to cab solver the characteristic states of cab suspension attachments (positions, orientations and their derivatives). Cab solver uses these information to solve bushings and beam elements, in order to compute the correct action and reaction forces acting on two numerically separated dynamic models.



When a simulation is performed using the cab model, the solver will produce [additional outputs](#) in the results file.

VI-TireScrub

The aim of the Tire Scrub Effect feature is to provide the driver of dynamic simulators informations about the limit of adhesion of the tires.

It is known by common experience and scientific documentation, that when tires are driven up to their lateral limit, they generate a chattering vibration due to a continuous jump between static and dynamic friction at the contact patch, scrubbing the road surface.

In real-life driving, this hysteretic behavior is transmitted through the chassis and through the steering assembly to be felt by the driver as an indication of which tire is approaching, or reaching, its limits of grip. For this reason, tire scrub phenomenon can be accurately reproduced on the driving simulator only if the effect is coming from tire channels.

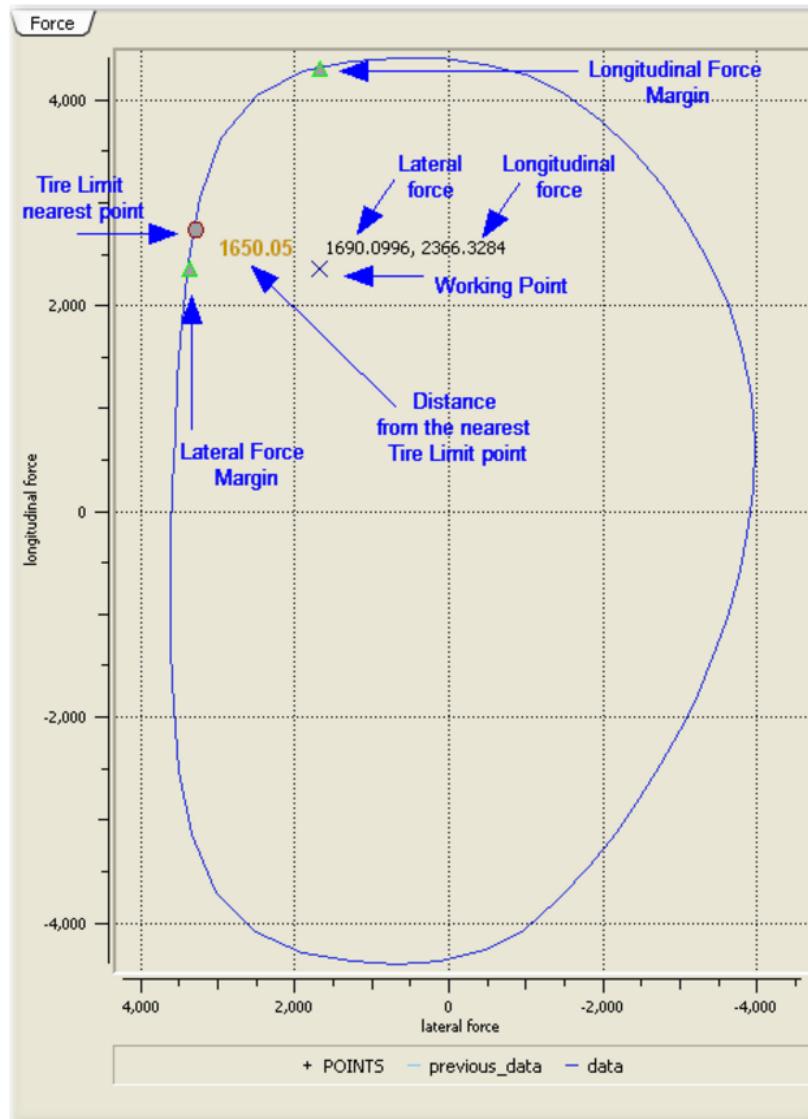
Typical tire models used in driving simulators are too simple to describe such high frequency scrub effect, which makes, by the way, a significant improvement in the overall realism of the driving experience. Thus, an ad hoc implementation of the scrub effect is accomplished by altering the lateral forces output channels coming from the tire model, adding a zero mean contribution that takes into account several physical dependencies, later described in this guide.

Reproduction of the scrub effect at tire-model level is the best way to give consistent additional information to the driver: chattering lateral tire forces are reflected by tire aligning torques and sprung mass accelerations channels, which in turn are the inputs for steering torque feedback and VI-MotionCueing algorithm. With a proper tuning of scrub effect control variables an informative and natural feeling is achieved when the driver is performing maneuvers close to the limit of adhesion of the tires.

The grip condition of each tire is evaluated with the internal VI-TireLimits utility and it's complemented by specific output channels; the instantaneous working point of a tire model is defined by:

- lateral slip angle and longitudinal slip ratio
- camber angle
- tire normal force

For each set of conditions, VI-TireLimits tool can compute the maximum possible friction ellipse area and calculate the distance of the instantaneous working point from the boundaries of the ellipse. For the scrub effect implementation, the distance from the working point to the nearest lateral bound is the main indicator of a tire approaching/reaching its lateral limit of adhesion.



According to VI-CarRealTime documentation, the distance from working point to lateral bound is defined for each tire by the following RTDB variables:

- VI_CarRealTime.Outputs.UserSensor.tirelimits_FL.marginLat
- VI_CarRealTime.Outputs.UserSensor.tirelimits_FR.marginLat
- VI_CarRealTime.Outputs.UserSensor.tirelimits_RL.marginLat
- VI_CarRealTime.Outputs.UserSensor.tirelimits_RR.marginLat

Note: VI-TireLimits can work only with Pacejka and MF-Tyre formulations.

The basic principle beyond the Tire Scrub Effect is adding a sinusoidal, zero-average disturbance to the lateral force channels. Based on the input variables that are governing the algorithm, this disturbance signal is instantaneously stretched in amplitude and frequency by means of a chain of multipliers.

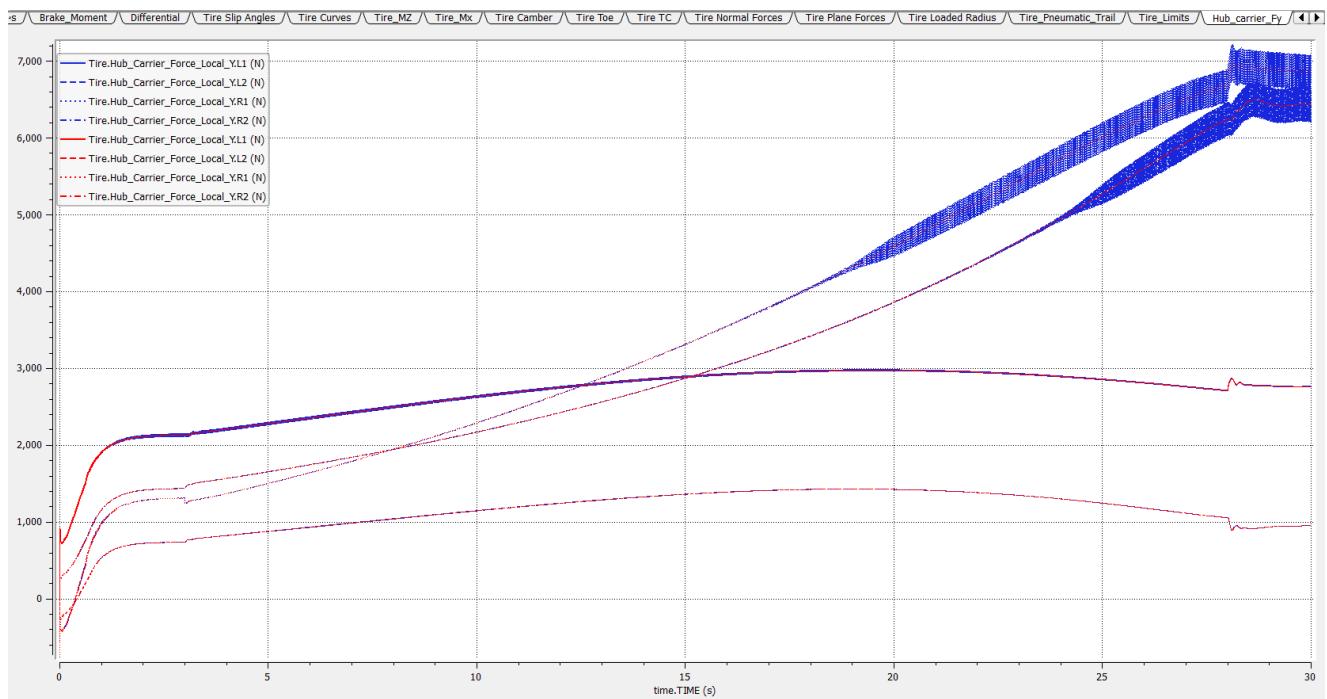
There are 4 gains acting as multipliers, each of them scaling the zero-average disturbance with respect to the relative dependency:

- Lateral Margin (linked to the internal TireLimits dependencies previously listed)
- Tire Longitudinal Slip Ratio
- Tire Normal Load
- Wheel Omega

Baseline case study

To explain how each variable is affecting the output channels, a Constant Radius Circle maneuver is used as an example. In this maneuver a FWD car is driven to its lateral limit, gradually building up its lateral acceleration, until the Tire Scrub mechanism is activated and the disturbance is added to lateral forces at each corner of the car.

In the following plots, lateral forces for the four hub carriers are shown: comparison is between the maneuver run without Tire Scrub Effect (red) and with Tire Scrub Effect (blue).



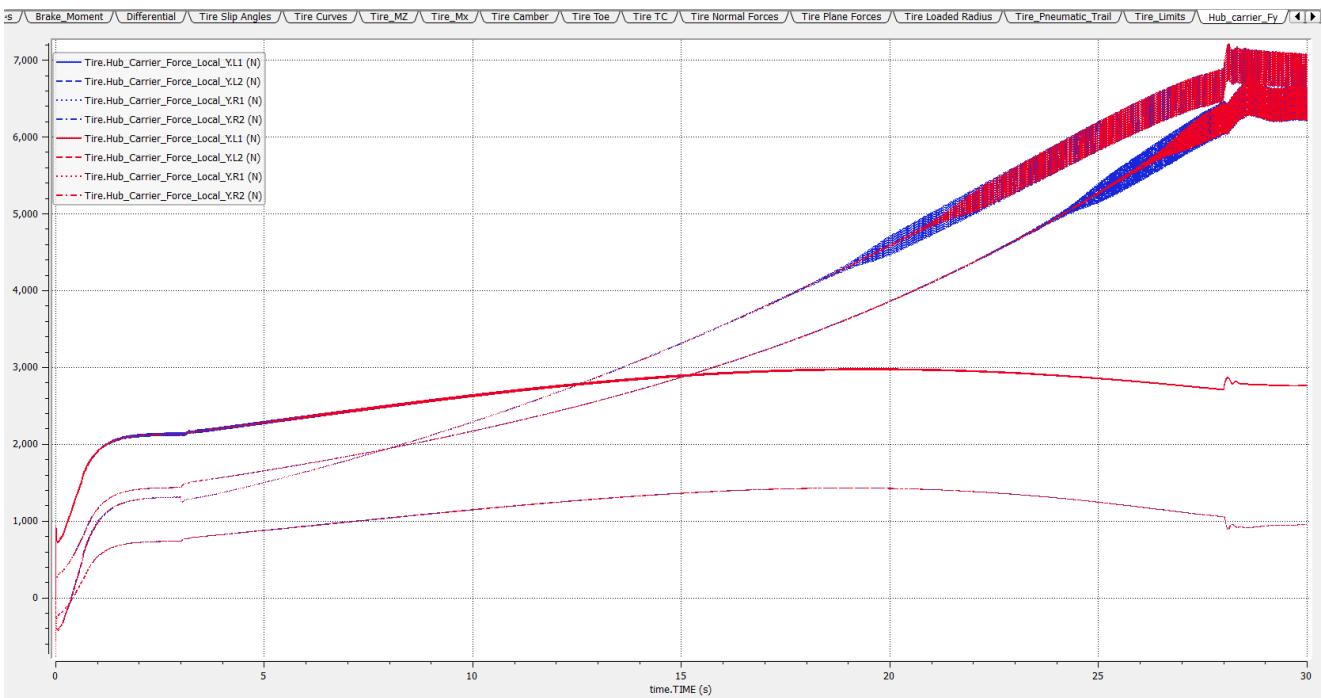
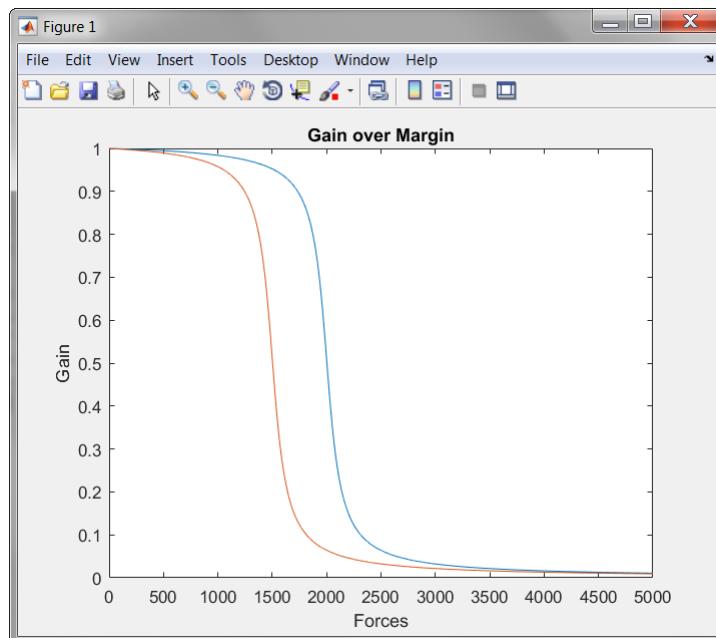
It's possible to notice that forces for the loaded pair of wheels, external to the corner, are growing undisturbed until the Tire Scrub Effect intervenes.

Gain Over Lateral Margin

This gain is managing the tire scrub activation with respect to the lateral margin computed by internal Vi-TireLimits tool. The multiplier is defined by a semi-gaussian function that can be tuned by means of two parameters, Threshold_M and Smoothness_M, to modify the shape of the gain.

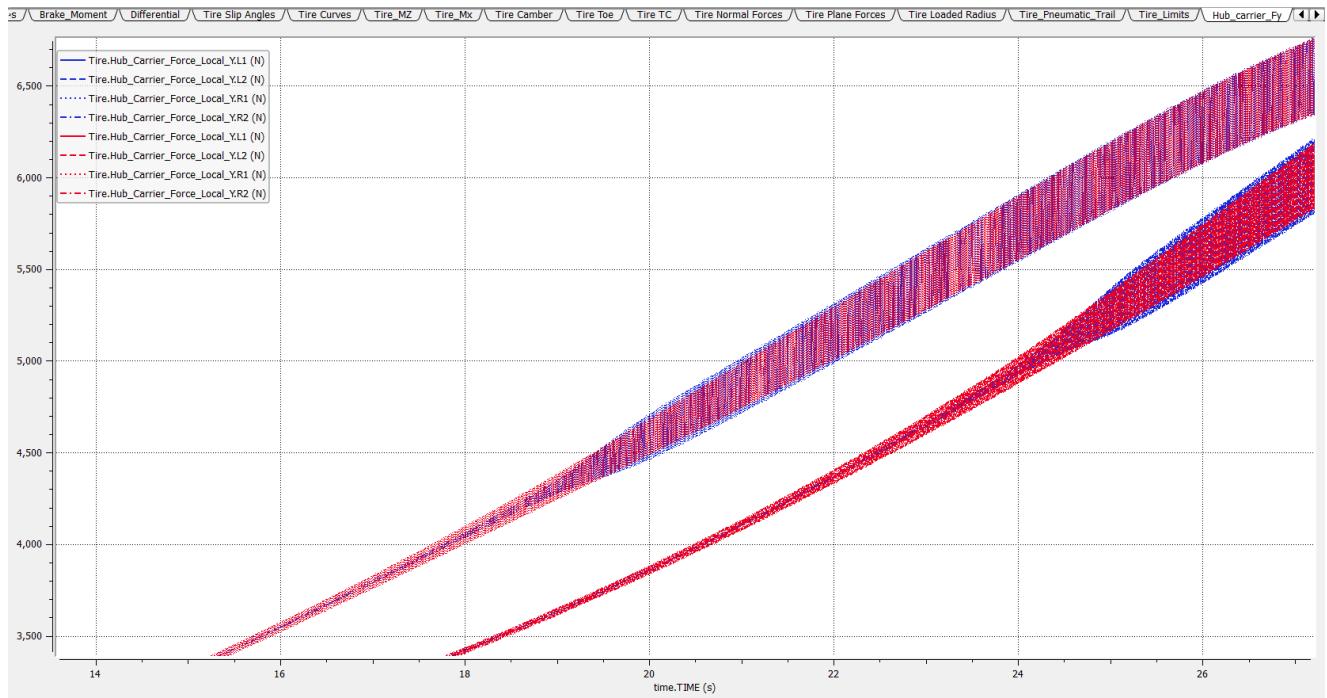
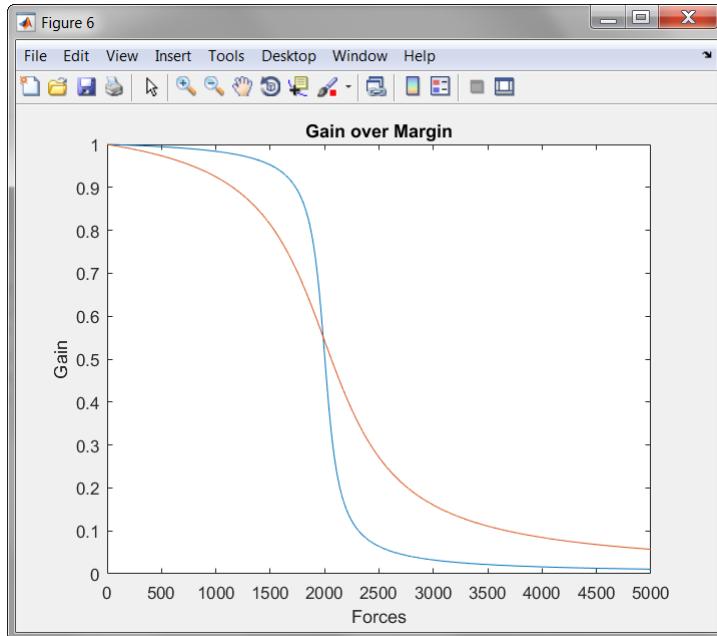
Threshold_M represents the lateral margin force (measured in Newton): the lower the value, the later the effect is activated in terms of distance between the working point of the tire and its instantaneous limit friction ellipse. In the following pictures, the test case maneuver is run with 2 different threshold forces, highlighting the effect of Threshold_M in isolation.

Case	Parameter Value	Analysis Colour
Baseline	Threshold_M=2000[N]	Blue
Mod	Threshold_M=1500[N]	Red



With the Smoothness_M parameter, it is instead possible to tune the slope of the gain, managing the smoothness of the transition between 0 and 1 gain: the higher the number, the steeper the slope, the harsher the transition. The effect of this parameter in isolation is shown in the following pictures.

Case	Parameter Value	Analysis Colour
Baseline	Smoothness_M=1	Blue
Mod	Smoothness_M=0.2	Red



Gain Over Longitudinal Slip

This gain is used to manage the activation of Tire Scrub Effect with respect to the longitudinal slip of the tire, in situations where the tires are worked in combined grip conditions. The physical phenomenon of tire scrub is typically erased when high longitudinal slips are present in combination with low lateral margins, because the tire

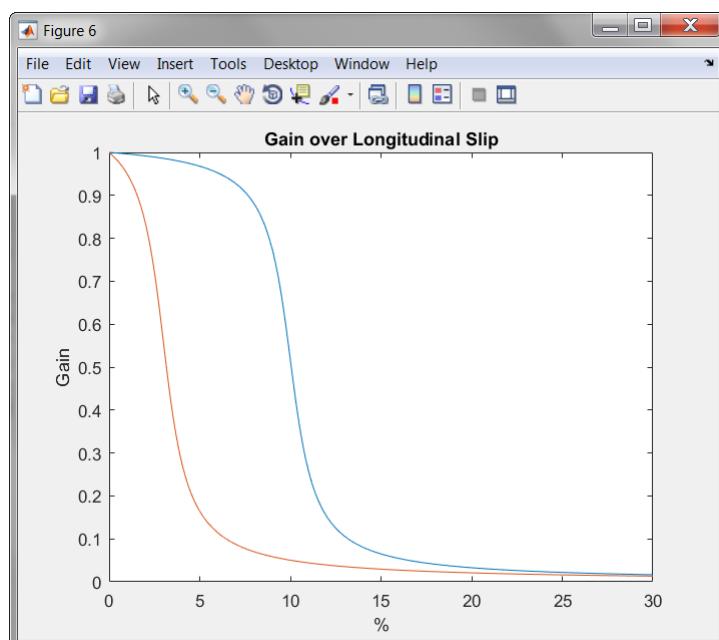
VI-CarRealTime

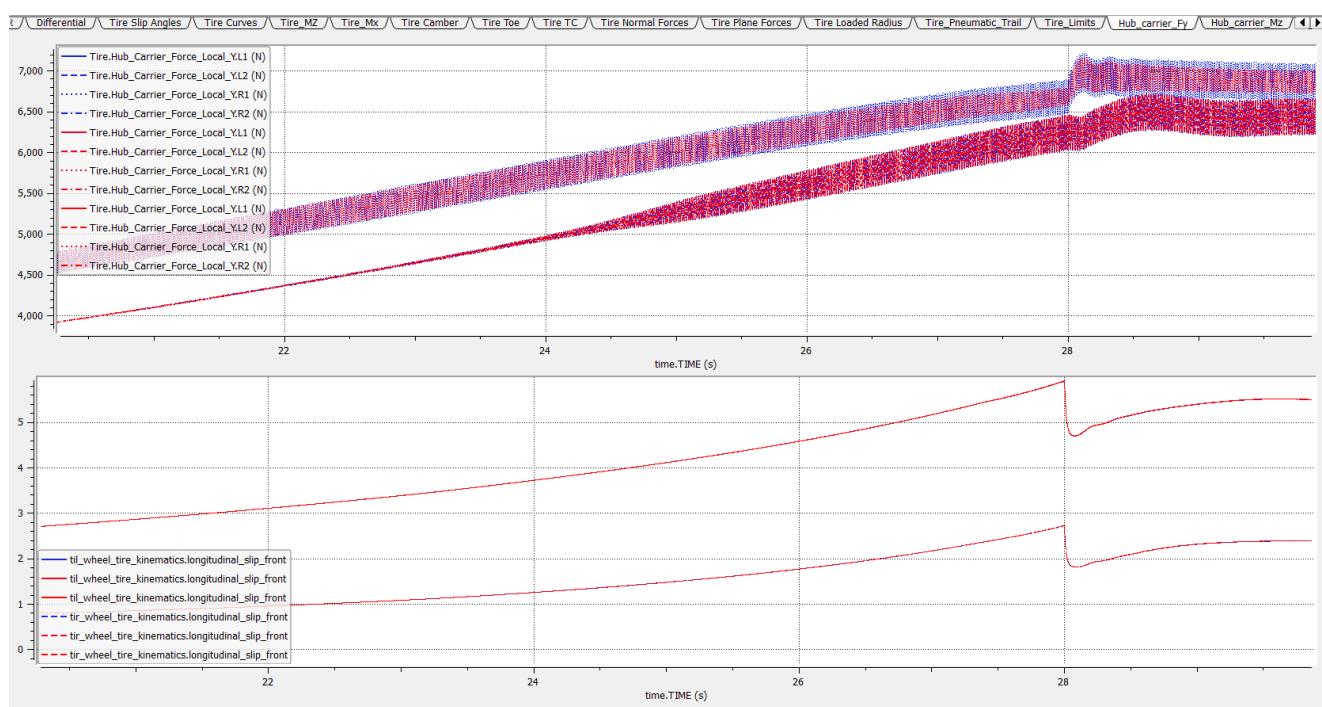
is pushed towards complete dynamic friction at contact patch. This is particularly true when power is applied on a FWD car while the front tires are already near their lateral friction limits.

The dependency from longitudinal slip is implemented with a semi-gaussian function, analogous to the gain over lateral margin. The two tuning parameters are Threshold_LS and Smoothness_LS.

In the following pictures, the test case maneuver is run with two different threshold slips (measured in percentage), highlighting the effect of Threshold_LS in isolation.

Case	Parameter Value	Analysis Colour
Baseline	Threshold_LS =10[%]	Blue
Mod	Threshold_LS =4[%]	Red





It is possible to observe that if Threshold_LS is reduced (red analysis), the scrub effect is also reduced on the front loaded wheel as soon as the value of 4% of longitudinal slip is exceeded.

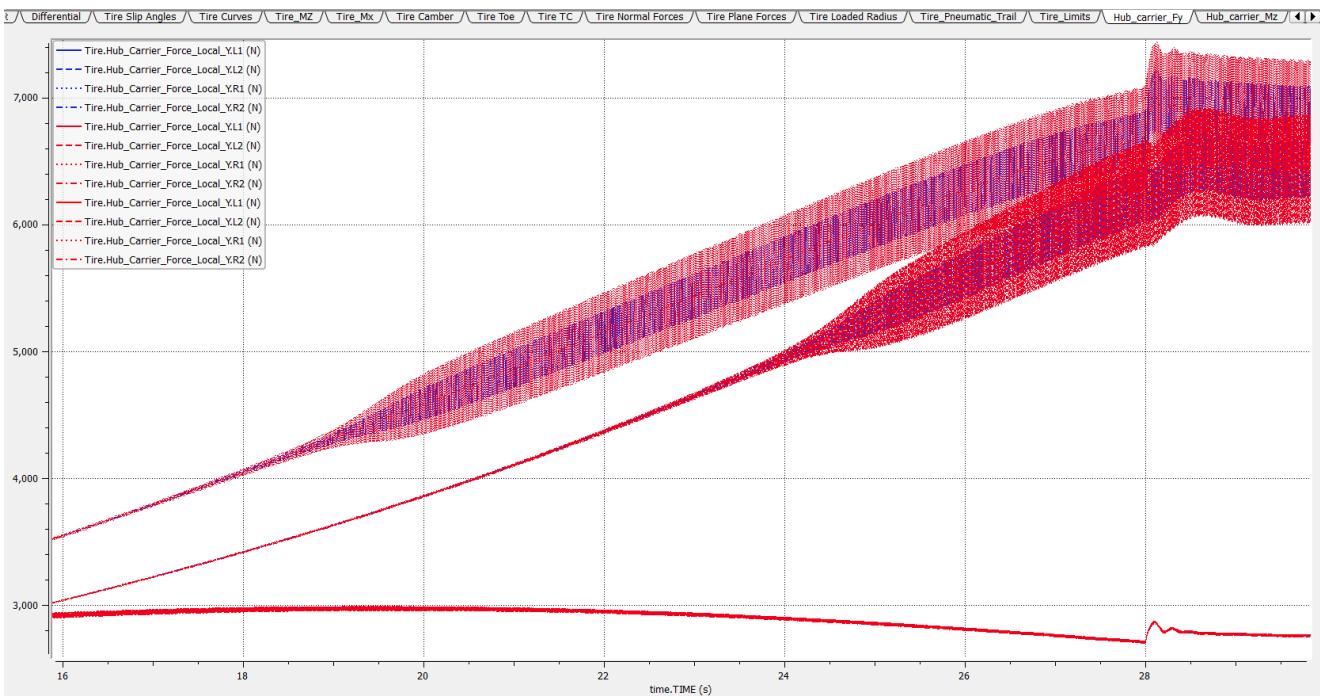
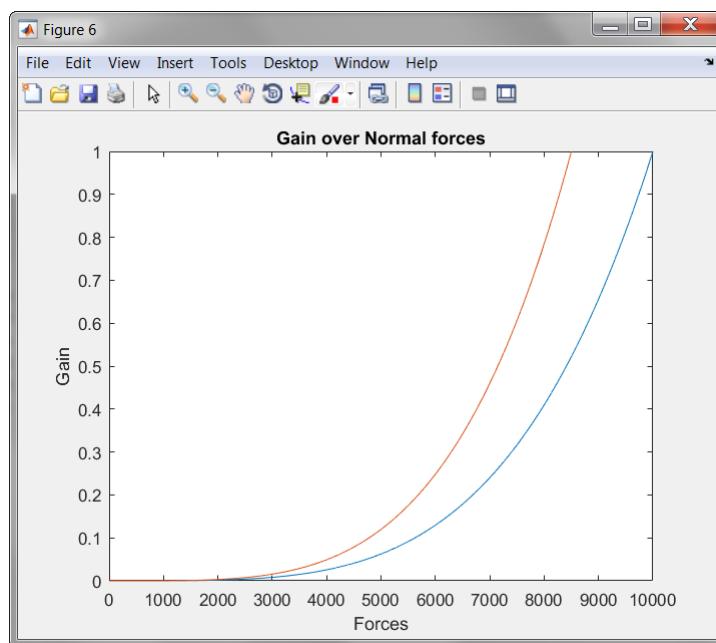
With the Smoothness_LS parameter, the user can tune the slope of the gain in the same way illustrated for the lateral margin.

In general, the gain over longitudinal slip has a narrow-to-none operating window if the car model is including a restrictive TCS logic.

Gain Over Tire Loads

Tire Scrub Effect activation is also dependent on the normal load of the tires: this dependency has been included to tune the tool with every possible vehicle weight and weight distribution. The relative gain is computed using a forth power function over tire loads, considering that, in cornering maneuvers, the inner tires are less stressed, thus the scrub effect is lower. The user can set a value for the NF_M parameter, specifying the tire load (measured in Newton) threshold corresponding to a gain value of 1. The isolated effect of this parameter is explained by the following pictures.

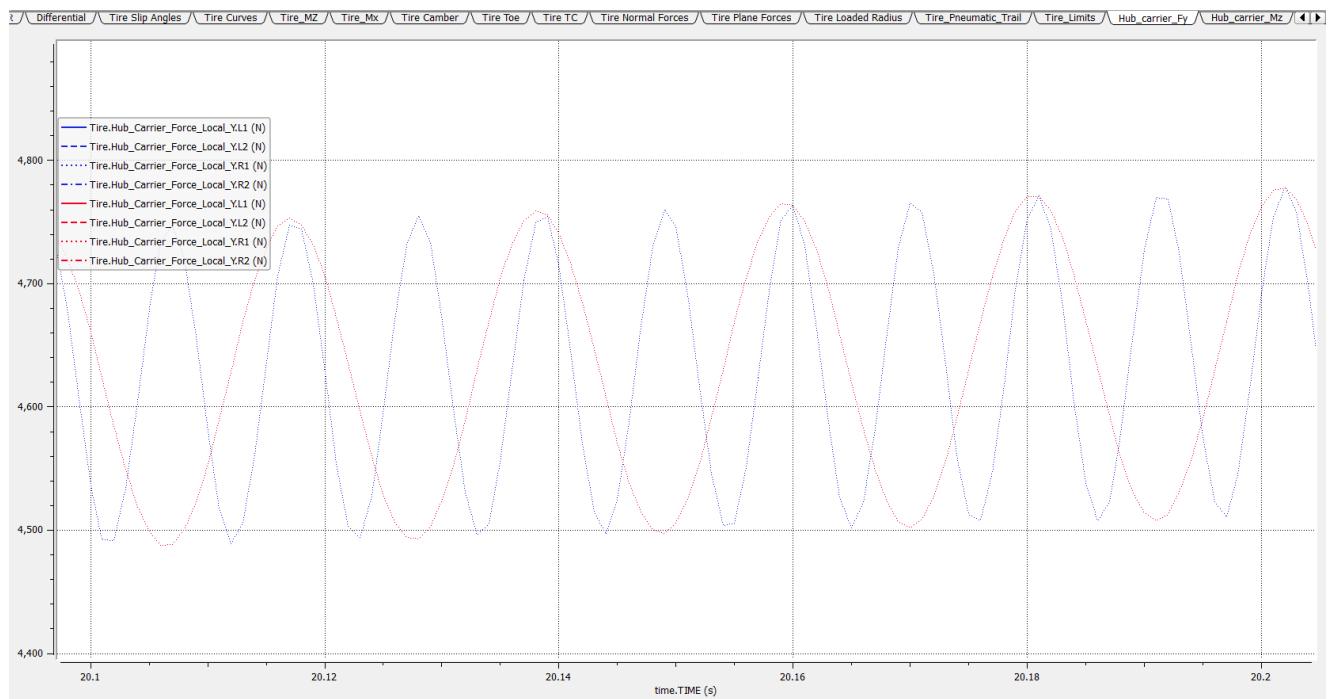
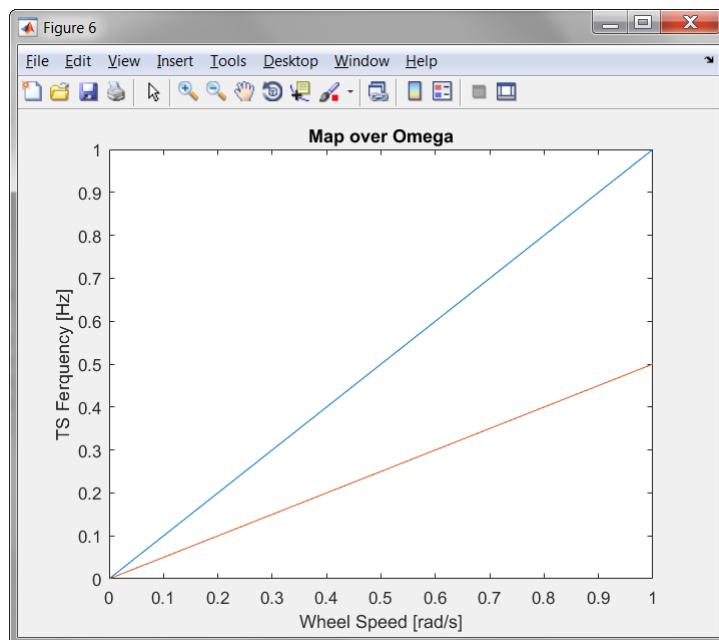
Case	Parameter Value	Analysis Colour
Baseline	NF_M =10000[N]	Blue
Mod	NF_M =8500[N]	Red



Mapping over Wheel Omega

This gain is meant to stretch the sinusoidal disturbance in its frequency. The user is able to adjust the mapping for the disturbance frequency with respect to rotational speed of the wheels. Values for ω_{\max} parameter are inversely proportional to the frequency of the Tire Scrub Effect, as shown by the comparison of the following plots.

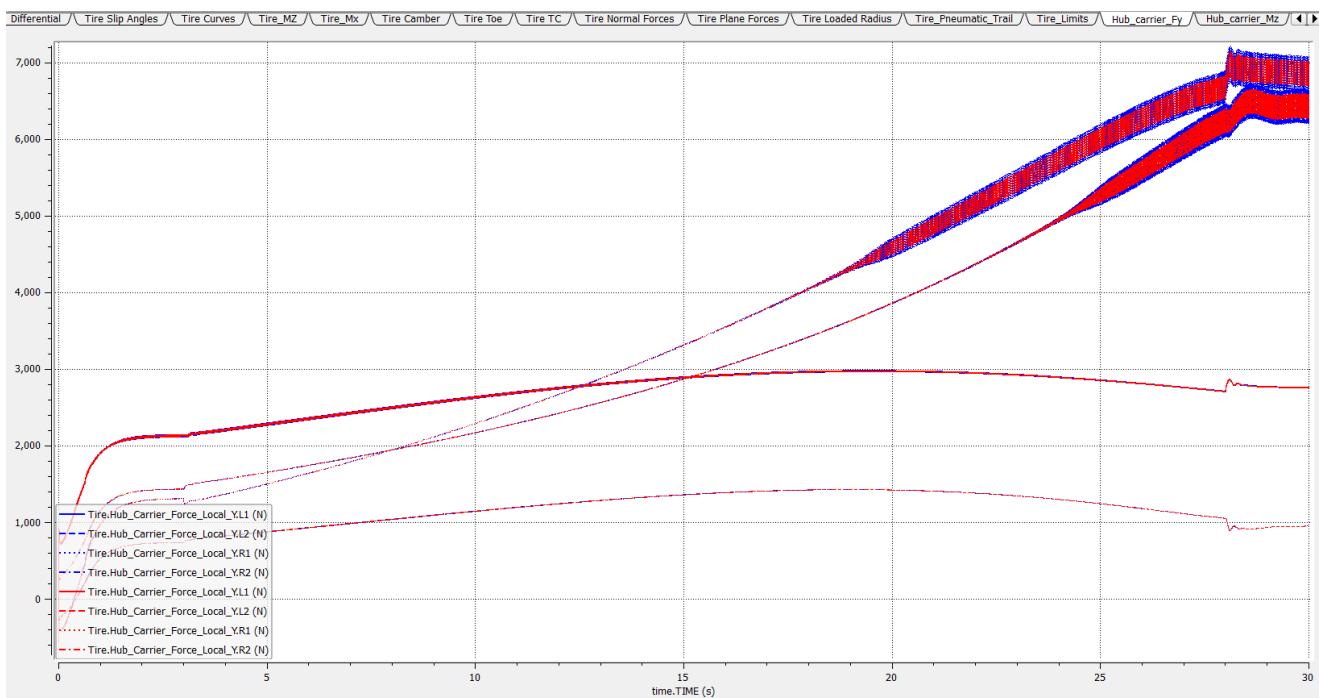
Case	Parameter Value	Analysis Colour
Baseline	$\omega_{\max}=1$	Blue
Mod	$\omega_{\max}=2$	Red



Global Gain

Finally, the user can adjust the global scaling of the tire scrub effect, directly proportional to the amplitude of the sinusoidal disturbance

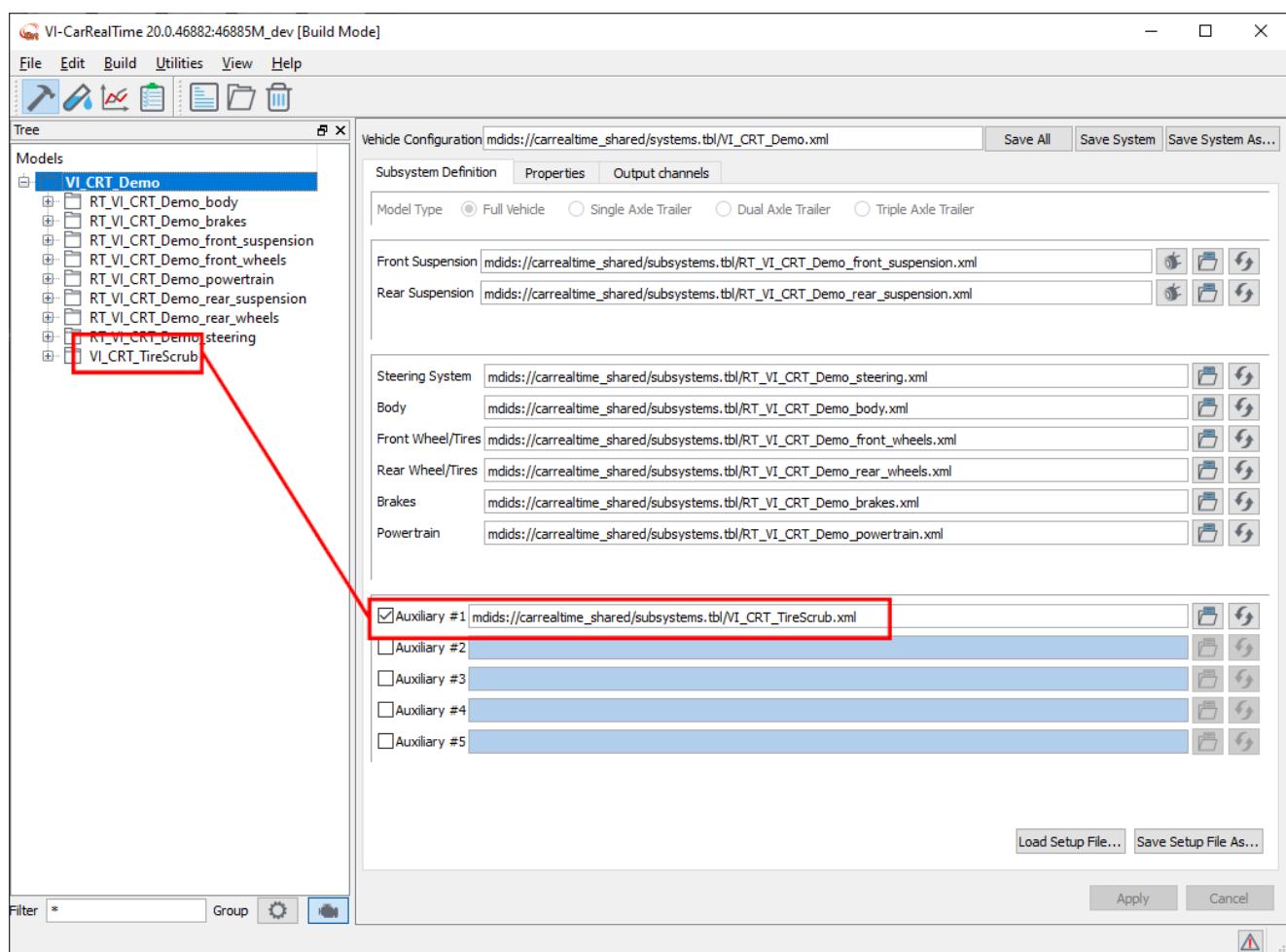
Case	Parameter Value	Analysis Colour
Baseline	G=1.7	Blue
Mod	G=1.2	Red



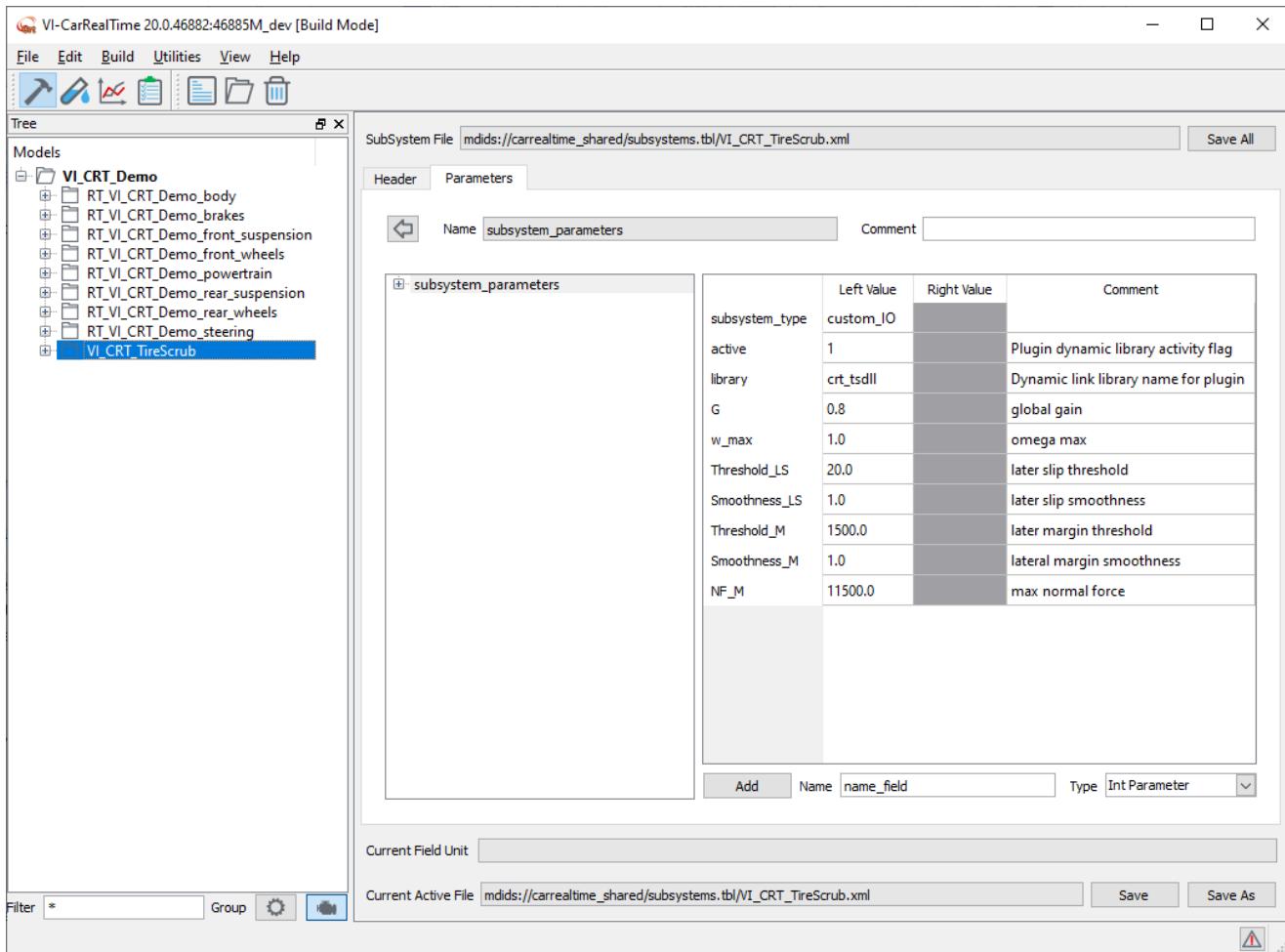
Auxiliary Subsystem

The Tire Scrub Effect is implemented in VI-CarRealTime with an auxiliary subsystem linked to the relative library. Whether the feature is used for offline simulation on Windows, or it is used on CCUR machine, the auxiliary subsystem must be added to the car model, so that the corresponding property block is translated into the send file.

From the Subsystem Definition tab, the Auxiliary flag must be switched on and the VI_CRT_TireScrub.xml subsystem must be called.



VI-CarRealTime

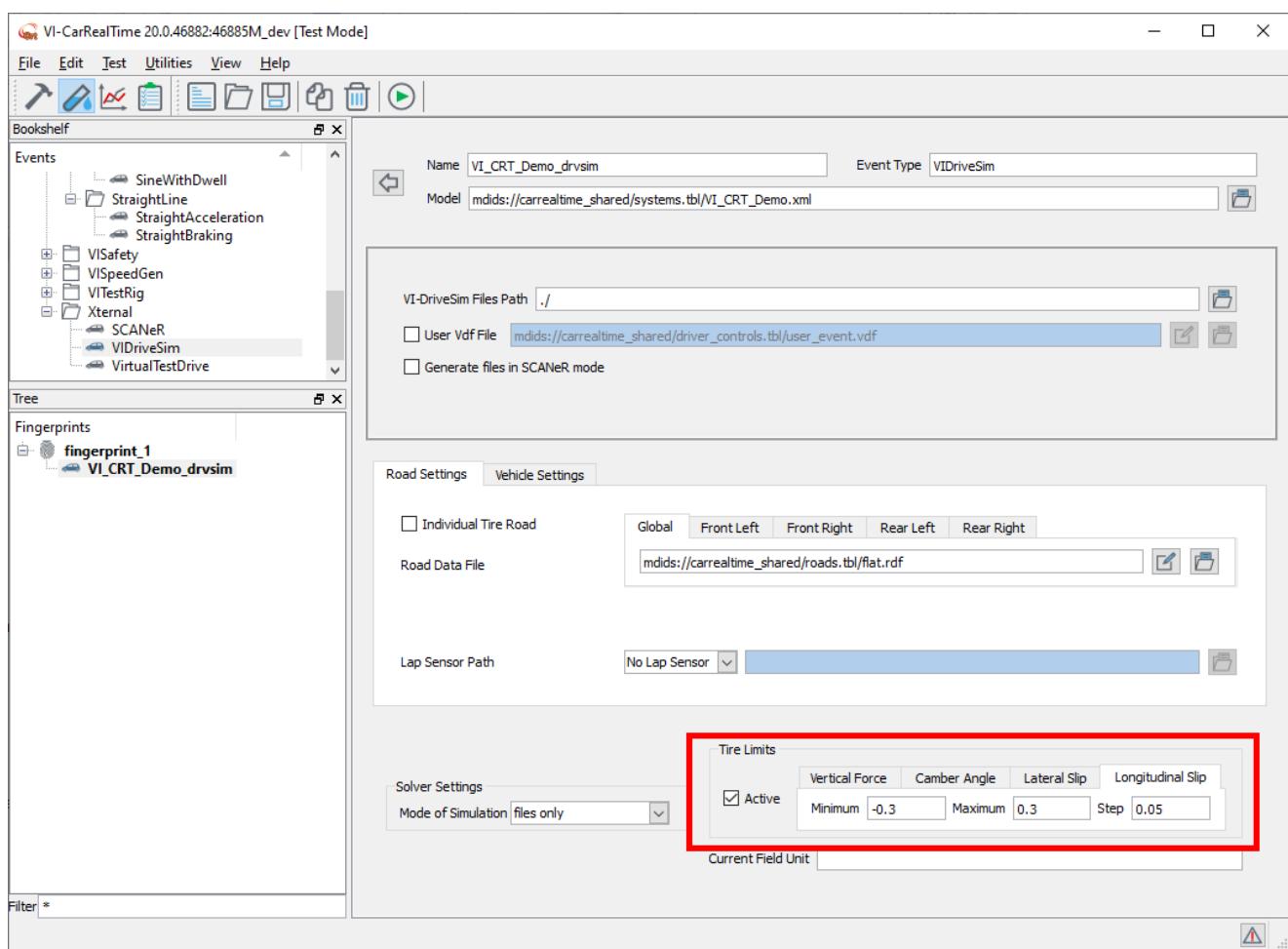


The tuning parameters can then be adjusted from VI-CarRealTime GUI and they will be treated as all other model properties. For both offline and online usage, the plugin files must be present in the solver directories:

Ambient	Plugin	Directory
Windows	crt_tsdll.dll	C:\Program Files\VI-grade\VI-CarRealTime 20\acarrt\win64
Linux	libcrt_tsdll.so	/home/vigrade/vicrt/standalone

Tire Limits Activation

The next step to enable Tire Scrub Effect is to activate VI-TireLimits calculation for VI-CarRealTime solver. When generating a send file for a VI-CarRealTime Xternal event, the relative flag must be activated from the VI-CarRealTime GUI and boundary values must be set accordingly to the tire file in use.



The parameters in the GUI must not exceed the maximum values defined in the tire property file, otherwise the TireLimits tool will fail in calculating the maps.

```
$-----long_slip_range
[LONG_SLIP_RANGE]
KPUMIN = -1.5 $Minimum valid wheel slip
KPUMAX = 1.5 $Maximum valid wheel slip
$-----slip_angle_range
[SLIP_ANGLE_RANGE]
ALPMIN = -1.5708 $Minimum valid slip angle
ALPMAX = 1.5708 $Maximum valid slip angle
$-----inclination_slip_range
[INCLINATION_ANGLE_RANGE]
CAMMIN = -0.26181 $Minimum valid camber angle
CAMMAX = 0.26181 $Maximum valid camber angle
$-----vertical_force_range
[VERTICAL_FORCE_RANGE]
FZMIN = 225 $Minimum allowed wheel load
FZMAX = 10125 $Maximum allowed wheel load
```

Offline and online, at the beginning of the simulation the tool will compute a map of limit states for the specified ranges of values, saving the envelopes in the RAM memory of the PC, ready to be used by VI-CarRealTime solver. Reducing the step specified for each parameter, means higher resolution of the maps generated to compute TireLimits, increasing the initialization time while starting the simulation.

When running the simulation on CCUR machine with TireLimits enabled, an increase of 100÷150 µs in computation time should be expected for the "vicrt" executable. It is therefore necessary for this executable to be run on a strictly dedicated CPU.

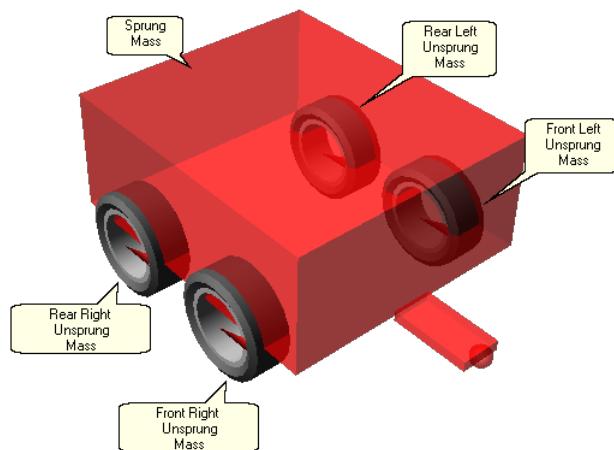
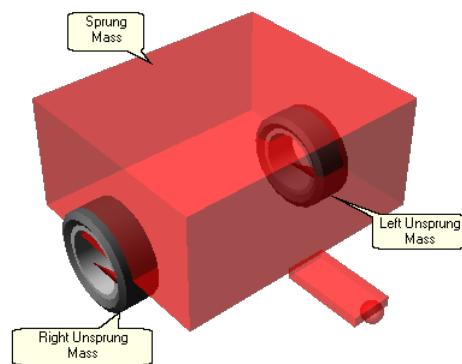
VI-CarRealTime

When trying the feature for the first time with a new car model or a new tire model, it is suggested to run the simulation with the motion platform not engaged, checking for eventual overruns. Furthermore, it is suggested to run the platform in DataCheck mode, double checking that the outputs of the motion cueing are as intended. In general, it can be convenient to use the Tire Scrub Effect offline plugin, and the offline version of VI-MotionCueing, to fine tune the parameters before using them on the driving simulator.

Trailer Model

VI-CarRealTime supports two simplified model of trailers:

- **single axle**
- **double axle**



Conventions and references

Trailer Location

The trailer model is positioned with respect to the global frame, having the hitch ball center in global x and y origin (the z value is not considered).

Trailer Orientation

The trailer is positioned having the global X+ axis pointing forward in the direction of motion (forward).

See [Vehicle Model Conventions and References](#) for further info about reference systems.

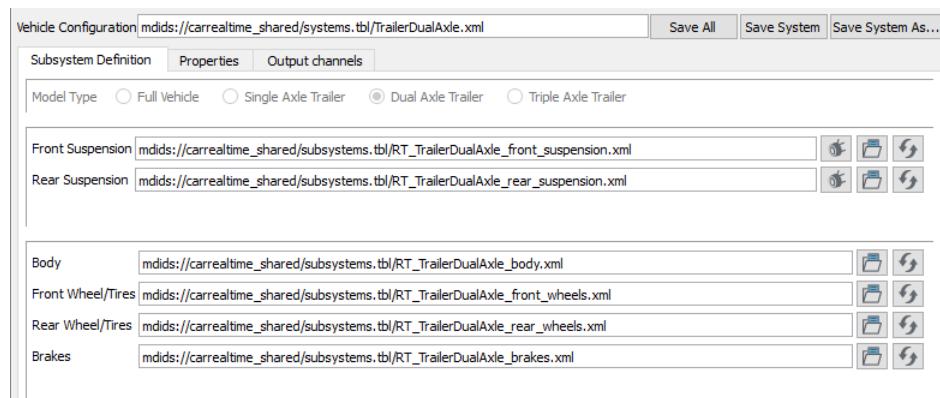
Model Configuration

The definition of the trailer model in VI-CarRealTime is realized using the system files, having xml format and stored in the database under `system.tbl` table.

The system file contains the following set of information:

- Units
- List of trailer subsystem files:
 - [Front Suspension](#)
 - [Rear Suspension](#)
 - [Body](#)
 - [Front Wheels and Tires](#)
 - [Rear Wheels and Tires](#)
 - [Brakes](#)

The property editor can be used to select appropriate property file for each of the subsystems.



It is possible to exchange subsystems among different trailer models by selecting them in the system property editor. There is no need to save the system file since the changes done to trailer composition (using the **Apply** push button) will be kept in memory during the complete VI-CarRealTime session.

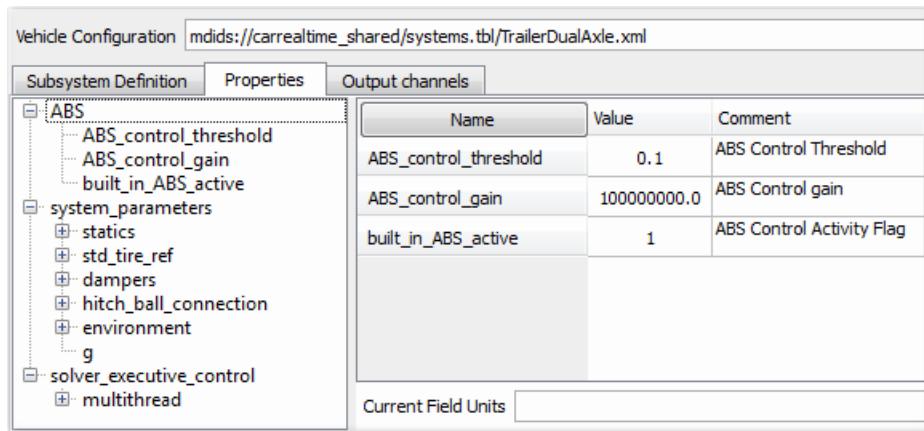
Subsystem files can be loaded into trailer model using the button, reloaded from file using the button.

Suspension subsystems can be created/edited using VI-SuspensionGen through button.

- [Trailer properties](#)
- [Output channels](#)

Properties

Trailer system files contain a set of information beyond the list of the subsystems. The auxiliary information stored in system files can be edited using property editor.



The treeview for system properties include the following sections:

- [ABS](#)
- [System Parameters](#)
- [Solver Executive Control](#)

ABS

The parameters included in this section are used to setup the control systems embedded in VI-CarRealTime model.

Embedded ABS system (ABS)

- **ABS_control_threshold**
longitudinal slip threshold for ABS engagement.
- **ABS_control_gain**
ABS controller proportional gain.
- **Built_in_ABS_active**
ABS controller activity flag (1: active; 0: inactive).
- **low_speed_threshold**
threshold vehicle speed below which the ABS system is switched off.

System Parameters

The system parameters section contains parameters mainly related to model setup, components, and general system settings.

- [Statics](#)
- [Std_tir_ref](#)
- [Dampers](#)
- [Hitch Ball Connection](#)
- [Environment](#)

g: gravity field value (m/s^2)

The Statics Parameters are:

- **statics_abort_time**
Abort simulation if static equilibrium is not achieved within this time.
- **statics_trans_speed_threshold**
Static equilibrium is not achieved until the norm of the system translational speeds is below this value.
- **statics_trans_accel_threshold**
Static equilibrium is not achieved until the norm of the system translational accelerations is below this value.
- **statics_angular_speed_threshold**
Static equilibrium is not achieved until the norm of the system angular speeds is below this value.
- **statics_angular_accel_threshold**
Static equilibrium is not achieved until the norm of the system angular accelerations is below this value.
- **statics_use_brakes**
Flag used to activate brakes activity during statics.
- **statics_use_linear_damper**
Flag used to switch to linear damper in statics. Useful to increase convergence rate of the statics.
- **statics_linear_damper_rate_[front/rear]**
damper rate used if statics_use_linear_damper flag is set to 1.
- **statics_use_linear_spring**
Flag used to switch to linear springs in statics.
- **statics_linear_spring_rate_[front/rear]**
spring rate used if statics_use_linear_spring flag is set to 1.
- **statics_brake_damping**
allows the user to define a rotational damping value to be applied in wheel rotational degree of freedom during static analysis.
- **statics_integration_time_step**
integration time step to be used for static/setup analysis only (non positive value to inherit from dynamic event).

Note: When using MATLAB/Simulink to replace built in springs or damper components, internal property files have to be replaced with zero force characteristics ones (i.e. `vi_null_spring.xml` , `vi_null_damper.xml` from carrealtime_shared database). Since external inputs coming from Simulink are not applied during statics it may cause convergence problems in case of springs. Enabling the `statics_use_linear_spring` flag can solve the issue.

STD_TIR_REF parameters are:

- **std_tire_ref_*(X,Y,Z,PSI,THETA,PHI)**
tire/road reference marker absolute position and orientation.

Dampers parameters are:

- **damper_lp_filter**
value used to set the activity of the filter acting on the main dampers. When this value is set to 0 the filter is inactive. For any other positive value the cutoff frequency of the filter is given by: $1/(output_step * damper_lp_filter)$.

The Hitch Ball Connection parameters are:

- **rotational_kp_[x, y, z]**
Proportional Gain for chassis rotational stiffness in x, y, z direction lock PID.

- **rotational_ki_[x, y, z]**
Integral Gain for chassis rotational stiffness in x, y, z direction lock PID.
- **rotational_kd_[x, y, z]**
Derivative Gain for chassis rotational stiffness in x, y, z direction lock PID.
- **translational_kp_[x, y, z]**
Proportional Gain for chassis translational stiffness in x, y, z direction lock PID.
- **translational_ki_[x, y, z]**
Integral Gain for chassis translational stiffness in x, y, z direction lock PID.
- **translational_kd_[x, y, z]**
Derivative Gain for chassis translational stiffness in x, y, z direction lock PID.

Environment parameters are:

- **wind_direction_angle**
sets the wind direction angle (AZ) in global reference system.

Solver Executive Control

Solver Execute Control parameters tree include settings belonging to the solver calculation:

Multithread subtree:

- **tire_calc_active_flag**
enables multi-thread calculation for tires. Currently this option affects only F-Tire model and requires a specific F-Tire license.

Output Channels

Output channels can be completely customized in VI-CarRealTime. Each system file contains the reference to an output mapping file (.xml). The file contains the list of all the outputs created by the solver.

When the vehicle model is modified (for example adding more sensors or introducing engine part), also the output channels update accordingly. The Output Channels Map file is not updated automatically, but it is possible to synchronize it to the model using the Update Map button. This procedure overwrite the selected files adding the missing channels (or removing obsolete ones) and maintains the settings of the existing valid channels.

Property File: mdids://carrealtime_shared//output_maps.tbl/VI_CRT_Demo_output_map.xml

The screenshot shows the 'Output_Channels' section of the configuration interface. On the left, a tree view lists various channels under categories like Animator_Widget and aero_forces. On the right, a table displays the configuration for each selected channel:

	Active	Result Set	Component	Scale
aero_forces_aero_balance	yes	aero_forces	aero_balance	1.0
aero_forces_center_downforce	yes	aero_forces	center_downforce	1.0
aero_forces_center_sideforce	yes	aero_forces	center_sideforce	1.0
aero_forces_drag_arm	yes	aero_forces	drag_arm	1.0
aero_forces_drag_force	yes	aero_forces	drag_force	1.0
aero_forces_drag_force_front	yes	aero_forces	drag_force_front	1.0
aero_forces_drag_force_rear	yes	aero_forces	drag_force_rear	1.0
aero_forces_drag_torque	yes	aero_forces	drag_torque	1.0
aero_forces_front_downforce	yes	aero_forces	front_downforce	1.0
aero_forces_front_sideforce	yes	aero_forces	front_sideforce	1.0
aero_forces_height_auxiliary	yes	aero_forces	height_auxiliary	1.0
aero_forces_height_front	yes	aero_forces	height_front	1.0
aero_forces_height_rear	yes	aero_forces	height_rear	1.0
aero_forces_lateral_velocity	yes	aero_forces	lateral_velocity	1.0
aero_forces_longitudinal_velocity	yes	aero_forces	longitudinal_velocity	1.0
aero_forces_pitch	yes	aero_forces	pitch	1.0
aero_forces_rear_downforce	yes	aero_forces	rear_downforce	1.0
aero_forces_rear_sideforce	yes	aero_forces	rear_sideforce	1.0
aero_forces_roll	yes	aero_forces	roll	1.0
aero_forces_roll_torque	yes	aero_forces	roll_torque	1.0
aero_forces_side_slip	yes	aero_forces	side_slip	1.0

For each of the available channels it is possible to customize:

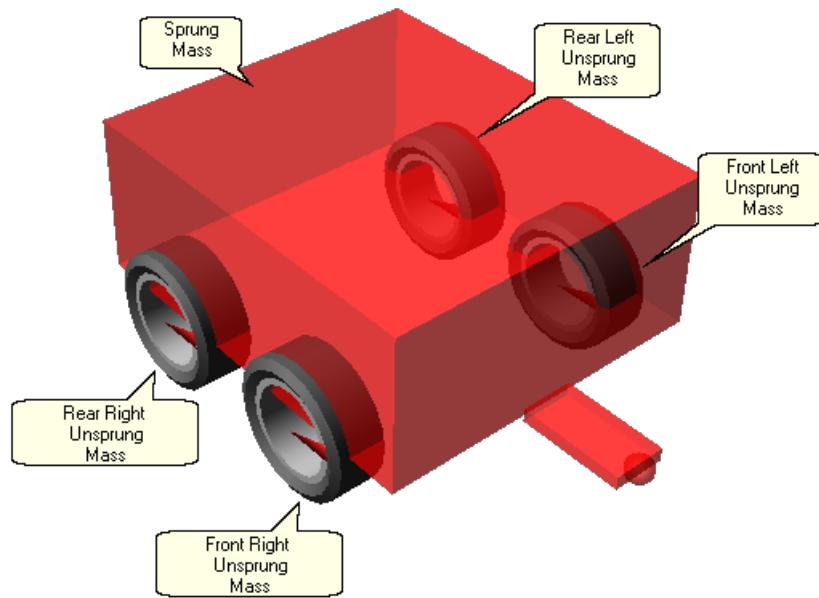
- **Activity**
- **Result Set Name**
- **Component Name**
- **Scale Factor**
- **Offset**
- **Live Animation** (setting this flag to "yes" will make the selected channel available during live animation).

The Output Channels File is stored into the database in output_maps.tbl table.

NOTE: Scale Factor and Offset do not affect output data when using the Matlab/Simulink Interface.

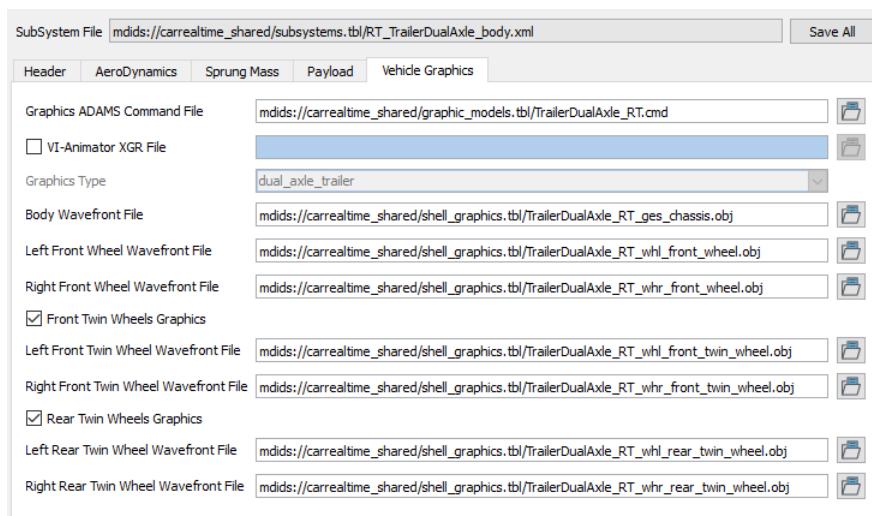
Body Subsystem

The body subsystem properties collect mass, inertia, setup and accessory information about the sprung mass part of VI-CarRealTime trailer model.

[Vehicle Graphics](#)[Aerodynamics](#)[Sprung Mass](#)[Payload](#)[Adjustable Weight Distribution Hitch](#)[Trailer Sway Control](#)

Vehicle Graphics

The Trailer Graphics Property Editor allows the user to refer to an external graphic file (or files) in order to define the graphics of the vehicle. Both an Adams command file (cmd) and a list of waveform files are supported for the graphic.



The fields of this panel are:

- **Graphics ADAMS Command File**

Path of Adams command file containing the graphic model for Adams/PPT animations.

When **VI-Animator XGR File** toggle button is checked:

- **VI-Animator XGR File**

This field allows the user to choose the Graphics File (XGR) embedding all the graphics of the parts.

When **VI-Animator XGR File** toggle button is unchecked:

- **Graphics Type**

Type of graphics allowed.

- **Body Wavefront File**

This field allows the user to set the waveform graphics file of the body of the trailer.

- **Left Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front left wheel of the trailer.

- **Right Front Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front right wheel of the trailer.

When **Front Twin Wheels Graphics** toggle button is checked the following fields are enabled:

- **Left Front Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front left twin wheel of the trailer.

- **Right Front Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the front right twin wheel of the trailer.

When **Rear Twin Wheels Graphics** toggle button is checked the following fields are enabled:

- **Left Rear Twin Wheel Wavefront File**

This field allows the user to set the waveform graphics file of the rear left twin wheel of the trailer.

- **Right Rear Twin Wheel Wavefront File**

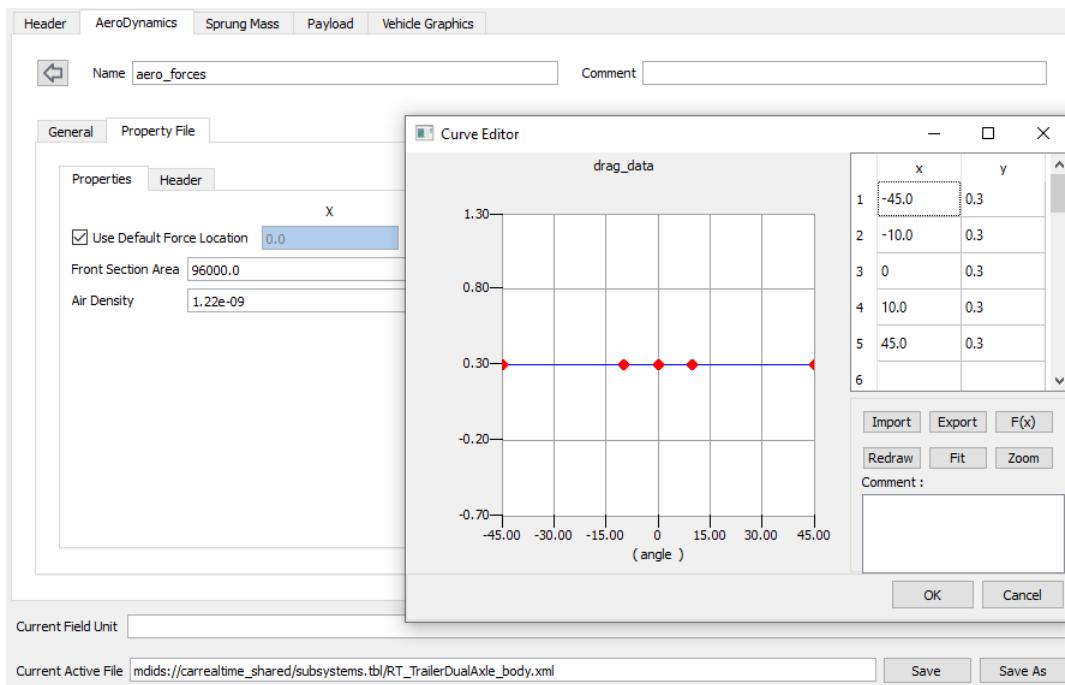
This field allows the user to set the waveform graphics file of the rear right twin wheel of the trailer.

Aerodynamics

The aerodynamic for the trailer is a drag force applied on a point that can be defined by the user.

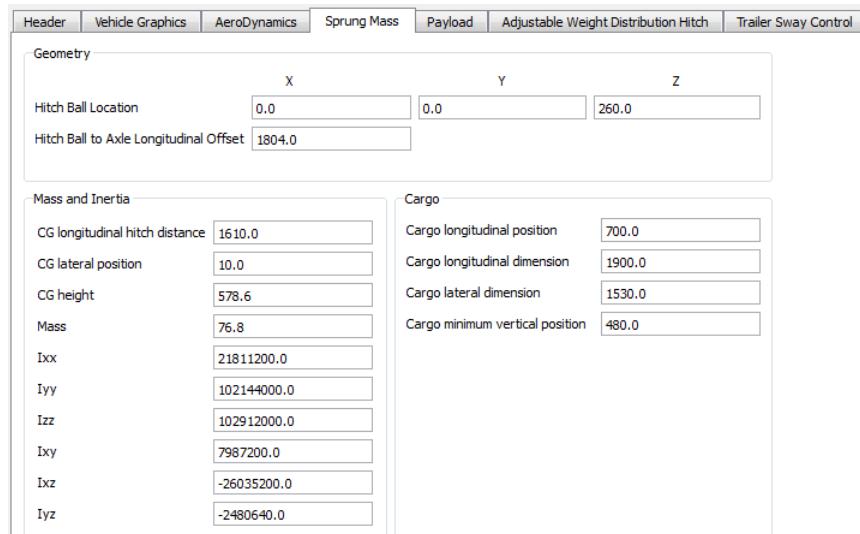
It's calculated using a drag coefficient table, a **Front Section Area** interested by the drag force and the air density value. The Drag Coefficient table can be edited by clicking on the **Plot/Edit Drag Coefficient vs Relative Wind Incidence** button.

Location is expressed in the trailer [Global Reference](#) frame.



Sprung Mass

The sprung mass Property Editor gives access to the sprung mass inertia property of trailer chassis. It also contains the reference to the Adams command file for trailer model animation in Adams/PPT, the vehicle wheelbase and the hitch ball data.



Geometry

Hitch Ball Location

Location of the hitch ball with respect to the [hitch_ball_loc_\[X, Y, Z\]](#) trailer parameters set in the vehicle system properties.

X positive in [Global Reference](#) X direction.

Hitch Ball - Axle Offset

Distance between the hitch ball and the front axle in [Global Reference](#) X direction.

Wheelbase

Distance between front and rear wheel center in [Global Reference](#) X direction.

Mass and Inertia

The container is used to set mass and inertia properties of trailer sprung mass.

Note: since the trailer model is conceptual, the sprung mass part is not the same thing as the body chassis part, since a portion of the suspension masses and inertia is seen as belonging to the sprung part.

CG longitudinal hitch distance

Longitudinal distance from hitch ball x center and sprung mass CG.

CG lateral position

Lateral position of sprung mass CG. It is expressed in [Global Reference](#) system.

CG height

Vertical position of sprung mass CG. It is expressed in [Global Reference](#) system.

Mass

Mass of sprung part.

I_{xx}

I_{xx} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yy}

I_{yy} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{zz}

I_{zz} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{xy}

I_{xy} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{xz}

I_{xz} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yz}

I_{yz} moment of inertia of sprung part. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

Cargo

The container is used to specify the cargo dimensions and location.

Cargo longitudinal position

Longitudinal distance from hitch ball x center and the beginning of cargo box.

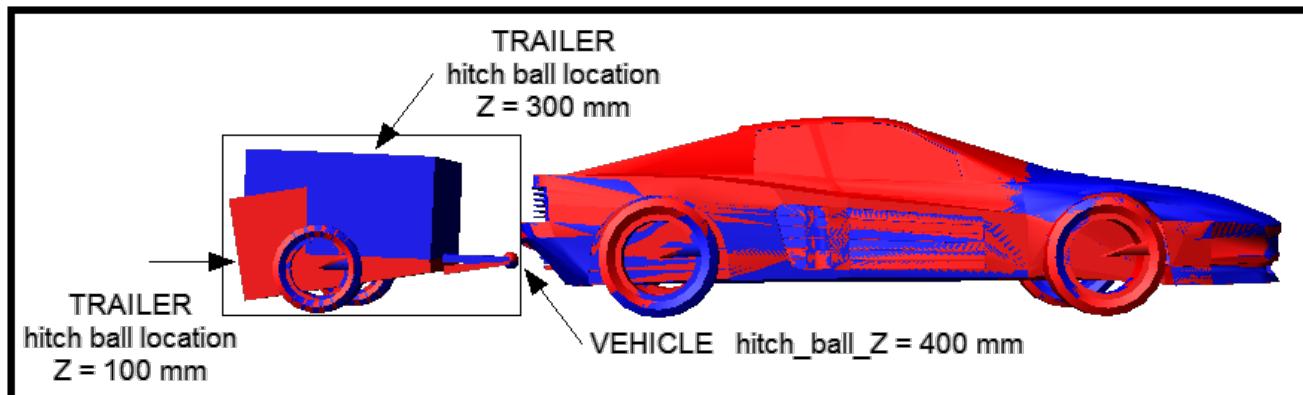
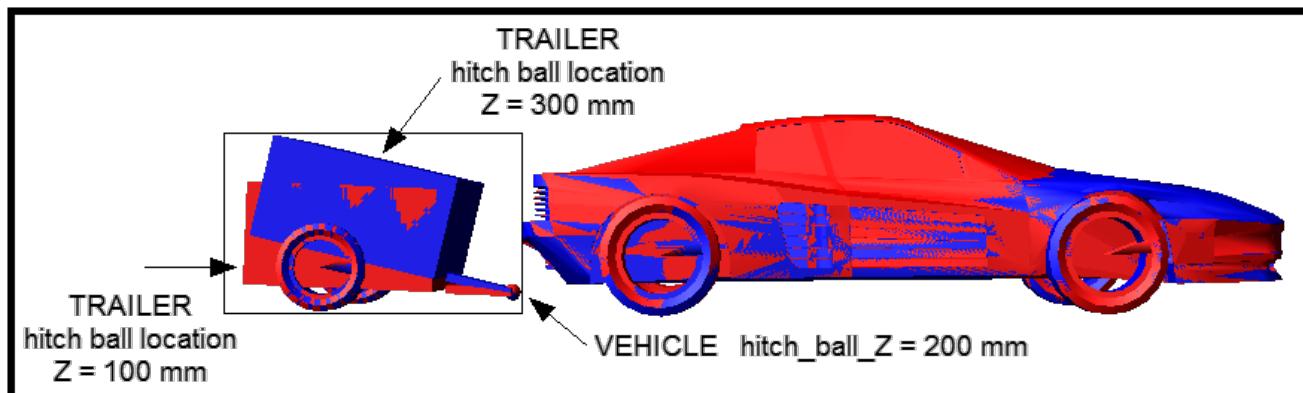
Cargo longitudinal / lateral dimension

Absolute dimensions of cargo footprint.

Cargo minimum vertical position

Vertical position of cargo floor, expressed in [Global Reference](#) system.

To better understand the differences in setting z values for the [hitch-ball on the vehicle](#) or in the trailer, please look at the following image.

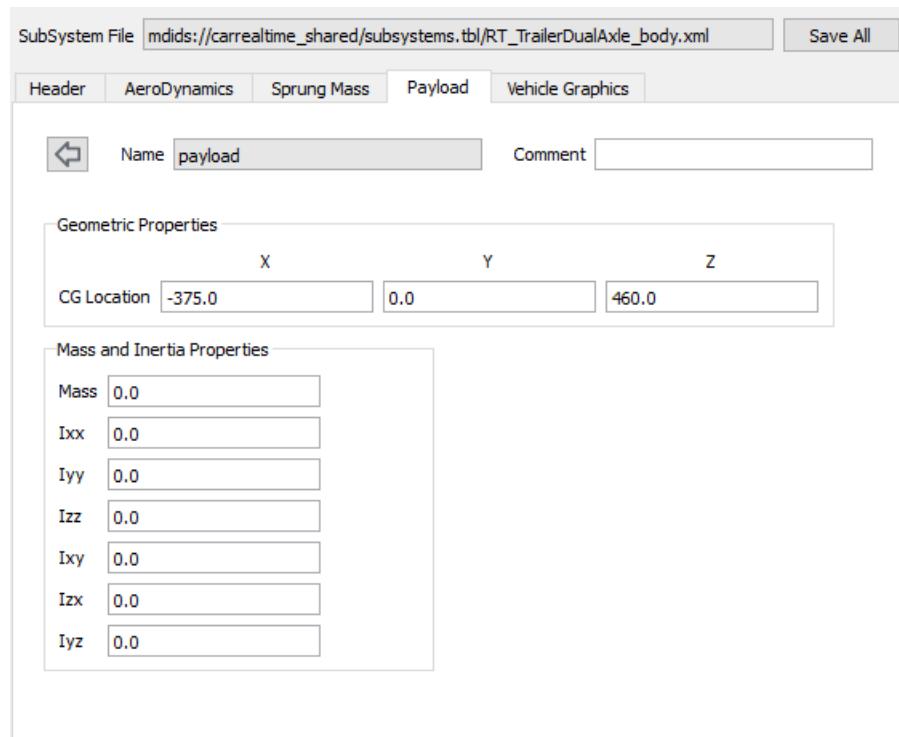


Payload

The Payload panel lets you add some payload points to the sprung mass.

Setting their CG w.r.t. the trailer and their inertia properties, different trailer loaded configurations can be analyzed.

Note: payloads are fixed with respect to the trailer reference system during all the simulation.



Click on the Display Table View button  to see the table resuming all the payloads created: additional payloads can be added simply **editing a name** for the new payload and then selecting the **Add** button.

Header	Vehicle Graphics	AeroDynamics	Sprung Mass	Payload
Name Filter *				
Name	Comment			
payload				
payload1				
payload2				

Name Type

Geometric Properties

CG Location

Location of the payload with respect to the **trailer** reference system.

Mass and Inertia

Mass

Mass of payload.

I_{xx}

I_{xx} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yy}

I_{yy} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{zz}

I_{zz} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{xy}

I_{xy} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{xz}

I_{xz} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

I_{yz}

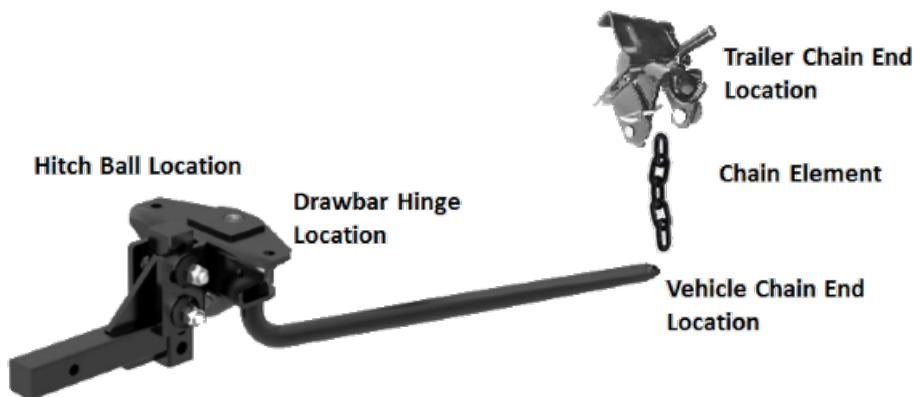
I_{yz} moment of inertia of payload. It is expressed in a reference frame having origin coincident with CG of the part and oriented as [Global Reference](#) system.

Adjustable Weight Distribution Hitch

Weight distribution systems use spring bars to help resolve the problems that often occur with standard hitch systems. Adding spring bars applies a leverage to either side of the system, which results in a transfer of the load that is pushing down on the rear of your vehicle to all of the axles on both your tow-vehicle and your trailer.

VI-CarRealTime

Header	Vehicle Graphics	AeroDynamics	Sprung Mass	Payload	Adjustable Weight Distribution Hitch	Trailer Sway Control																												
<input checked="" type="checkbox"/> Active FALR <input type="text" value="100.0"/> Chain Stiffness <input type="text" value="1000.0"/> Chain Damping <input type="text" value="0.1"/>																																		
Geometry <table border="1"> <thead> <tr> <th></th> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <td>Drawbar Hinge Left Location</td> <td>200.0</td> <td>200.0</td> <td>240.0</td> </tr> <tr> <td>Drawbar Hinge Right Location</td> <td>200.0</td> <td>-200.0</td> <td>240.0</td> </tr> <tr> <td>Vehicle Chain End Left Location</td> <td>-400.0</td> <td>200.0</td> <td>220.0</td> </tr> <tr> <td>Vehicle Chain End Right Location</td> <td>-400.0</td> <td>-200.0</td> <td>220.0</td> </tr> <tr> <td>Trailer Chain End Left Location</td> <td>-400.0</td> <td>200.0</td> <td>400.0</td> </tr> <tr> <td>Trailer Chain End Right Location</td> <td>-400.0</td> <td>-200.0</td> <td>400.0</td> </tr> </tbody> </table>								X	Y	Z	Drawbar Hinge Left Location	200.0	200.0	240.0	Drawbar Hinge Right Location	200.0	-200.0	240.0	Vehicle Chain End Left Location	-400.0	200.0	220.0	Vehicle Chain End Right Location	-400.0	-200.0	220.0	Trailer Chain End Left Location	-400.0	200.0	400.0	Trailer Chain End Right Location	-400.0	-200.0	400.0
	X	Y	Z																															
Drawbar Hinge Left Location	200.0	200.0	240.0																															
Drawbar Hinge Right Location	200.0	-200.0	240.0																															
Vehicle Chain End Left Location	-400.0	200.0	220.0																															
Vehicle Chain End Right Location	-400.0	-200.0	220.0																															
Trailer Chain End Left Location	-400.0	200.0	400.0																															
Trailer Chain End Right Location	-400.0	-200.0	400.0																															

**Geometric Properties**

Following distance are defined w.r.t. hitch ball position and expressed according [Global Reference](#) system.

Draw Bar Hinge Location

Location of weight distribution bar attachment on tow-vehicle hitch side.

Vehicle Chain End location

Location of chain attachment to weight distribution bar.

Trailer Chain End location

Location of chain attachment on trailer.

Element Properties**FARL %**

FALR is the change in front axle load due to weight distributing hitch application divided by change in front axle load due to addition of trailer tongue weight, expressed in percent.

$$FARL = 100 * \frac{(W_{fCoupWD} - W_{fCoup})}{(W_{fUncoup} - W_{fCoup})}$$

Where:

$W_{fUncoup}$ is front axle weight of the uncoupled tow vehicle (N)

W_{fCoup} is front axle weight of the coupled tow vehicle (N) without weight distribution.

W_{fCoup_WD} is front axle weight of the uncoupled tow vehicle (N) with weight distribution.

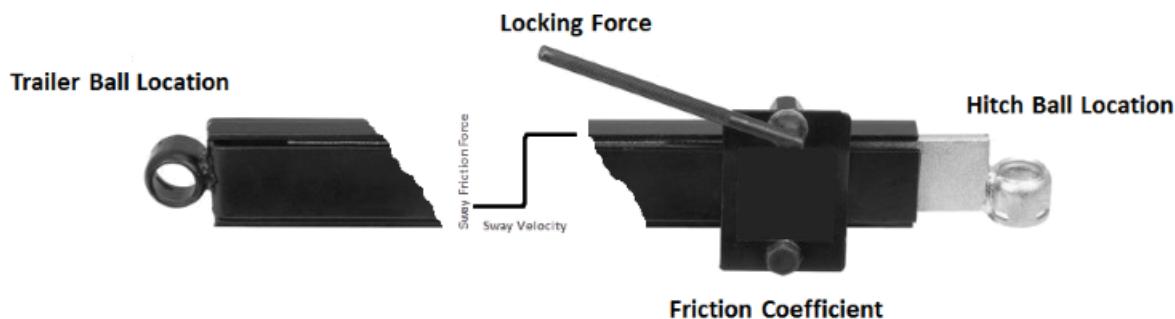
Chain Stiffness / Damping

Chain elasticity parameters.

Trailer Sway Control

Trailer Sway Control let you add a frictional damper, in order to reduce trailer sway effects.

Header	Vehicle Graphics	AeroDynamics	Sprung Mass	Payload	Adjustable Weight Distribution Hitch	Trailer Sway Control																				
<input checked="" type="checkbox"/> Active Parameters <table border="1"> <tr> <th></th> <th>X</th> <th>Y</th> <th>Z</th> </tr> <tr> <td>Hitch Ball Location</td> <td>0.0</td> <td>500.0</td> <td>260.0</td> </tr> <tr> <td>Trailer Ball Location</td> <td>-300.0</td> <td>200.0</td> <td>260.0</td> </tr> <tr> <td>Locking Force</td> <td>1000.0</td> <td></td> <td></td> </tr> <tr> <td>Friction Coefficient</td> <td>0.5</td> <td></td> <td></td> </tr> </table>								X	Y	Z	Hitch Ball Location	0.0	500.0	260.0	Trailer Ball Location	-300.0	200.0	260.0	Locking Force	1000.0			Friction Coefficient	0.5		
	X	Y	Z																							
Hitch Ball Location	0.0	500.0	260.0																							
Trailer Ball Location	-300.0	200.0	260.0																							
Locking Force	1000.0																									
Friction Coefficient	0.5																									



Geometric Properties

Following distance are defined w.r.t. hitch ball position and expressed according [Global Reference](#) system.

Trailer Ball Location

Location of sway attachment on trailer side.

Hitch Ball Location

Location of sway attachment on tow-vehicle side.

Element Properties

Locking Force

Normal preload force applied to sway device.

Friction Coefficient

Dynamic friction coefficient of sway.

Resultant friction force will be so computed:

$$F = \frac{V * \chi}{\sqrt{1 + (\frac{V * \chi}{N * \mu})^2}}$$

Where

V is the sway velocity

c is the slope of friction force vs velocity near static condition ($\sim 1e8$ N*s/m)

N is the sway locking force

m is the friction coefficient

Suspension Subsystem

The trailer suspension subsystem is similar to the [vehicle model suspension subsystem](#).

The main difference is that in the trailer model there is no access to suspension setup sheet.

[Wheel location](#)

[Wheel Orientation](#)

[Compliance](#)

[Springs](#)

[Dampers](#)

[Bumpers](#)

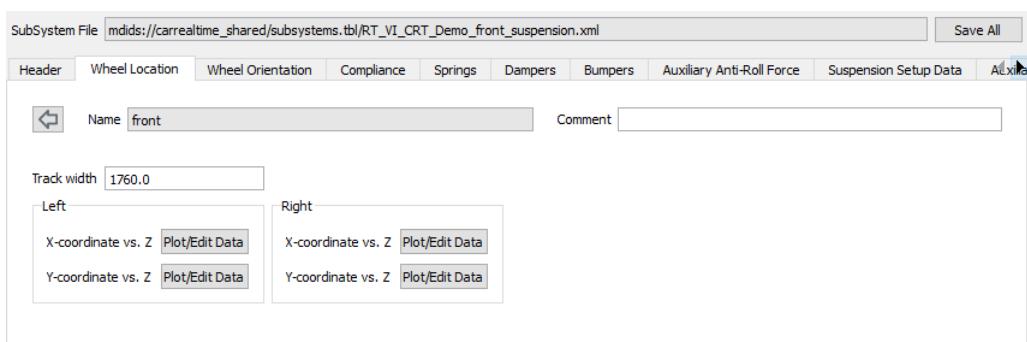
[Auxiliary Anti Roll Force](#)

[Auxiliary Vertical Force](#)

Wheel Location

Wheel location is described using the following data:

- **Track width**
the distance among left/right wheel centers at design time (Y direction of global reference system).
- **Wheel center X-coordinate vs Z (Left/Right)**
wheel base variation in wheel location reference system as a function of wheel jounce.
- **Wheel center Y-coordinate vs Z(Left/Right)**
wheel track variation in wheel location reference system as a function of wheel jounce.



The track and base variation are described by spline data editable via the **Curve Editor**. Curve data can be imported or exported from/to external ascii file using the specific push buttons.

Each curve has wheel jounce as an independent variable (X).

For [dependent suspensions](#) (**only rear**) an extra independent variable (Z) inputs the paired wheel jounce.

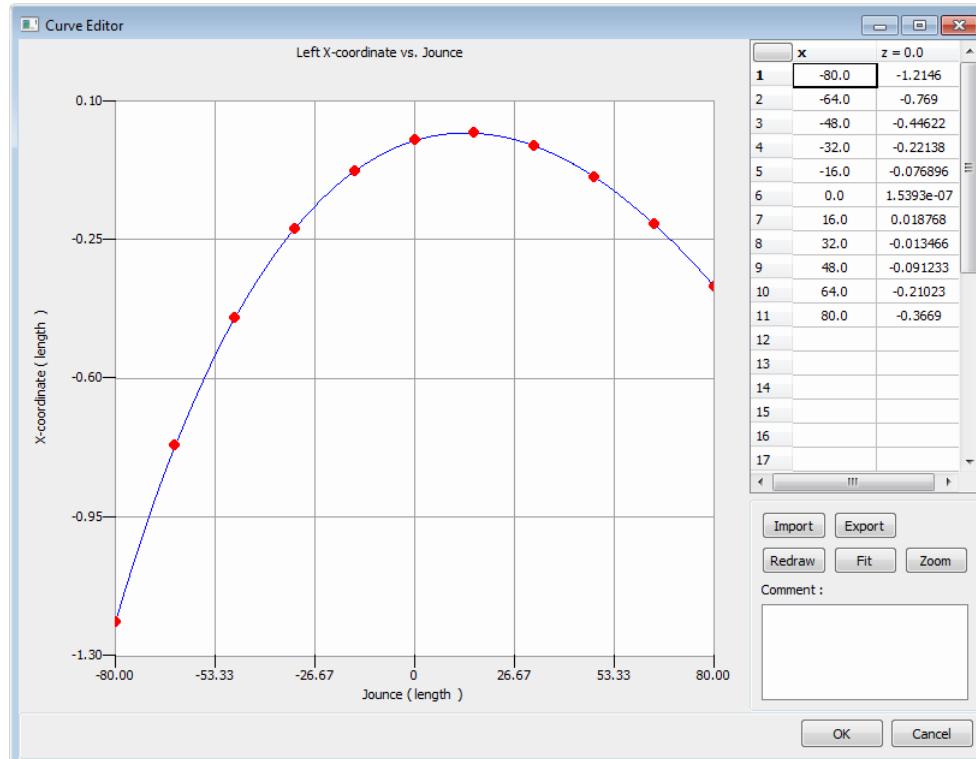
The **Wheel Location Reference System** is defined by a triad positioned at wheel center (left and right, at design time); the orientation is the same of VI-CarRealTime global reference system with X+ forward, Z+ vertical up, Y+ left.

For **front suspensions** the track and base variation are set as depending on steering wheel angle and jounce. A specific set of curve data is present in the [steering subsystem](#) file.

In that case the curve data in the Wheel Location editor:

- are not used by VI-CarRealTime solver if the front suspension curves are defined as [independent](#);
- are automatically summed up by VI-CarRealTime solver if the front suspension curves are defined as [dependent](#).

Note: the same concept also applied for [Wheel Orientation](#) panel.



Wheel Orientation

Wheel orientation in VI-CarRealTime is described using the following data:

- **Side View Angle**

is the [variation](#) of the wheel carrier side-view rotation angle with respect to the caster angle at design time. It is positive for a clockwise rotation, as seen from the left side of the vehicle.

For front suspensions, this spline is used only when it is mapped as a 3D spline [$sva = f(jounce_L, jounce_R)$], otherwise the [Side View Angle vs Input Steer vs Jounce](#) spline is used.

- **Toe Angle**

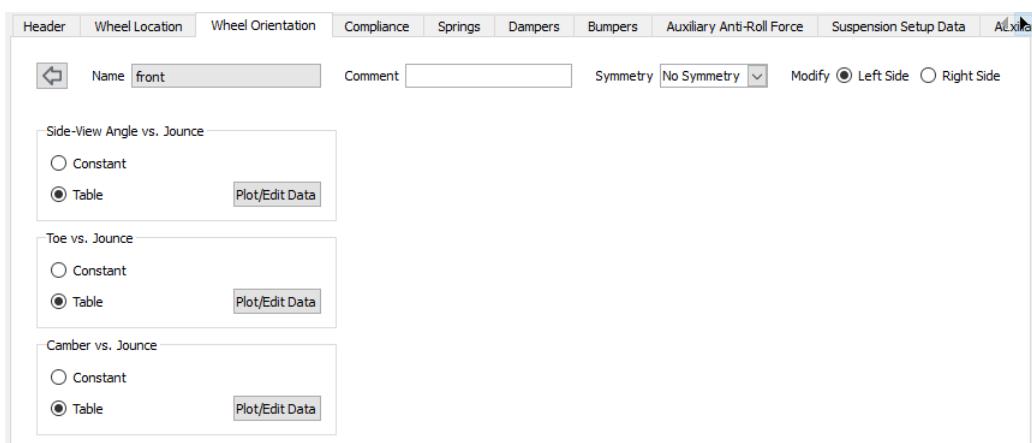
is the angle between the longitudinal axis of the vehicle and the line of intersection of the wheel plane and the vehicle's XY plane. It is positive if the wheel front is rotated in towards the vehicle body.

For front suspensions, this spline is used only when the [Use Steer input and Toe vs jounce Tables](#) radio button is checked in [Kinematic](#) panel of the steering subsystem.

- **Camber Angle**

is the angle the wheel plane has with respect to the vehicle's vertical axis. It is positive when the top of the wheel leans outward from the vehicle body.

For front suspensions, this spline is used only when it is mapped as a 3D spline [$camber = f(jounce_L, jounce_R)$], otherwise the [Camber Angle vs Input Steer vs Jounce](#) spline is used.

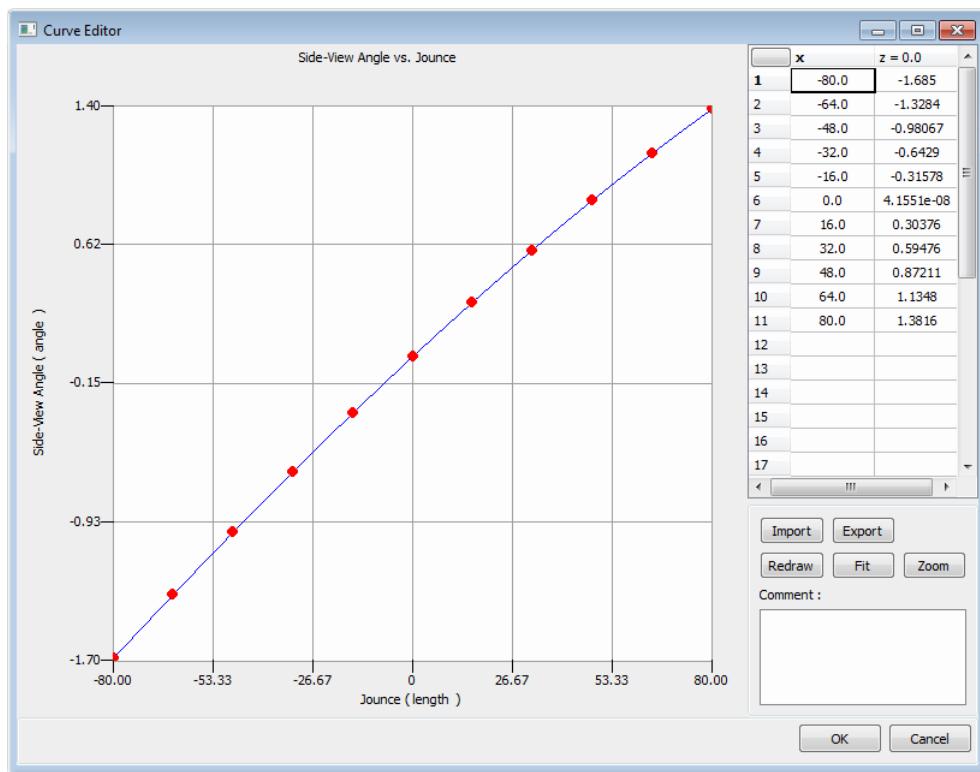


For each of the properties above the user can specify the dependency on suspension jounce as **constant rate** or **table data**:

- **Constant** value is set by the **Property Editor**.
- **Table** data can be input/edited using the **Curve Editor**. Each curve has wheel jounce as independent variable (X). For dependent suspensions (**only rear**) an extra independent variable (Z) input the paired wheel jounce.

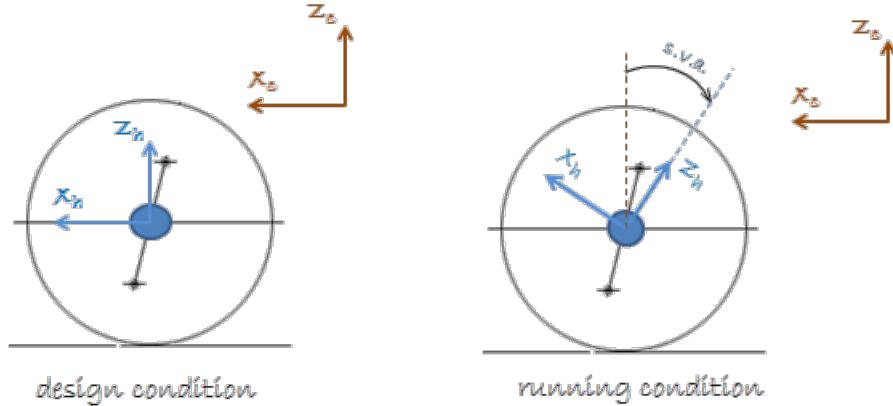
Symmetry

Wheel orientation includes symmetry option; the user can select it through the option menu in the Property Editor. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.



Note: The inclination angle, a measurement available in full-vehicle analyses, is the angle the wheel plane makes with respect to the road surface. The inclination angle is used for tire calculations.

The figure below illustrates the meaning of the side view angle (s.v.a.) spline.



The s.v.a. vs. Jounce spline measures the variation of the wheel carrier side-view rotation angle with respect to the caster angle at design time. In design condition the reference system of the Hub (blue part in figure) has the same orientation as the chassis reference (z_h is parallel to z_c), so the side view angle is 0; with respect to this design configuration, the side view angle changes as a function of the suspension jounce and of the steering wheel angle (**Side View Angle vs Jounce** and **Side View Angle vs Input Steer vs Jounce** splines).

More details on the **Side View Angle** can be found in the *Output Description* paragraph of the VI-SuspensionGen documentation.

The derivative of the **Side View Angle vs Jounce** curve is used to evaluate the anti-dive and anti-squat behaviour of the suspension.

VI-CarRealTime

Such a spline is not used by VI-CarRealTime solver for suspension kinematic computation; in fact the caster angle does not affect the kinematic behaviour of a suspension, it changes only the steering axis geometry (kingpin) and this phenomenon is characterized by the splines defined in the [kingpin panel](#) of the [steering subsystem](#).

Compliance

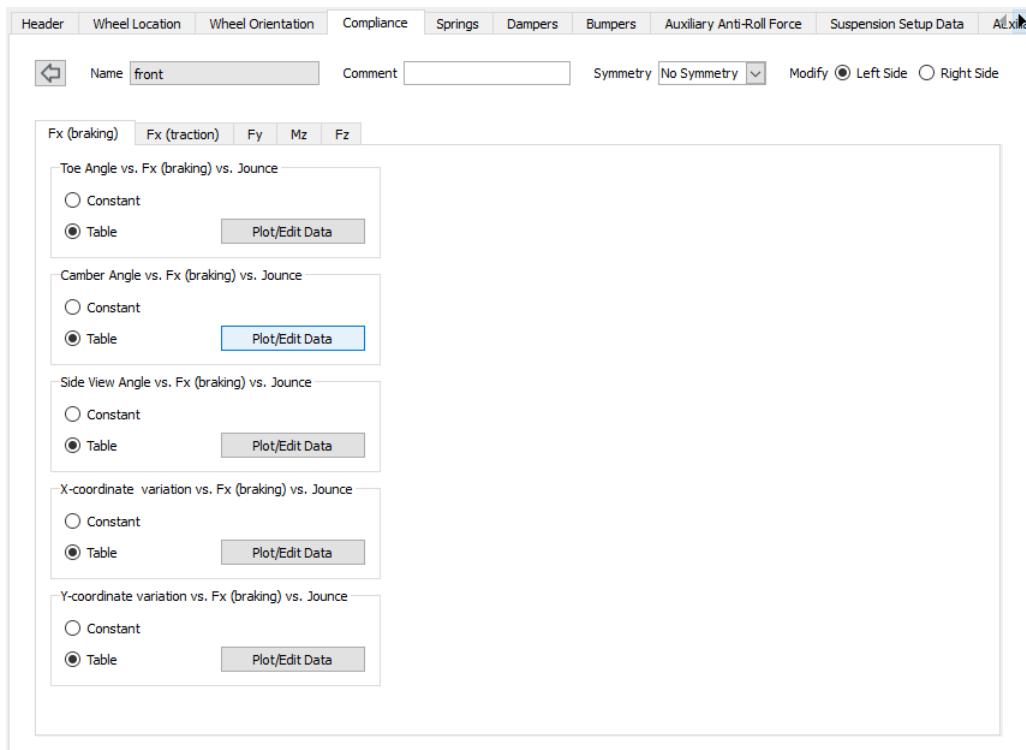
In VI-CarRealTime the wheel orientation is implemented using the effect superposition principle.

Given that the kinematic position and orientation being defined by the dependency on suspension jounce (left and right for dependent rear suspensions) and eventually steering wheel angle (for front suspensions), the effect of external loads on a given suspension property (track or base variation, toe, camber, side view angle) is computed as the sum of the effect given by all active loads taken singularly.

$$\varepsilon = \varepsilon_k + \sum_{i=1}^n \varepsilon_f$$

where:

- ε is the total value of a suspension property (toe angle, side view angle etc.).
- ε_k is the value of the suspension property given by kinematic dependencies (jounce, steering wheel angle).
- ε_f is the value of the suspension property **variation** given by a specific load (Fx, Fy, Tz etc.).



VI-CarRealTime vehicle model includes the dependency of the following data on external loads:

- **Toe Angle**
- **Camber Angle**
- **Side View Angle**
- **Wheel Center Y-coordinate Variation**

- **Wheel Center X-coordinate Variation**

The following external loads are considered as inputs:

- **F_x (braking)** - [ISO-W](#)
- **F_x (traction)** - [ISO-C](#)
- **F_y** - [ISO-W](#)
- **M_z** - [ISO-W](#)
- **F_z** - [ISO-W](#)

Note: for each wheel the longitudinal compliance switch between *F_x (traction)* spline and *F_x (braking)* spline is controlled by the braking moment; *F_x (traction)* spline are used until a braking moment is applied on the wheel.

The table below summarizes the compliance dependencies for a single corner.

Entity	Dependencies				
Toe Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Camber Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Side View Angle	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
Y-coordinate Variation	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z
X-coordinate Variation	$F_{x(braking)}$	$F_{x(traction)}$	F_y	T_z	F_z

Cross compliances

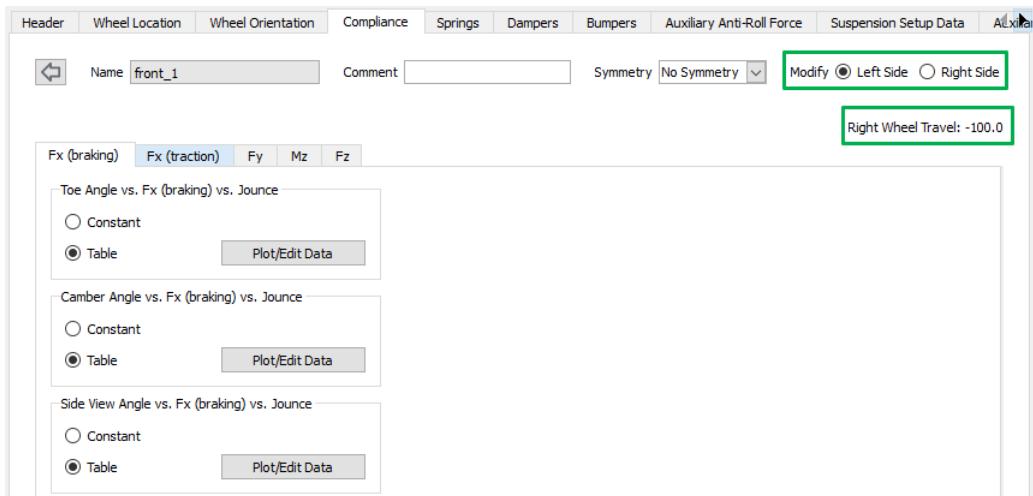
A VI-CarRealTime vehicle model also includes the dependency of the compliance of a wheel given by a force applied on the paired wheel. An equivalent set of data is input for direct and cross compliance (with the exclusion of FZ cross dependency).

For each of the properties above the user can specify the dependency on suspension jounce as **constant rate** or **table data**:

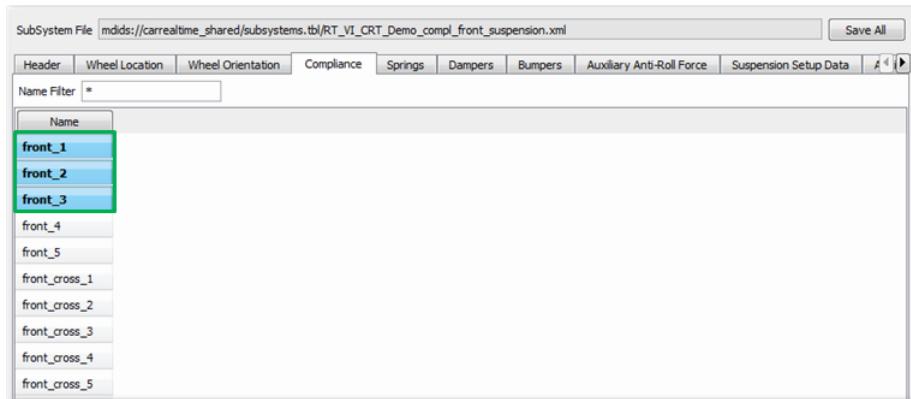
- **Constant** value is set by the **Property Editor**;
- **Table** data can be input/edited using the **Curve Editor**. Each curve has an external load as independent variable (X). It is also possible to specify a dependency on wheel jounce by an extra independent variable (Z) in the **table data** (exception for F_z which is a 2D spline).

In VI-CarRealTime there are two possible options regarding the number of compliance loadcases:

1. One direct compliance and one cross compliance set: in this case the compliance dependency of external force is expressed at different wheel travel levels (3D table columns), common between left and right wheel;
2. Multiple sets of direct and cross compliance: in this case each compliance set is corresponding to curves representing the dependency of suspension states on applied loads and wheel travel for a single wheel; the other wheel is kept at a fixed value which is shown in the VI-CarRealTime graphical user interface below the side radio box (top right of compliance editor). VI-CarRealTime is performing a multi-dimensional interpolation among all the available sets as a function of: applied force, left wheel travel, right wheel travel.



Example: the case of three sets of direct compliance curves for left and right works as follows:



Left compliance

1. the jounce of the *right* suspension is set to the **lowest** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
2. the jounce of the *right* suspension is set to the **intermediate** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
3. the jounce of the *right* suspension is set to the **highest** value and the jounce of the *left* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.

Right compliance

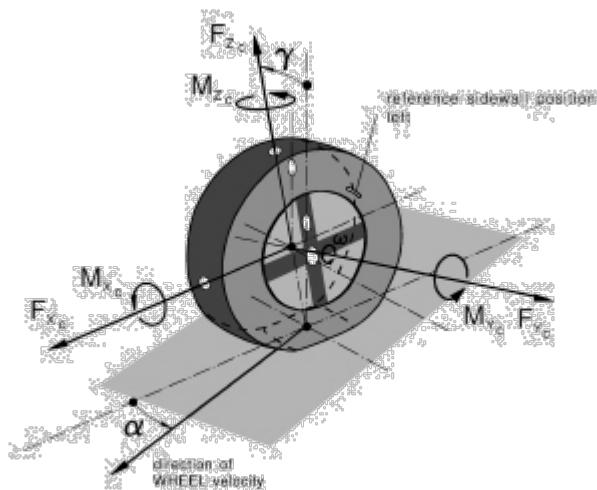
1. the jounce of the *left* suspension is set to the **lowest** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
2. the jounce of the *left* suspension is set to the **intermediate** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.
3. the jounce of the *left* suspension is set to the **highest** value and the jounce of the *right* suspension is swept from the lowest to the highest value. For each left/right wheel travel combination, external loads are swept from minimum to maximum.

Compliance Reference Systems:

$F_{x(\text{traction})}$ forces are applied at **ISO-C Axis System**.

- ISO-C Axis System:

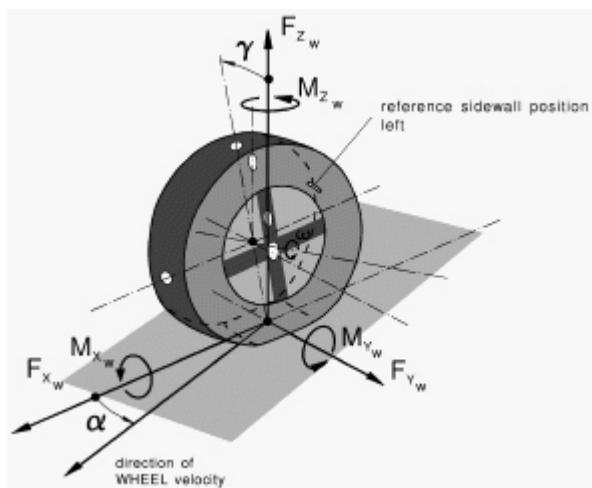
- The origin of the ISO-C axis system lies at the wheel center.
- The **+ x-axis** is parallel to the road and lies in the wheel plane.
- The **+ y-axis** is normal to the wheel plane and, therefore, parallel to the wheel's spin axis.
- The **+ z-axis** lies in the wheel plane and is perpendicular to x and y (such as $z = x \times y$).



$F_{x(\text{braking})}$ F_y T_z and F_z forces are applied at **ISO-W Axis System**.

- ISO-W Contact-Patch Axis System:

- The origin of the ISO-W contact-patch system lies in the local road plane at the tire contact point.
- The **+ x-axis** lies in the local road plane along the intersection of the wheel plane and the local road plane.
- The **+ z-axis** is perpendicular (normal) to the local road plane and points upward.
- The **+ y-axis** lies in the local road plane and is perpendicular to the + x-axis and + z-axis (such as $y = z \times x$).



VI-CarRealTime

Suspension properties are measured using the same convention adopted for wheel location and wheel orientation for suspension kinematics.

Note: a compliance curve is actually used by the solver only if the related activity flag has been set to on in [Suspension Data Compliance](#) panel.

Springs

The Spring is the main elastic component in the suspension.

Two types of springs are available in VI-CarRealTime:

- [Coil springs](#)
- [Air springs](#)

The component properties of a coil spring are described by:

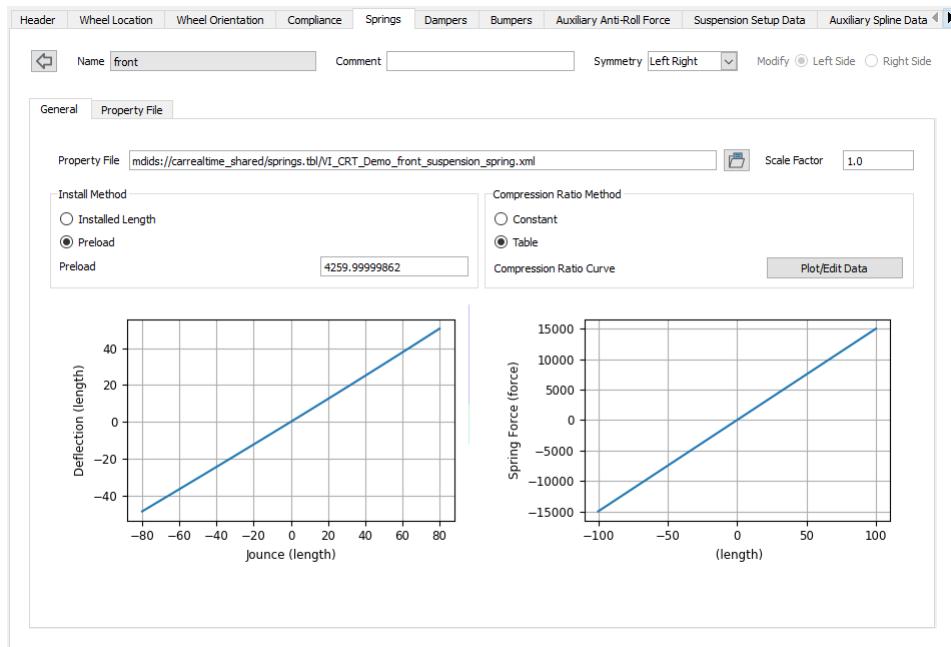
- **General panel**

used to project at wheel center the effect of spring component:

- [Install method](#)
- [Compression ratio method](#)

- **Property File panel**

used to describe component specific properties.



Suspension subsystem spring data:

Installation method

There are two possible installation methods for springs that can be selected using the Property Editor:

- **Installed Length**

This parameter describes the length of the spring at design time; the preload is computed depending on spring free length.

- **Preload**

It is the force acting at spring ends at design time.

Depending on the chosen method, Preload or Installed Length files display.

Installed Length and Preload are two different approaches for defining the same thing, that is how the spring is mounted on the vehicle. VI-CarRealTime internal spring always use the installed length, so if preload is set, the corresponding installed length is computed based on spring characteristics. If preload is specified rather than the installed length, then the installed length must be calculated by inverse table lookup. The value of the force when the spring compression is equal to (free length – installed length) should equal the preload.

Note: being Preload and Installed length parameters referring to the same xml file entry (InstallValue) and being the units variable depending on the type of spring (Translational or Rotational) and on the Installation Method chosen, the field units will not be shown. It is assumed that they are entered in a unit system congruent with the other subsystem data.

Compression ratio method

It is used to convert the spring component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

Value is set by **Property Editor** and expresses the ratio between the wheel jounce and component deflection.

- **Table**

Data can be input/edited using the **Curve Editor**. The curve has wheel jounce as independent variable (X) and expresses the spring deflection(Y).

Property File

The spring xml [property file](#) is read and the data are used to fill up the Properties panel. The user can modify the spring element characteristics and then save the modification in the spring property file.

- **Scale Factor**

scaling factor which multiplies the spring characteristic curve.

Method

- **Linear**

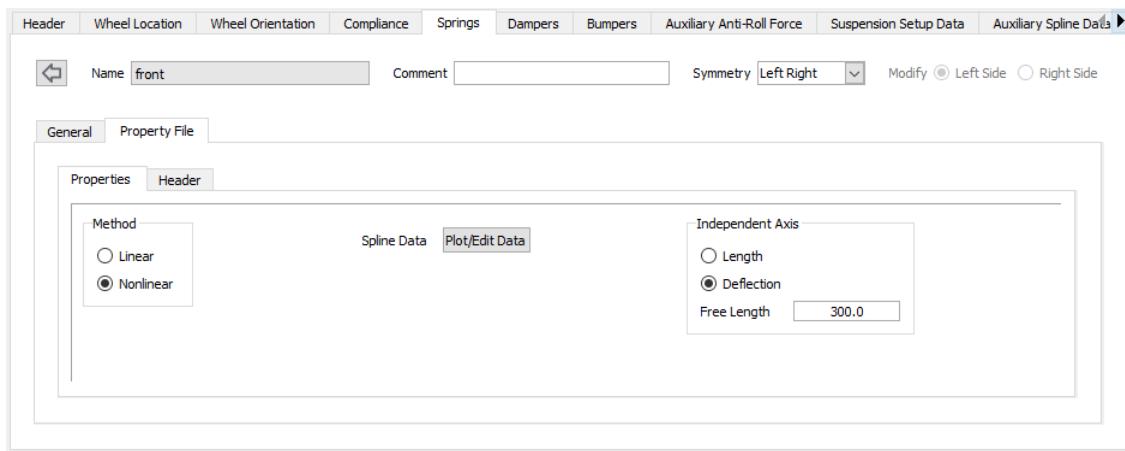
Allows to define a linear spring property file characteristic. When such toggle is selected the user is allowed to define the spring Free Length and the spring Linear Rate.

- **Nonlinear**

Allows to define a non-linear spring property file characteristic. When such toggle is selected the user is allowed to define the Spline Data, that is the spline characteristic curve. Furthermore the Independent Axis type must be chosen, between:

- **Length:** it is the "real" length of the spring, the distance between the spring ends. Since in VI-CarRealTime the spring is conceptual, the term "length" loses its intrinsic meaning: the spring length in VI-CarRealTime is 0 in design condition, so when *Length* is used as independent axis, the Spline Data must be entered considering spring length at design equals to 0.

- **Deflection:** it is the spring deformation with respect to its design length. When *Deflection* is selected as Independent Axis, also the *Free Length* parameter must be entered in order to properly define the spring characteristic.



Symmetry

Spring properties include symmetry option; the user can select it inside the option menu using the Property Editor. The GUI also features the capability of defining which side the properties will belong to. The effect of the selection is to copy the data content of the selected side into the paired one.

VI-CarRealTime supports both rotational and translational spring properties.

Available methods for linear spring property file data are:

- **Linear**

user will specify linear rate and spring free length.

- **Non Linear**

spring data will be input by a curve editable using the Curve Editor; other data needed are the independent axis in spline data (Length or Deflection) and Free Length.

For rotational springs the available parameter is the linear torsional stiffness.

The component properties of an air spring are described by:

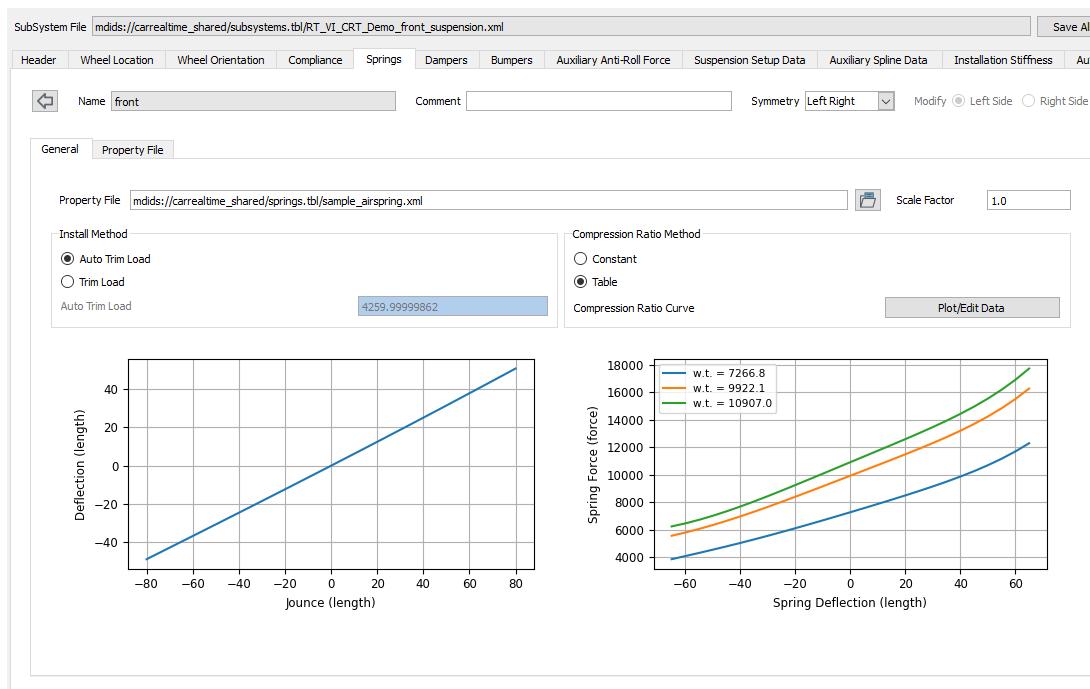
- **General panel**

used to project at wheel center the effect of air spring component:

- [Install method](#)
- [Compression ratio method](#)

- **Property File panel**

used to describe component specific properties.



Suspension subsystem air spring data:

Install method

There are two possible installation methods for air springs that can be selected using the Property Editor:

- **Auto Trim Load**

A PID controller is used during the static analysis in order to have at the static equilibrium an air spring deflection equals to 0 (that is an air spring length equals to the design time length).

- **Trim Load**

It is the force in the air spring when the suspension is at the trim height.

Compression ratio method

It is used to convert the air spring component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

Value is set by **Property Editor** and expresses the ratio between the wheel jounce and component deflection.

- **Table**

Data can be input/edited using the **Curve Editor**. The curve has wheel jounce as independent variable (X) and expresses the spring deflection(Y).

Property File

It sets the path of the property file containing component data. The air spring component data can be edited in the Property File tab.

- **Scale Factor**

scaling factor which multiplies the spring characteristic curve.

Symmetry

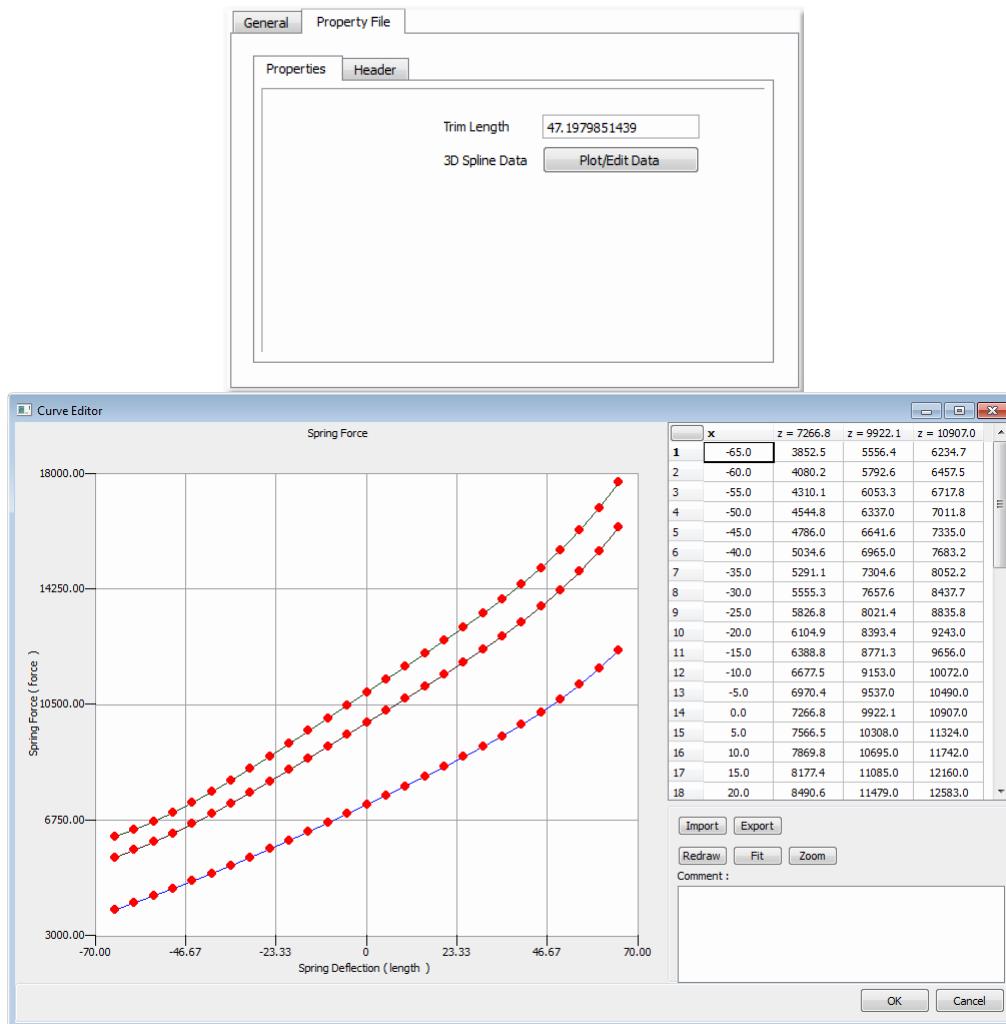
Spring properties include symmetry option; the user can select it inside the option menu using the Property Editor. The GUI also features the capability of defining which side the properties will belong to. The effect of the selection is to copy the data content of the selected side into the paired one.

Property File data:

The air spring property file is defined as a 3D spline [$Y = f(X, Z)$] where:

- X is the air spring deflection;
- Z is the load in the air spring when the suspension is at trim height. The load corresponds to the load defined in the [Install Method](#) panel.

The **Trim Length** is defined as the difference between the length of the air spring when the suspension is at trim height and its design time length. Spring design time length is 0 in VI-CarRealTime model.



The air spring force is computed as:

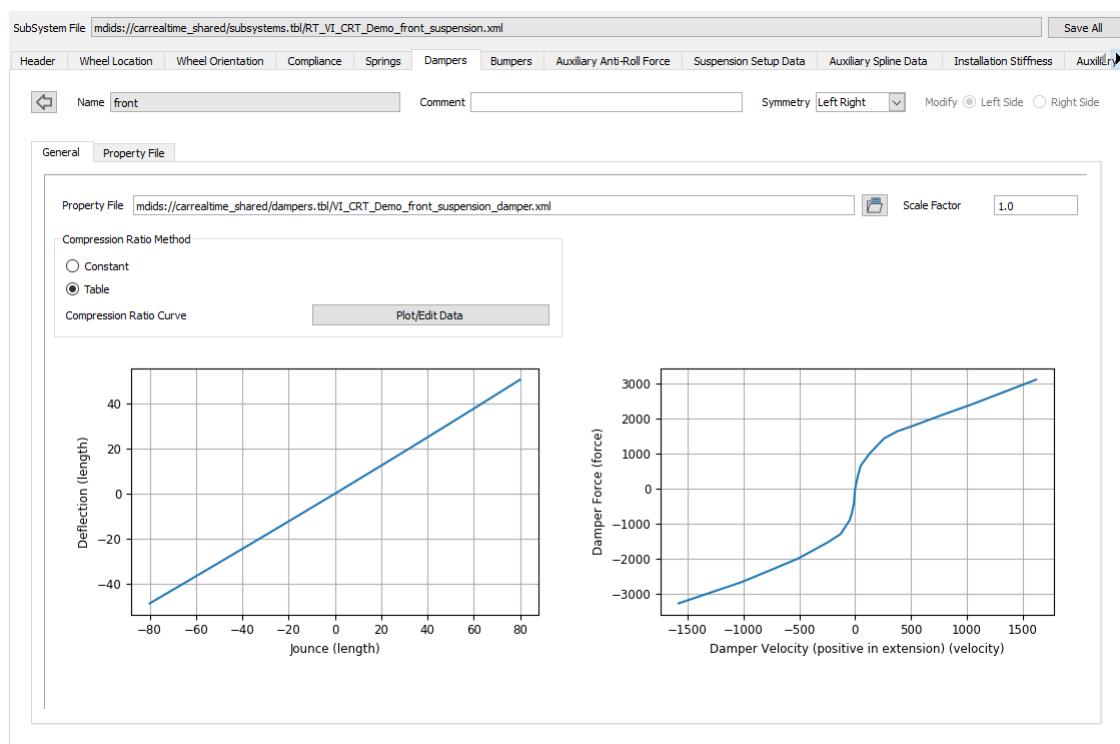
$$F_{airspring} = SPLINE(trimLength - deflection, trimLoad)$$

Dampers

The Damper is the main elastic component in the suspension.

In VI-CarRealTime the component properties are described by:

- **Suspension data**
used to project the effect of damper component at wheel center:
 - [Compression ratio method](#)
- **Property File**
used to describe component specific properties.



Suspension subsystem damper data:

Compression ratio method

It is used to convert damper component deformation to the equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**
the value is set by the **Property Editor** and expresses the ratio between the wheel jounce and the component deflection.
- **Table**
data can be input/edited using the **Curve Editor**. The curve has the wheel jounce as an independent variable (X) and expresses the damper deflection(Y).

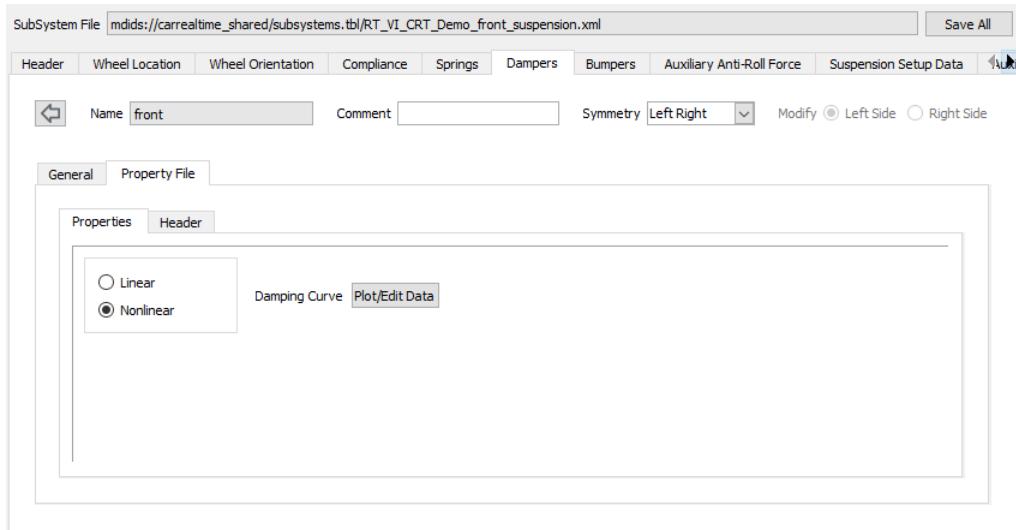
Property File

The Suspension data Damper Property Editor shows the path of the property file containing component data. Damper component data can be edited in a separate tab.

- **Scale Factor**
scaling factor which multiplies the damper characteristic curve.

Symmetry

Damper properties include symmetry option; the user can select it by the option menu in the **Property Editor**. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.



Available methods for property file damping are:

- **Linear**

the user has to specify the linear rate.

- **Non Linear**

damper data will be input by a curve editable using the Curve Editor. VI-CarRealTime solver supports both 2D damper and 3D damper curves. 2D damper curve defines the damper element force as a function of [damper velocity](#) (positive in extension); 3D damper curve defines the damper element force as a function of [damper velocity](#) (first independent variable, positive in extension) and [damper displacement](#) (second independent variable, positive in extension). Damper displacement is the damper deflection with respect to the design condition (0 damper displacement). An example of 3D damper property file is shipped with carrealtime_shared database: [sample_3D_damper.xml](#).

VI-CarRealTime supports the presence of an auxiliary damper in parallel with the main damper. The auxiliary damper is automatically created in suspension subsystem during model export from Adams/Car if [Auxiliary Damper](#) flag is activated.

When the auxiliary damper instance is present in a subsystem, further [outputs](#) will be generated in order to review auxiliary damper force, velocity and displacement.

To manually add an auxiliary damper in a suspension subsystem:

- edit suspension subsystem xml file in a text editor;
- copy the entire <DamperPair> block and paste it right below the end of it;
- change the damper name (*name* parameter);
- save the subsystem file.

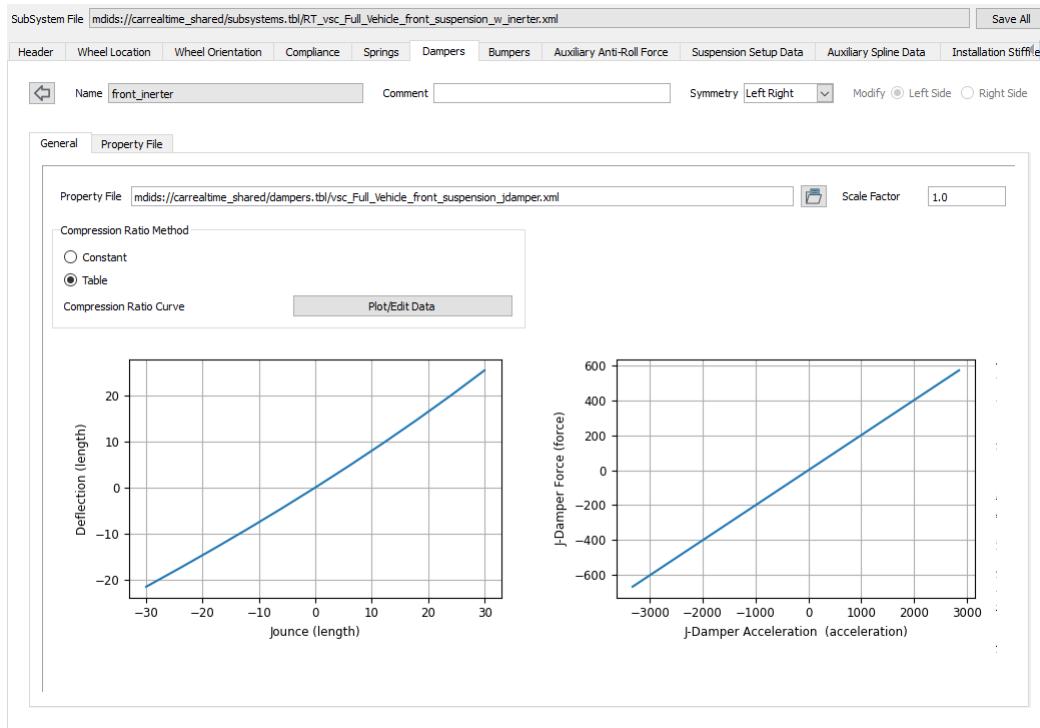
Dampers panel will now contain two damper instances:

SubSystem File: mdids://private/subsystems.tbl/FS_double_damper.xml					Save All	
Header	Wheel Location	Wheel Orientation	Compliance	Springs	Dampers	Bumpers
Name Filter: *						
Name	Left Active	Right Active	Comment			
front	yes	yes				
front_auxiliary	yes	yes				

The VI-CarRealTime model may also include the presence of a third central damper (often present in some race suspension architecture). The GUI will show an appropriate editor with the same features of main dampers. Third damper instance is going to be automatically created when exporting the model from Adams/Car when the original full model includes it or when exporting the model from VI-SuspensionGen using a suspension template supporting the central damper. In the other cases the user will have to add it manually to the suspension xml file. An example of a vehicle including a third damper can be found in VI-CarRealTime shared database (\vsc_Full_Vehicle).

J-Dampers

The VI-CarRealTime model supports the presence of J-damper or *inerters*.



The element is similar to a standard damper having the force proportional to the relative acceleration rather than velocity.

J-Dampers support a specific property file and include the following methods:

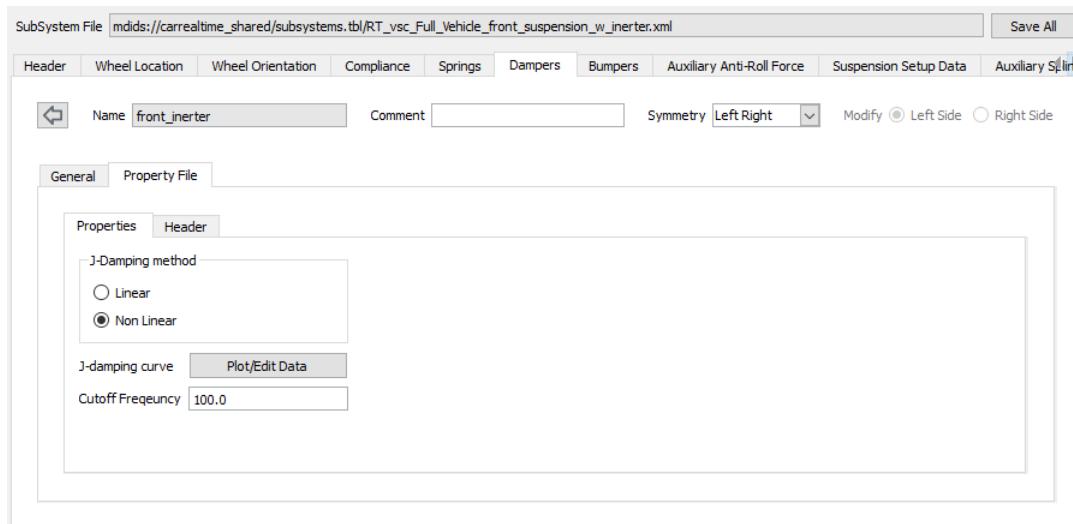
- **Linear**
the user has to specify the linear rate.
- **Non Linear**
damper data will be input by a curve editable using the Curve Editor.

This element exposes an additional parameter:

- **Cutoff Frequency**

VI-CarRealTime

It is the cutoff frequency of the low-pass filter used on the input acceleration to avoid numerical issues during integration.



Note: an example of front suspension subsystem with j-damper element (*RT_vsc_Full_Vehicle_front_suspension_w_inerter.xml*) is shipped with carrealtime_shared database.

Bumpers

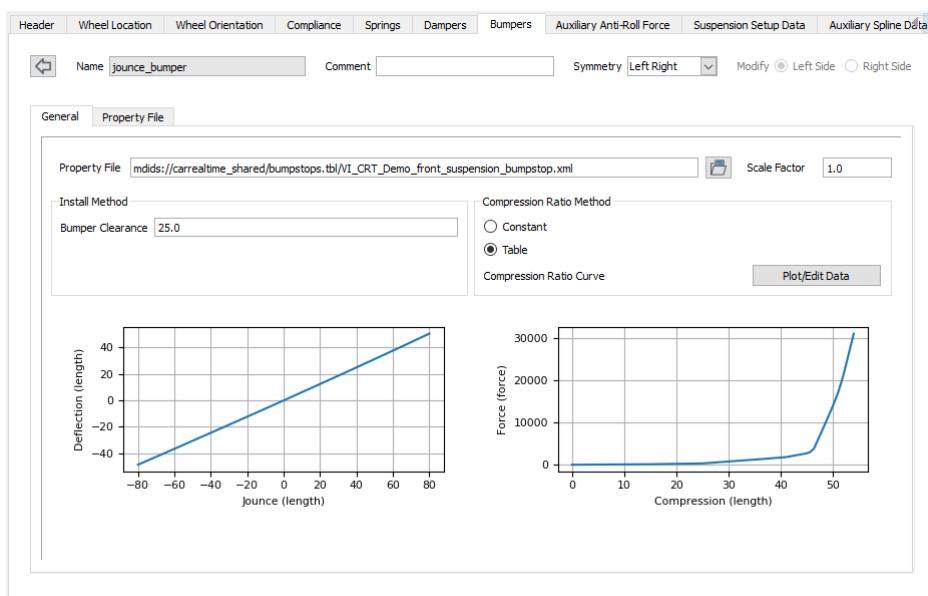
Bumper elements are used to limit suspension jounce.

In each VI-CarRealTime suspension model there are two pairs of instances of bumpers, one for jounce and the other for rebound stop.

Header	Wheel Location	Wheel Orientation	Compliance	Springs	Dampers	Bumpers	Auxiliary Anti-Roll Force	Suspension Setup Data	A
Name Filter *									
Name	Left Active	Right Active	Comment						
rebound_bumper	yes	yes							
jounce_bumper	yes	yes							

The bumper properties are described by:

- **Suspension data**, used to project at wheel center the effect of spring component:
 - Bumper clearance;
 - Compression ratio method.
- **Property File**, used to describe component specific properties.



Suspension subsystem spring data:

Compression ratio method

It is used to convert the bumper component deformation to an equivalent wheel jounce, since in VI-CarRealTime the suspension is conceptual. Available methods are:

- **Constant rate**

value is set by **Property Editor** and express the ratio between the wheel jounce and component deflection.

- **Table**

data can be input/edited using the **Curve Editor**. The curve has wheel jounce as an independent variable (X) and expresses the bumper deflection (Y).

Bumper Clearance

Describes the deformation that the component must undergo before engaging and applying a force. For models including suspension adjustments the value refers to conditions after the setup, in the other cases to design conditions.

Property File

It sets the path of the property file containing component data. Bumper component data can be edited in a separate tab.

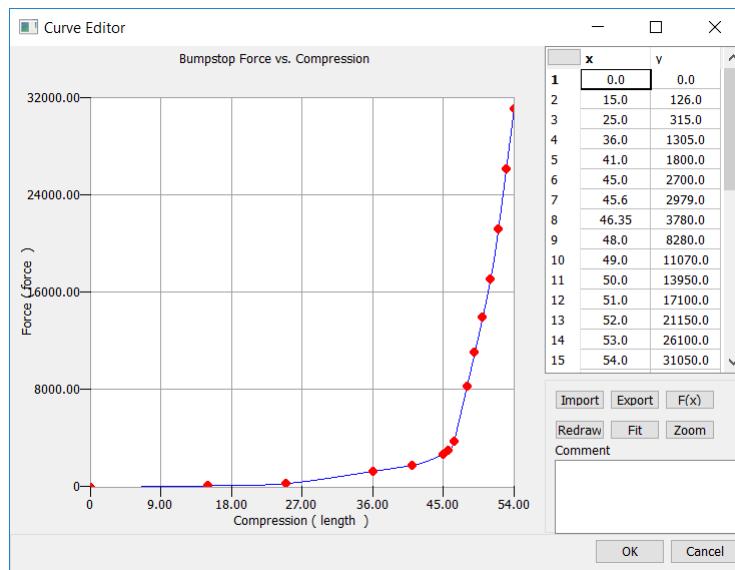
- **Scale Factor**

scaling factor which multiplies the bumper characteristic curve.

Symmetry

Bumper properties include symmetry option; the user can select it via the option menu in Property Editor. The GUI also features the capability of defining which side the properties that the user is modifying will belong to. The effect of the selection is to copy the data content of the selected side onto the paired one.

Property File data will be input by a curve editable using the Curve Editor, expressing the component force as a function of deformation after engagement.



The VI-CarRealTime model may also include the presence of a third central elastic element (often present in some race suspension architecture). The GUI will show an appropriate editor with the same features of paired bumpers (third_bumper). Third bumper instance is going to be automatically created when exporting the model from Adams Car when the original full model includes it. In the other cases user will have to add it manually to suspension xml file. An example of a vehicle including third bumper can be found in VI-CarRealTime shared database (\vsc_Full_Vehicle).

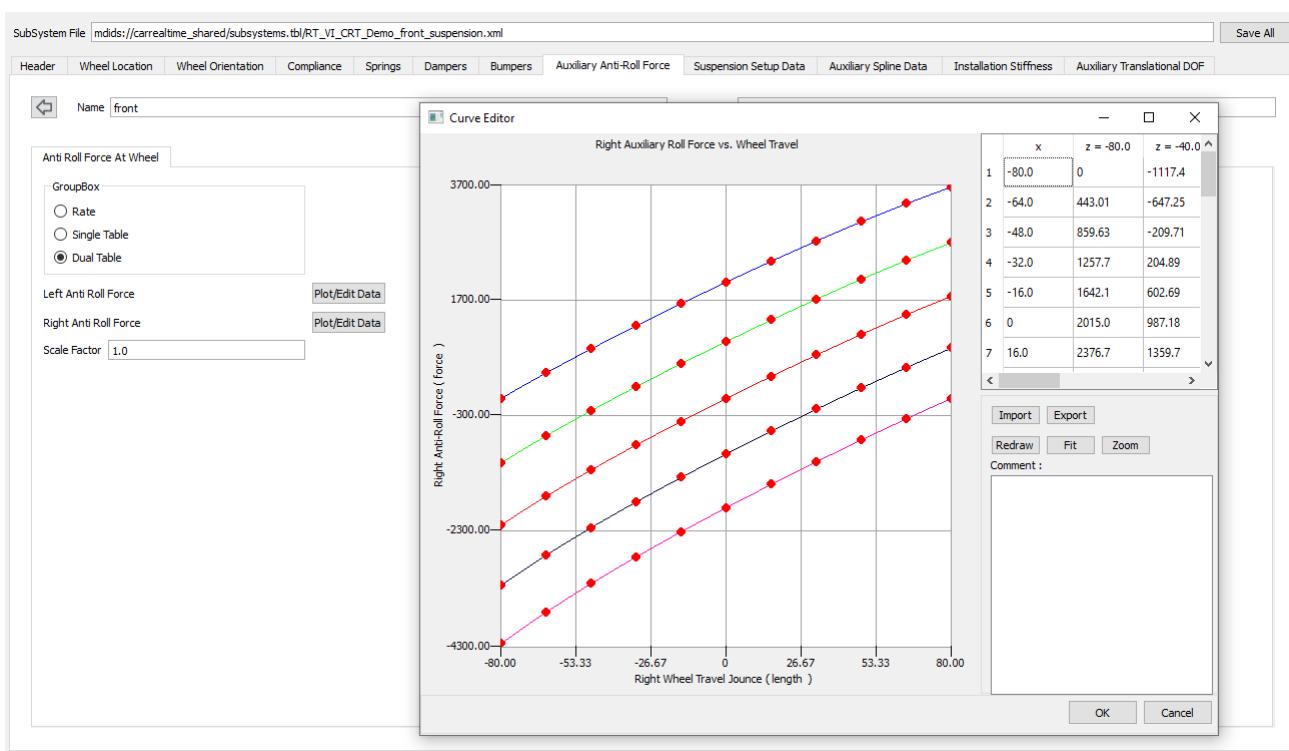
Auxiliary Anti Roll Force

Auxiliary anti roll forces are used in VI-CarRealTime to introduce the effect of elastic elements connecting left and right paired wheels (typically anti roll bars).

Depending on the conceptual suspension model the force is projected at the wheels as a pair of opposite vertical forces acting between the wheel (unsprung mass) and the body chassis.

The value of the force is related to the left/right wheel jounce difference and can be input using two methods:

- **Rate**
the force intensity is computed by the product of the rate value and the L/R jounce difference.
- **Single Table**
the force is introduced using a spline having as first independent variable (X) the L/R jounce difference (delta jounce). It is also possible to have a second independent variable (Z) which is the L/R average jounce. This allows to take into account the non linear behaviour of the anti roll bar due to suspension geometry.
- **Dual Table**
the force is introduced using two splines having as first independent variable (X) the wheel travel of one wheel and as second independent variable (Z) the other wheel travel. The Dual Table option supports the possibility of asymmetric auxiliary anti roll forces.
- **Scale Factor**
scaling factor which multiplies the anti roll force characteristic curve.



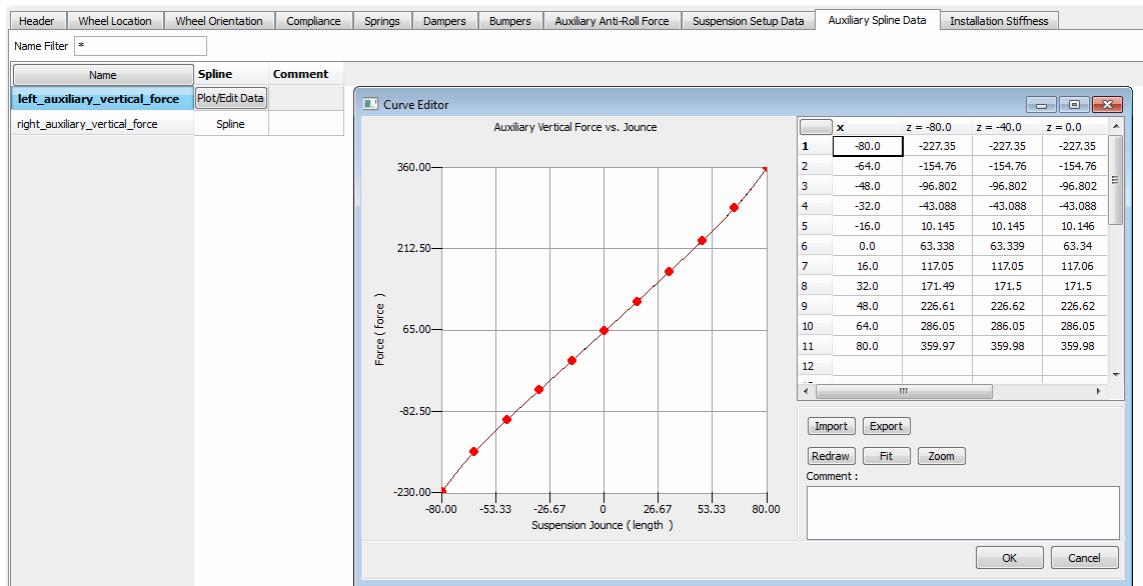
Note: It is possible to specify a proportional damping applied to the anti roll auxiliary forces using the auxiliary_roll_damping tree in [System Parameters](#).

Auxiliary Vertical Force

Auxiliary vertical forces introduce the effect of suspension vertical elasticity not due to standard components (springs, bumpers etc.).

It is defined by two sets of spline data expressing the wheel auxiliary vertical force as a function of wheel jounce. Optionally a second independent variable (Z) can be used to support the auxiliary vertical Force dependency of both left and right axle wheels travels.

Spline data can be edited using the Curve Editor.



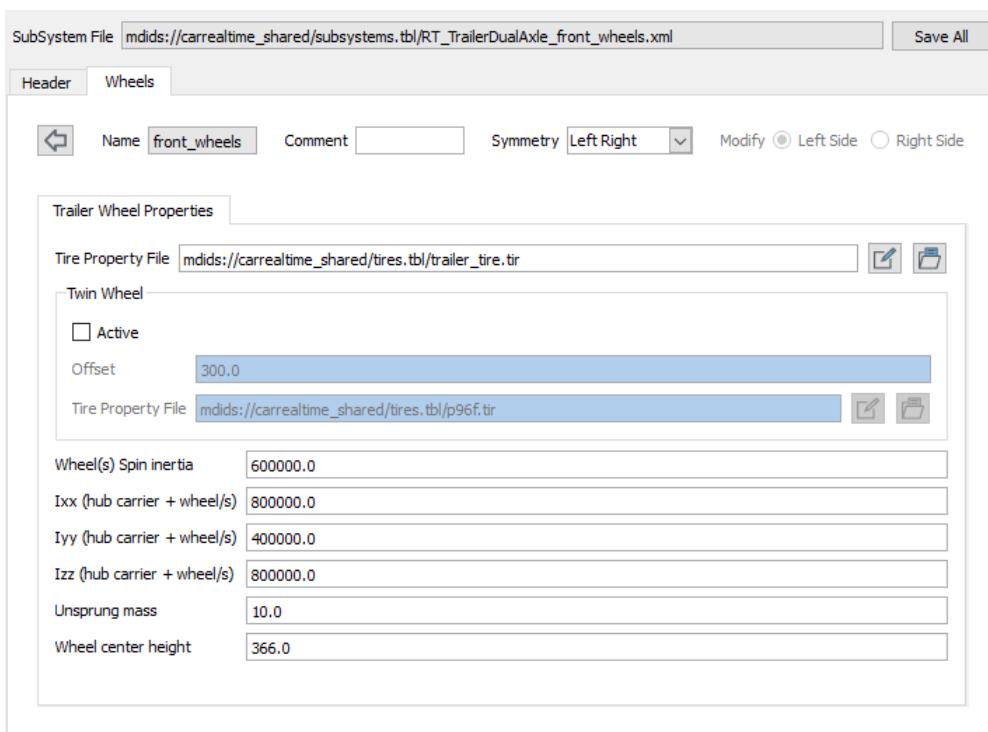
Wheels Subsystem

The wheels subsystem properties collect mass, inertia tire property file information of unsprung mass pairs of VI-CarRealTime trailer model.

Wheels subsystem Property Editor includes the following data:

- **Tire Property File**
property file for tire forces computation; supported format are: MF_05, PAC2002, PAC_MC, VI_TIRE.
- **Twin Wheel**
Check the Twin Wheel box to activate twin wheels and second wheel parameters:
 - **Offset**
set the distance from the first wheel to the second one (positive values).
 - **Tire Property File**
property file for tire forces computation; supported format are: MF_05, PAC2002, PAC_MC, VI_TIRE.
- **Wheel(s) Spin Inertia** (one wheel)
it sets the spin inertia of the wheel.
- **Ixx (hub carrier + wheel/s)**
Ixx moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Iyy (hub carrier + wheel/s)**
Iyy moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Izz (hub carrier + wheel/s)**
Izz moment of inertia of unsprung mass. it is measured in the [Wheel Location](#) reference frame.
- **Unsprung Mass**: it includes the mass of a single wheel and the portion of suspension masses not belonging to sprung part.
- **Wheel Center Height**: it is the vertical position of wheel center expressed in [Global Reference Frame](#).

Note: Cross moment of inertia of unsprung mass are neglected.



Brakes Subsystem

For the trailer model two different brake models can be implemented: the first is similar to the one used for the vehicle model (see [vehicle model brake subsystem](#) for further info), the second uses the longitudinal force applied on the hitch ball to apply a proportional force on the brakes.

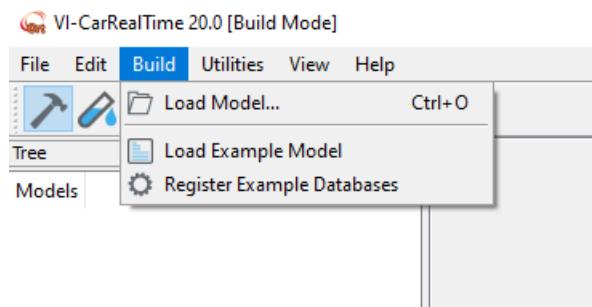
The brake model includes the following parameters:

- **brake_active_flag**
to activate the brake on the trailer model.
- **overrun_brake_flag**
if active (1) then the longitudinal force read on the hitch ball will be used to calculate the brake force.
- **lower/upper_longitudinal_threshold**
sets the longitudinal force thresholds. Used only if the overrun_brake_flag is set to 1.
- **normal_force_max** (front/rear)
sets the maximum value for the force to be applied on brakes.
- **mu** (front/rear, left/right)
friction coefficient.
- **effective_piston_radius** (front/rear, left/right)
radius for applying friction force.

	Left Value	Right Value	Comment
brake_active_flag	1	1	
overrun_brake_flag	1	1	
lower_longitudinal_threshold	1000.0	5000.0	
upper_longitudinal_threshold	5000.0	2000.0	
normal_force_max	2000.0	180.0	
mu	0.4	0.4	
effective_piston_radius	180.0	180.0	
normal_force_max	2000.0	180.0	
mu	0.4	0.4	
effective_piston_radius	180.0	180.0	

Example models and databases

Example vehicle models and databases can be accessed in VI-CarRealTime from the **Build** button in menu toolbar.



The following actions are possible:

- **Load Model...**
it will load any vehicle model.

- **Load Example Full Vehicle Model**
it will load the VI_CRT_Demo vehicle model from carrealtime_shared database.

- **Register Example Databases**
it will add a set of databases to the current VI-CarRealTime session, each database including a specific vehicle model. The new models offer the access to realistic data - consistent with modern vehicles available in the market - for the most popular segments. These models have been specifically developed for those customers and companies that, not being directly involved in the development of their own cars, require the availability of realistic vehicle models for a wide variety of simulation scenarios, ranging from components (dampers, tires, suspension elements, etc.) to systems and sensor development (ABS, EPS, TCS, etc.) or specific offline/online applications (SIL, HIL, Fuel Consumption Analysis, ADAS etc.).

All vehicle models have been tested for the typical events involved in the above mentioned applications, including start/stop routines prescribed by emission assessment cycles. The models can be also used in conjunction with a driver simulator environment.

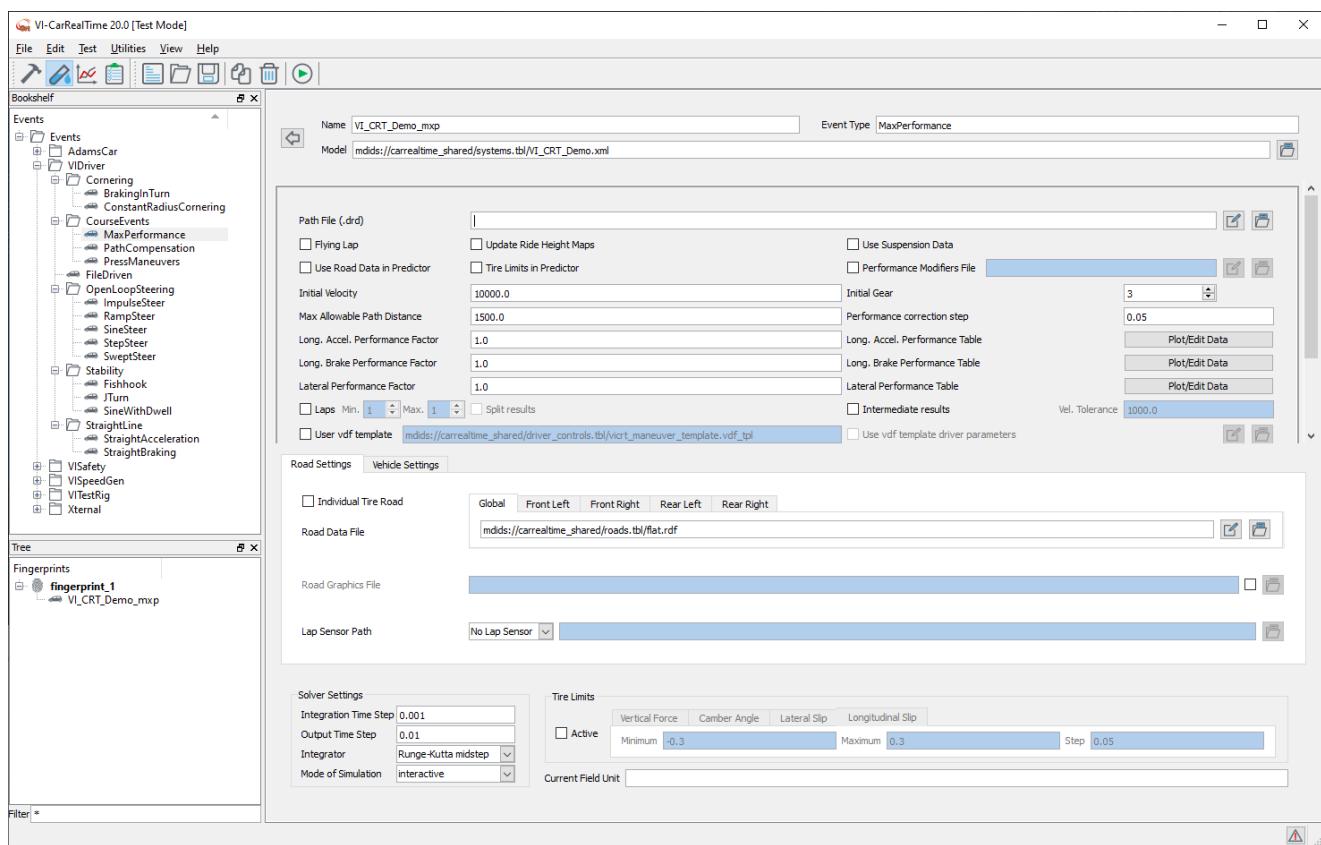
1.2.4 Test Mode

In the VI-CarRealTime Test mode simulation events can be created and run.

The main window is organized in the following elements:

- **Event Bookshelf**
shows the list of the available event classes; it is used to add a specific event instance to a fingerprint.
- **Fingerprint Treeview**
shows the tree of [fingerprints](#) present in session; double click on a single event instance opens the specific Property Editor.
- **Property Editor**
sets the parameter for the single event.

[Learn how to use Event bookshelf and fingerprint tree view in test mode](#)



Simulations are started clicking on the Play icon.



Events

Events in VI-CarRealTime are class objects defining a specific type of simulation.

Available events are divided according to event category:

- [Adams Car](#)
- [VIDriver](#)
- [VISpeedGen](#)
- [VITestRig](#)
- [Xternal](#)

[Learn how to create a new event instance](#)

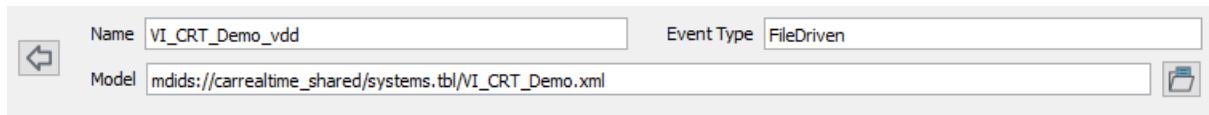
When a new event instance has been created it can be edited by Property Editor; some of the event properties in VI-CarRealTime are common to some or all the event classes:

- [Event Header](#)
- [Road Settings](#)
- [Vehicle Settings](#)
- [Solver Settings](#)

- [Tire Limits](#)

Event Header

A common header is available for each event.



The following fields are available:

- **Name**

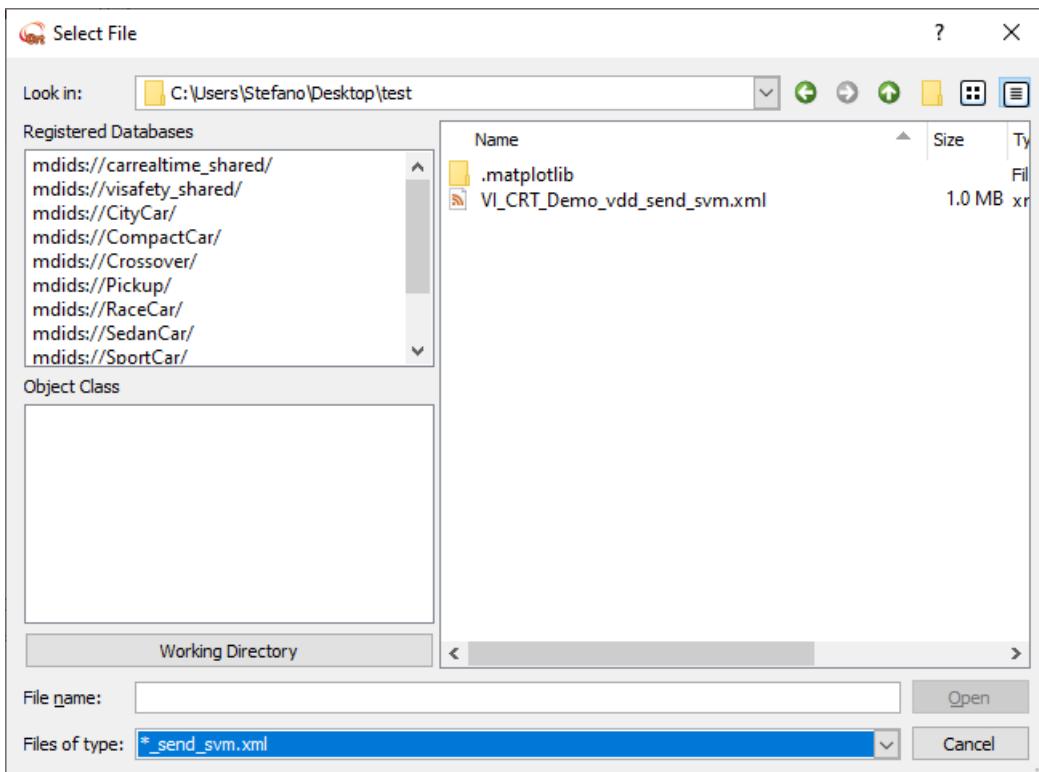
it defines the name of the event to be submitted.

- **Event Type**

it is the type of the event to be submitted: it corresponds to the event name in the Events tree.

- **Model**

it references the model that will be used during the simulation. By default it is the system file the user has selected, anyway it is also possible to select a *.send_svm.xml file as Model input by browsing on button:



send_svm.xml file can be either a standard send file in plain xml format or an [obfuscated](#) one.

Limitations:

The following events are not supported for send model files input :

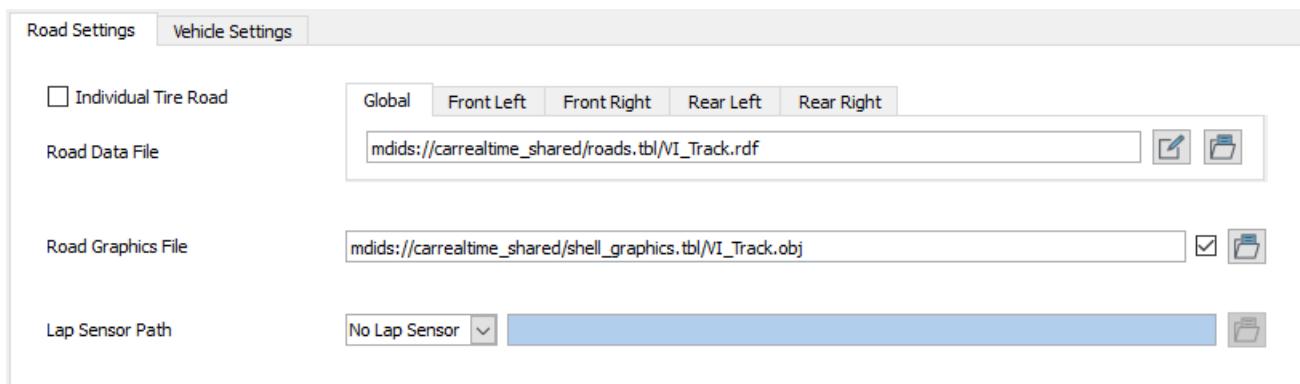
- [Fishhook](#)
- [JTum](#)
- [SineWithDwell](#)

Obfuscated send model files input do not support the following events:

- [Fishhook](#)
- [JTurn](#)
- [SineWithDwell](#)
- [MaxPerformance](#)
- [CurbTripRollover](#)
- [SCANeR](#)

[Live Animation](#) is not supported when a send file is used as Model.

Road Settings



- **Individual Tire Road**

When the toggle button is unchecked, the user is asked to enter a single road data file which represents the road that will be used during the simulation. With the toggle button checked, the individual road feature is enabled and the user must enter four road data files for each tire (Front Left, Front Right, Rear Left, Rear Right).

- **Road Data File**

It is used in VI-Driver and Driving Machine events; supported formats are [RDF](#), CRG.

VI-Grade offers a standalone product, VI-Road, which allows the user to model, edit and export road data files (rdf); furthermore VI-Road can be used to generate path trajectories for VI-Driver for a given road. For further details please refer to VI-Road documentation.

- **Road Graphics File**

It is used in [Driving Machine](#) and [Max Performance](#) events; it includes graphics for road to be used in post-processing; supported formats are .shl and .obj; path graphics file are loaded if found when having the same road graphics file name+_drv.*

- **Lap Sensor Path File**

It is used for optional Lap Time computation of the track. Three option are available:

- **No Lap Sensor**

no lap time computation when running the event.

- **User Defined**

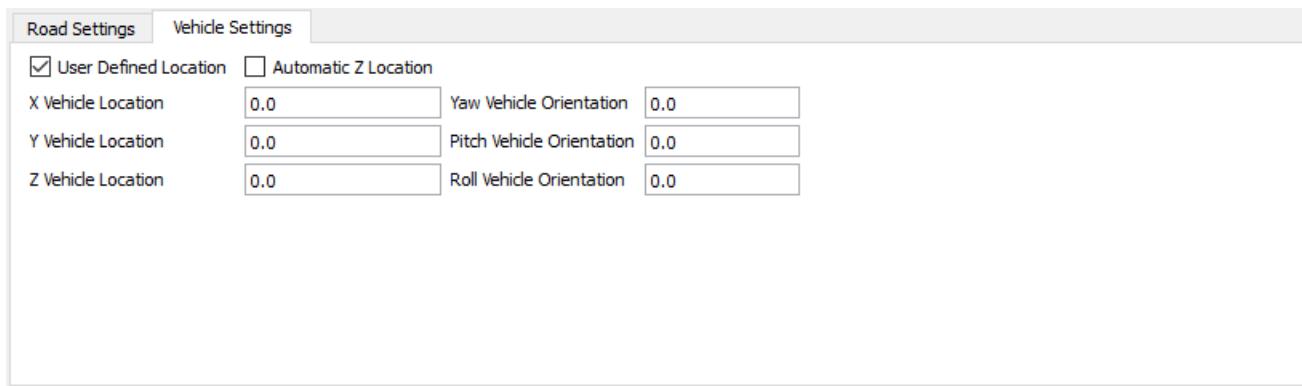
select a DRD or a supported Road Data File that will be used to compute the lap time.

- **Automatic**

uses the selected [Road Data File](#) to compute the lap time.

Note: Supported Road Data Files are CRG and [Mesh RDF](#) containing the [TRACK_PATH](#) block.

Vehicle Settings



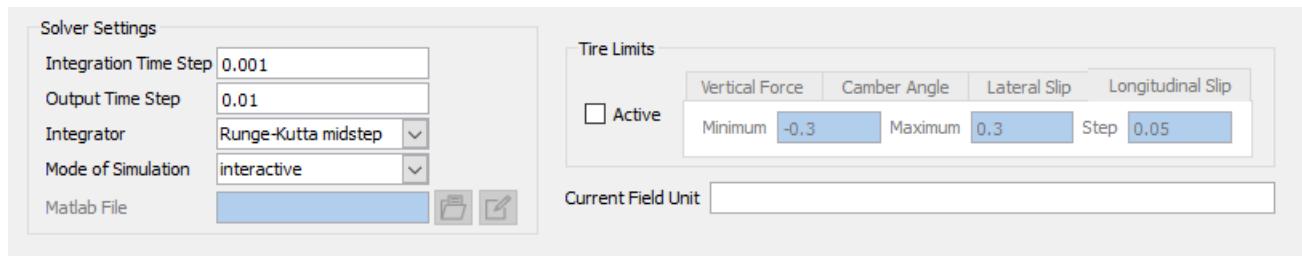
- **User Defined Location**

When the toggle button is checked, the user is allowed to offset the vehicle with respect to its design location/orientation. Such values (X, Y, Z, Yaw, Pitch, Roll) represents the footprint location of [driver sensor marker](#) on the road plane.

- **Automatic Z Location**

When the toggle button is checked the user is not allowed to enter the Z location of the vehicle: the Z is automatically computed by the solver by looking at the height of the road in that point.

Solver Settings



- **Integration Time Step**

It is the time step at which equations of motion are solved.

- **Output Time Step**

It is the time step used to write out result file. It must be a multiple of the Integration Time Step; otherwise it is computed as $ROUND(Output\ Time\ Step / Integration\ Time\ Step) * Integration\ Time\ Step$.

- **Integrator**

Select the integrator to be used to integrate the equations of motion: *Runge-Kutta midstep* or *Euler*.

- **Mode of Simulation**

Available modes are:

- *files only*

only the files needed to run the simulation using the standalone solver are created. It also creates files needed for further post-processing (graphic models, graphic files etc.). No simulation is run. Click [here](#) to learn how to run an analysis in batch mode.

- *files only (obfuscated)*

using files only (obfuscated) simulation mode, VI-CarRealTime creates a folder in the working directory, named `<event_name>.sim`, and containing all the files needed for running the simulation in [batch mode](#). For all the details about the folder content, please refer to [Files Only \(obfuscated\)](#) paragraph, while for further details on obfuscation process please refer to [Property File Obfuscation](#) topic.

- *interactive*
runs the simulation and creates all needed files for future post-processing of results (graphic models, graphic files etc.).
- *live animation*
runs the simulation and if VI-Animator is installed sends simulation results data through a socket.
- *matlab simulink*
runs the simulation in MATLAB/Simulink using the selected model in combination with the .mdl (or .m startup script file) specified in Matlab File filed.

Tire Limits

The tab container activates the possibility of enabling the computation of [tire limit states](#) for the selected event.

Files Only (obfuscated)

Files Only (obfuscated) option generates the simulation files in a separate folder. Vehicle model data are saved in a specific obfuscated file as well as all referenced TeimOrbit property files. The files are referenced in the send file (which is not obfuscated). It is then possible to run the event in [batch mode](#) by selecting <event_name>_send_svm.xml as input file. The folder is portable in a different computer.

The folder is called <event_name>_sim and it is created in the working directory. It contains the following files:

- <event_name>_model_data.xml
it is the obfuscated model file; all vehicle model data are defined in it.
- <event_name>_send_svm.xml
it is a xml file which contains all the references to analysis files (<event_name>_model_data.xml , TeimOrbit property files , model output map) that will be used by VI-CarRealTime solver during the simulation. Furthermore it stores useful information for the final user like the names of the subsystems used to generate the obfuscated model_data.xml file. <event_name>_send_svm.xml file is not obfuscated so the user can edit it and modify it according to his needs: for example it is possible to change the reference to the [rdf](#), [vdf](#), [tir](#) or [aer](#) property file to be used or modify simulation parameters as the integration time step or output step, change the [output map](#), and so on. The file can also be used as [Model input](#) for an event.
- *obfuscated property files*
an obfuscated copy of all the property files referenced in the system model is created in the simulation folder. All the property files are referenced in <event_name>_send_svm.xml file.
- *output map*
the [output map](#) xml file is copied in the simulation folder. Such file is referenced in <event_name>_send_svm.xml file.
- [vicrt_cdb.cfg](#)
a copy of the vicrt_cdb.cfg file defined in the VI-CarRealTime working directory is placed in the simulation folder. Such file is necessary during the run in [batch mode](#). In case of usage in a different computer, database paths needs to be updated accordingly.

Obfuscation Limited Support

The following events doesn't support obfuscation:

- [Path compensation](#)

- [Stability](#) ([Fishhook](#), [JTurn](#), [SineWithDwell](#))
- [StaticLapTime](#)
- [VirtualTestDrive](#)

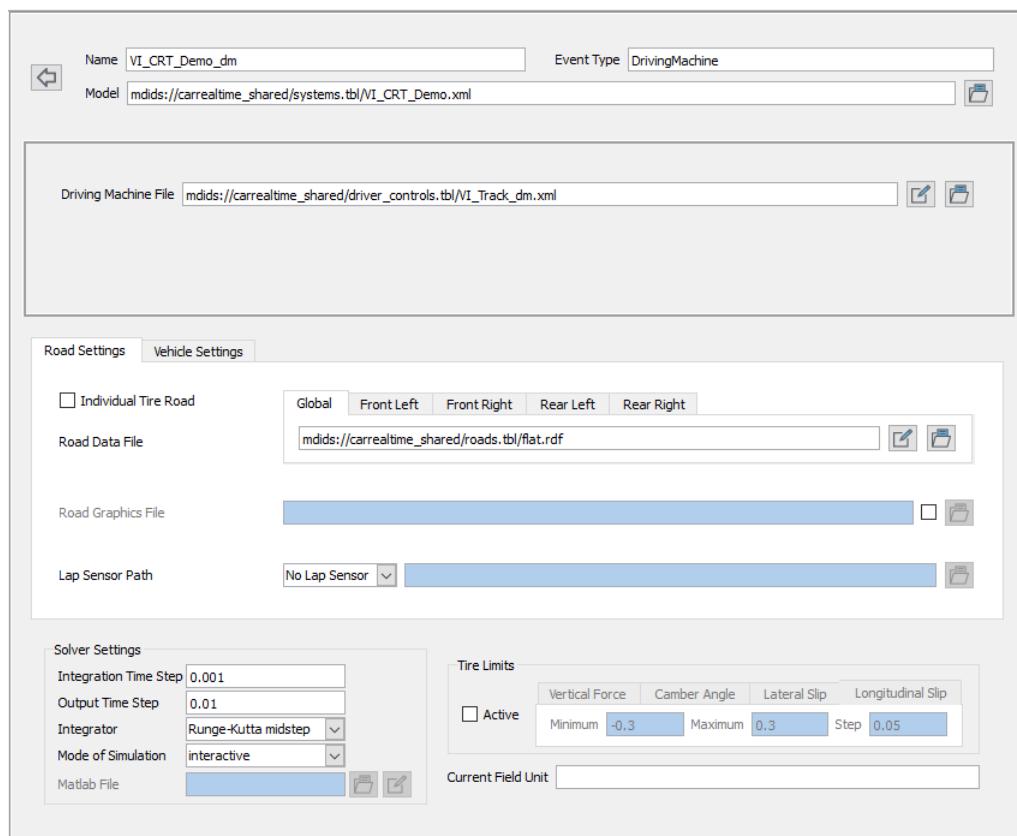
Adams Car

Driving Machine events run dynamic vehicle simulations using SmartDriver.

[DrivingMachine](#)

Driving Machine

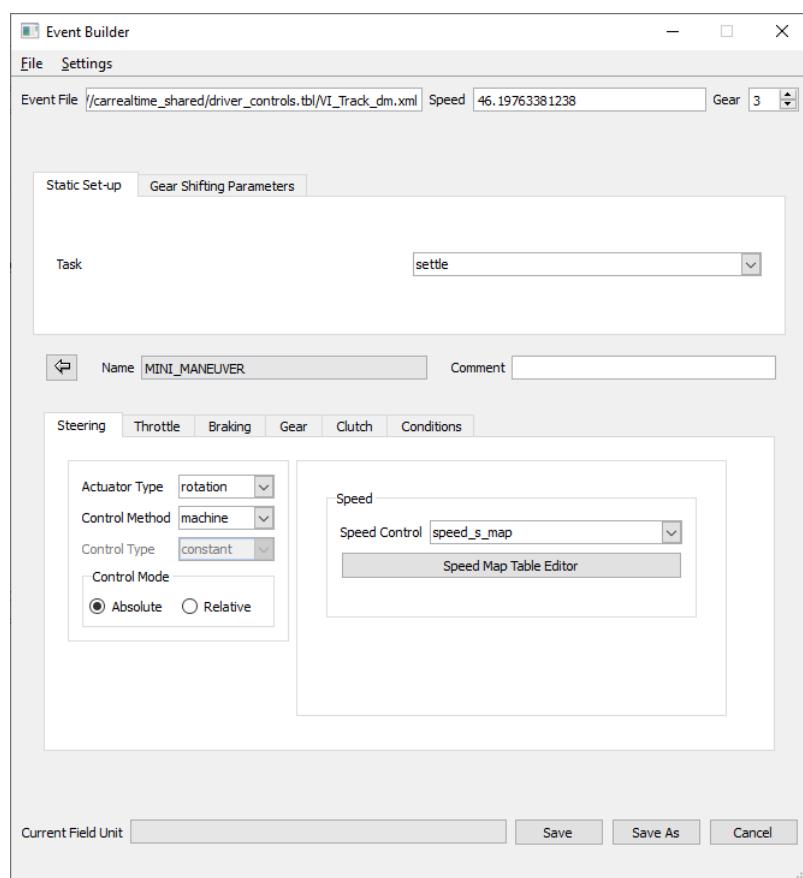
The event is controlled by a Driving Machine File (.xml or [.sdf](#)) typically stored in driver_controls table of databases.



Driving Machine File can control:

- Event initial condition;
- Static setup;
- Gear Shifting parameters;
- Mini maneuver list (multiple mini maneuvers are supported);
- Mini Maneuver properties (open loop and closed loop driving are supported but steer release maneuvers are not supported);
- End Conditions;
- Abort time.

When having xml format the Driving Machine File can be edited using the [Event Builder](#)



The default setting of the static setup is "settle" mode, which performs the static analysis before starting the dynamic run.

Static analysis is done by means of a dynamic simulation performed with no aero forces active and no action by driver. The simulation continues until the kinetic energy in the model falls below a given threshold or until a time end condition is reached.

The thresholds are controlled in the system parameter tree of the vehicle (build mode). See [Static Parameters](#) for further info.

VI-CarRealTime

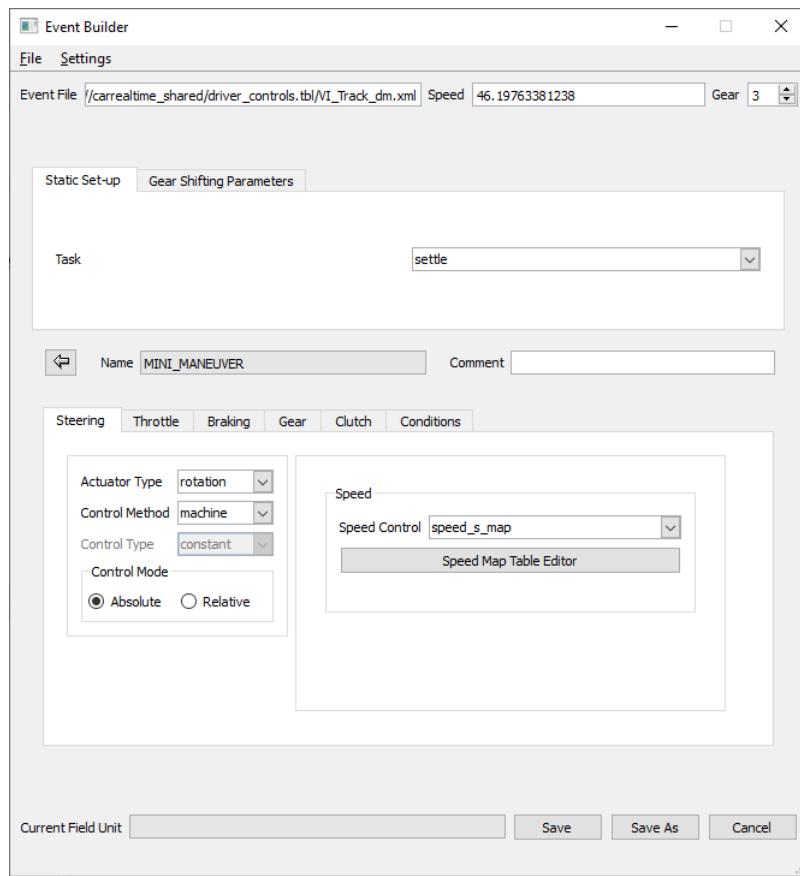
Vehicle Configuration mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml

Subsystem Definition Properties Output channels			Save All	Save System	Save System As...																																													
<div style="border: 1px solid #ccc; padding: 5px;"> custom_solver plugin VI-DriveSim driver_parameters speedgen_parameters seven_pstrig_parameters suspension_testrig_parameters live_animation_parameters solver_executive_control ABS TCS system_parameters trailer aerodynamics powertrain circular_buffer statics body_compliance setup differential anti_roll_bar auxiliary_vertical std_tire_ref dampers update_ride_height_maps g output_files external_suspension simulink path_sensor </div>																																																		
<table border="1"> <thead> <tr> <th></th> <th>Value</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>statics_abort_time</td> <td>100.0</td> <td>Abort simulation if static equilibrium is not achieved within this time</td> </tr> <tr> <td>statics_trans_speed_threshold</td> <td>0.001</td> <td>Static equilibrium is not achieved until the norm of the system translational speeds is below this value</td> </tr> <tr> <td>statics_trans_accel_threshold</td> <td>0.001</td> <td>Static equilibrium is not achieved until the norm of the system translational accelerations is below this value</td> </tr> <tr> <td>statics_angular_speed_threshold</td> <td>0.001</td> <td>Static equilibrium is not achieved until the norm of the system angular speeds is below this value</td> </tr> <tr> <td>statics_angular_accel_threshold</td> <td>0.001</td> <td>Static equilibrium is not achieved until the norm of the system angular acceleration is below this value</td> </tr> <tr> <td>statics_use_brakes</td> <td>0</td> <td>Vehicle brakes are engaged during statics when the parameter is set to 1</td> </tr> <tr> <td>statics_use_linear_damper</td> <td>0</td> <td>Linear dampers are used during statics when the parameter is set to 1</td> </tr> <tr> <td>statics_linear_damper_rate_front</td> <td>5000.0</td> <td>Front dampers damping rate used in statics</td> </tr> <tr> <td>statics_linear_damper_rate_rear</td> <td>5000.0</td> <td>Rear dampers damping rate used in statics</td> </tr> <tr> <td>statics_use_linear_spring</td> <td>0</td> <td>Linear springs are used during statics when the parameter is set to 1</td> </tr> <tr> <td>statics_linear_spring_rate_front</td> <td>100.0</td> <td>Front springs stiffness used in statics</td> </tr> <tr> <td>statics_linear_spring_rate_rear</td> <td>100.0</td> <td>Rear springs stiffness used in statics</td> </tr> <tr> <td>statics_brake_damping</td> <td>0.0</td> <td>auxiliary wheel damping used in statics</td> </tr> <tr> <td>statics_integration_time_step</td> <td>-1.0</td> <td>integration time step to be used during static/setup analysis (<=0 -> use dynamic event dt)</td> </tr> </tbody> </table>							Value	Comment	statics_abort_time	100.0	Abort simulation if static equilibrium is not achieved within this time	statics_trans_speed_threshold	0.001	Static equilibrium is not achieved until the norm of the system translational speeds is below this value	statics_trans_accel_threshold	0.001	Static equilibrium is not achieved until the norm of the system translational accelerations is below this value	statics_angular_speed_threshold	0.001	Static equilibrium is not achieved until the norm of the system angular speeds is below this value	statics_angular_accel_threshold	0.001	Static equilibrium is not achieved until the norm of the system angular acceleration is below this value	statics_use_brakes	0	Vehicle brakes are engaged during statics when the parameter is set to 1	statics_use_linear_damper	0	Linear dampers are used during statics when the parameter is set to 1	statics_linear_damper_rate_front	5000.0	Front dampers damping rate used in statics	statics_linear_damper_rate_rear	5000.0	Rear dampers damping rate used in statics	statics_use_linear_spring	0	Linear springs are used during statics when the parameter is set to 1	statics_linear_spring_rate_front	100.0	Front springs stiffness used in statics	statics_linear_spring_rate_rear	100.0	Rear springs stiffness used in statics	statics_brake_damping	0.0	auxiliary wheel damping used in statics	statics_integration_time_step	-1.0	integration time step to be used during static/setup analysis (<=0 -> use dynamic event dt)
	Value	Comment																																																
statics_abort_time	100.0	Abort simulation if static equilibrium is not achieved within this time																																																
statics_trans_speed_threshold	0.001	Static equilibrium is not achieved until the norm of the system translational speeds is below this value																																																
statics_trans_accel_threshold	0.001	Static equilibrium is not achieved until the norm of the system translational accelerations is below this value																																																
statics_angular_speed_threshold	0.001	Static equilibrium is not achieved until the norm of the system angular speeds is below this value																																																
statics_angular_accel_threshold	0.001	Static equilibrium is not achieved until the norm of the system angular acceleration is below this value																																																
statics_use_brakes	0	Vehicle brakes are engaged during statics when the parameter is set to 1																																																
statics_use_linear_damper	0	Linear dampers are used during statics when the parameter is set to 1																																																
statics_linear_damper_rate_front	5000.0	Front dampers damping rate used in statics																																																
statics_linear_damper_rate_rear	5000.0	Rear dampers damping rate used in statics																																																
statics_use_linear_spring	0	Linear springs are used during statics when the parameter is set to 1																																																
statics_linear_spring_rate_front	100.0	Front springs stiffness used in statics																																																
statics_linear_spring_rate_rear	100.0	Rear springs stiffness used in statics																																																
statics_brake_damping	0.0	auxiliary wheel damping used in statics																																																
statics_integration_time_step	-1.0	integration time step to be used during static/setup analysis (<=0 -> use dynamic event dt)																																																
<input type="text" value="Current Field Units"/>																																																		
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>																																																		

Note: During the static equilibrium preceding a dynamic analysis also performs all [vehicle](#) and [suspension](#) adjustments. If the Static setup mode is set to None, no adjustments can be completed.

Event Builder

You use the Event Builder to create or modify an XML event file. Using the Event Builder, you create a series of mini-maneuvers that define a file-driven, full-vehicle analysis. The same event files used in Adams Car can be used, too.



[Event File](#)

[Speed](#)

[Gear](#)

[Static Set-up](#)

[Gear Shifting Parameters](#)

[Current Field Unit](#)

[Mini-Maneuvers](#)

You can start **Event Builder** in

- **new file mode**

If you start the Event Builder without selecting an event file in the VI-CarRealTime Test mode interface, it appears in new file mode.

If you want to generate a new event file, from the **File** menu select **New**.

- **modify mode**

If you want to edit an existing event file, from the **File** menu, select **Open**.

The carrealtime_shared database includes some sample event files. You can use the Event Builder to modify these files or view them to understand their content. However, you can't view existing TiemOrbit form driver control files (.dcf) without first converting them to XML event files.

If you want to convert existing .dcf files to .xml, you will have to do that using Adams Car. Within Adams Car point to the **Tools** menu, point to **Database Management**, point to **Version Upgrade**, and then select **TiemOrbit->XML**.

The upper part of the Event builder is always the same, independent of which mode it is in. Here you can set the file name of the event as well as its initial conditions.

Event File

Enter a name for the event file. VI-CarRealTime automatically generates a file with extension .xml in the working directory and updates the Event File text box with the file name you entered.

Speed

Enter the initial vehicle speed for the maneuver.

Gear

Enter the initial gear position for the maneuver.

Static Set-up tab

Select the desired static setup method to be performed before the dynamic maneuver. This method is used to set up the vehicle in the desired initial conditions of lateral and longitudinal accelerations.

You can specify the following static set-up methods :

none

VI-CarRealTime does not perform a static equilibrium analysis. Rather, it locks the wheel rotations and then performs an acceleration initial-conditions analysis, which initializes VI-Tire. VI-CarRealTime then deactivates the wheel lock joint primitives before executing the first mini-maneuver.

settle

Locks the body's fore-aft, lateral, and yaw displacement using joint primitives and performs a static equilibrium to settle the vehicle on the road. Then, before executing the first mini-maneuver, VI-CarRealTime deactivates the joint primitives.

For example selecting settle and specifying an initial velocity of 27777.78 mm/s is equivalent to driving a stake vertically through the vehicle body and constraining the body to move vertically about the stake. It allows the vehicle to roll (cornering) and pitch (braking), but does not allow rotation about the axis of the stake (Yaw). When the vehicle is released from this condition, it should be relatively well balanced to remove initial transient effects. An imbalance in the tire forces could, however, cause a slight steering effect.

Gear Shifting Parameters tab

Defines the gear-shifting properties and the shape of the upshift and downshift curves for throttle and clutch signals.

The options are:

Gear Shift

Duration of the gear shifting event (> 0.0).

Clutch Raise

Delta time of the step function for the ascending curve of the clutch signal (> 0.0).

Clutch Fall

Delta time of the step function for the descending curve of the clutch signal (> 0.0).

Throttle Raise

Delta time of the step function for the ascending curve of the throttle signal (> 0.0).

Throttle Fall

Delta time of the step function for the descending curve of the throttle signal (> 0.0).

RPM Control

Enables an algorithm for RPM limiting during gear shifting.

Clutch Off Throttle On Delay

Delta time between the releasing the clutch and engaging of throttle (>0.0).

Clutch On Throttle Off Delay

Delta time between the releasing the throttle and engaging of the clutch (>0.0).

Current Field Unit

This field shows the expected unit of measurement for the field you currently have selected. For example, if you specify the vehicle speed, this text box loads the corresponding unit string [velocity (meter/s)]. The default units are SI. You can review and modify these units by selecting **Units** from the **Settings** menu, at the top of the Event Builder.

Unsupported parameters

The following are parameters not yet supported in the Event Builder, but you can modify them directly in the .xml file:

smoothingTime

Allows you to filter the output of the Machine Control at the concatenation of mini-maneuvers, therefore, helping you avoid discontinuities in the output from the Machine Control. [Real value > 0.0]

samplePeriod [DcfMachine]

Time interval between two consecutive evaluations of the controller during a closed-loop event. [Real value > 0.0]

samplePeriod [DcfSmartDriver]

Time interval between two consecutive evaluations of the controller during an VI-CarRealTime event. [Real value > 0.0]

raxSaturation

When set to true and the vehicle velocity is far from the target, the corresponding acceleration reference profile is altered to let the vehicle reach the target faster. [true, false]

tireFzControl

When set to true, the Machine Control checks if the driving tires are lifted, and in such a case, cuts throttle. [true, false]

feedForward

When set to true, an instantaneous estimation of the longitudinal control action value is calculated, improving the responsiveness of the control system. [true, false]

kickdownControl

Gear shifting method that chooses the gear that would give the maximum vehicle acceleration. [true, false]

pathDistanceComp

When set to true and if the vehicle is not exactly over the reference profile, the path distance compensation controller acts to correct the vehicle so that it is over the profile. [true, false]

pathDistanceCompTolerance

Value of the path distance under which the path distance compensation controller is activated. [Real value > 0.0]

minPreviewDistance

Minimum value of the preview distance; even if the vehicle is very slow, the preview distance does not become smaller than this value. It must be set at least as long as the vehicle wheel base. [Real value > 0.0]

activationTime

Not currently implemented. [Real value > 0.0]

The following six parameters are PID controller gains. Increasing gains generally produces a quicker controller response, but it also increases controller sensitivity to input disturbance making it unstable.

axP

Proportional gain used in the correction phase of the feed-forward calculation when following an acceleration profile. [Real value > 0.0]

axI

Integral gain used in correction phase of the feed-forward calculation when following an acceleration profile. [Real value > 0.0]

axD

Derivative gain used in correction phase of the feed forward calculation when following an acceleration profile. [Real value > 0.0]

vxP

Proportional gain used in correction phase of the feed-forward calculation when following a velocity profile. [Real value > 0.0]

vxI

Integral gain used in correction phase of the feed-forward calculation when following a velocity profile. [Real value > 0.0]

vxD

Derivative gain used in correction phase of the feed forward calculation when following a velocity profile. [Real value > 0.0]

throttleControl

When set to on, it will control the open loop throttle signal to zero during gear shifting. [on,off]

knowledge

Not currently implemented.

frontAxleCoupling

VI-CarRealTime uses this performance parameter during the calculation of the speed profile (both vehicle limits and user defined limits). This parameter affects the front axle of the vehicle. [Real value > 0.0 default 0.75]

rearAxleCoupling

VI-CarRealTime uses this performance parameter during the calculation of the speed profile (both vehicle limits and user defined limits). This parameter affects the rear axle of the vehicle. [Real value > 0.0 default 0.75]

useBrake

For staticSetup = Straight or Skidpad, setting useBrake = yes instructs the Driving Machine that it can use the brake to achieve desired longitudinal velocity and acceleration. [yes,no]

haltOnFailure

Select yes if you want to stop the simulation in case the quasi-static setup phase is not successful. [yes,no]

performLinear

Select yes if you want to use the brake subsystem to achieve the desired longitudinal and lateral acceleration. [yes,no]

yawRateCorrection

Enables the internal yaw rate controller that operates on the steering, to compensate unpredicted yaw rate events (that is, oversteering). [true, false]

pathPositioning

Defines whether the path must be positioned automatically or whether its location is absolute and must remain unchanged. [automatic, absolute, default]

A mini-maneuver is a set of smaller, simpler analysis steps, such as a straight-line mini-maneuver. Mini-maneuvers are contained in event files (.xml).

You can use two modes for defining various mini-maneuvers:

- [Table Editor](#)
- [Property Editor](#)

Table Editor Mode

To display, select the Arrow tool .

Mini-maneuvers can be added by entering a name in the Name text box and then pressing Add button at the bottom of the editor.

Active

Set a mini-maneuver to active (yes) or inactive (no). VI-CarRealTime only performs active mini-maneuvers.

Abort Time

Specify the relative time from the beginning of the mini-maneuver when the Driving Machine stops the mini-maneuver. It executes the next mini-maneuver if defined. Value must be > 0.0.

Property Editor Mode

To display, you can do either of the following:

- Double-click the mini-maneuver name;
- Right-click the mini-maneuver name and select Modify with Property Editor.

When editing a mini-maneuver in Property Editor mode, you define the five control signals (steering, brake, throttle, clutch, and gear) and the conditions that end or abort the mini-maneuver.

[Steering Tab](#)

[Throttle Tab](#)

[Braking Tab](#)

[Gear Tab](#)

[Clutch Tab](#)[Condition Tab](#)

Steering tab - Define the parameters that control the steering signal during the mini-maneuver that you are creating.

Actuator Type

When defining the steering control for a mini-maneuver, you must specify an actuator type. You use the actuator type to specify whether the Driver steers the vehicle at the steering wheel or steering rack and whether the Driver uses a force or motion. For example, when you set Actuator Type to rotation, the Driver steers using a motion on the steering wheel.

The actuator type you select for steering determines how the Driver interprets the units of other parameters associated with the steering signal. For example, if you set Actuator Type to torque, the Driver interprets the amplitude argument for an open-loop sinusoidal input as torque (with units of length*force). If you set Actuator Type to rotation, however, the Driver interprets the amplitude as an angle.

Force

Driving Machine steers the vehicle by applying a force to the steering rack.

Rotation

Driving Machine steers the vehicle using a MOTION statement on the steering-wheel revolute joint.

Torque

Driving Machine steers the vehicle by applying torque to the steering wheel.

Trans

Driving Machine steers the vehicle using a motion on the steering rack translational joint.

Note that the actuator types do not necessarily correspond to the actuators defined in your vehicle model. The Event Builder cannot browse your model and populate the Actuator Type option with the actuators actually defined in your model. In case of VI-CarRealTime models only Rotation actuation type is supported.

Control Method

When defining a mini-maneuver, you must specify a control method for each control signal. Different options appear depending on the Control Method you selected. The control methods are:

- **open** - Uses open-loop control where the control signal is a function of time. Depending on your selection of the control type, you have to specify information to define the shape of the steering control signal.
- **machine** - Uses closed-loop lateral and longitudinal controllers to follow a path and target velocity or longitudinal acceleration profiles.

For machine control, you have to select the type of closed-loop target for the appropriate application area; available options are:

- **ay_s_map** - User enters the target lateral acceleration as a function of traveled distance; data is input through a spline and edited using Ay Map Table Editor
- **ay_t_map** - User enters the target lateral acceleration as a function of time; data is input through a spline and edited using Ay Map Table Editor

- **file** - Steering demand is defined using an external xml file
- **path_map** - User enters the target path (Y vs X); data is input through a spline and edited using Path Map Table Editor
- **skidpad** - User enters steering controller data for a constant radius cornering event by specifying the radius, turn direction and entry distance to the skidpad
- **straight** - No parameters are required by the user.

Control Type

When using the open control method you must select a control type. Either one of the function based control types:

- **constant** - sets the steering at a constant value
- **step** - will change the steering demand during a given duration to set a Final Value beginning at a given Start time
- **ramp** - changes the steering demand at constant Ramp Value per second beginning at a given Start time
- **impulse** - impulsively changes the steering demand for a Duration while peaking in the Maximum value and beginning at a given Start time
- **sine** - gives a single sinusoidal steering demand change beginning at a given Start time and with the given Amplitude and Cycle length
- **swept_sine** - gives a sinusoidal steering demand beginning at gives a sine shaped steering demand beginning at a given Start Time with an Amplitude. The frequency can be changed from an Initial Frequency to a Max Frequency at a given Frequency Rate.

or the data map driven Data_map control type in which case you must specify an a data map using the steering open loop demand editor.

Control Mode

If you set Control Method to open, you must also define the Control Mode as either absolute or relative. If Control Method is not open, Control Mode is always absolute. With all open-loop maneuvers, the final value of the preceding mini-maneuver is used as the initial value for the next mini-maneuver. This is irrespective of whether you set Control Mode to relative or absolute. For example, your vehicle is driving on a skid pad at 30 mph, with 20° of steering wheel angle, when the first mini-maneuver is finished. The steering-wheel angle for the next mini-maneuver will be 20°.

The following Control Mode options are only available in the constant and step control type during an open loop maneuver:

- **Absolute** - Indicates that the final value is absolute. For example, for a step input to the steering where the initial steering is 10 degrees and the final value is 50 degrees, the steer at the end of the step equals 50 degrees.
- **Relative** - Indicates that the final value is relative to the initial value. For example, for a step input to the steering where the initial steering is 10 degrees and the final value is 50 degrees, the steer at the end of the step equals 60 degrees.

Throttle/Braking/Gear/Clutch tabs

The tabs are used to setup the properties of the specific driver signal listed by the tab name. They all affect the behavior of vehicle longitudinal controller.

Control Method

When defining a mini-maneuver, you must specify a control method for each control signal. Different options appear depending on the Control Method you selected. The control methods are:

- **open** - Uses open-loop control where the control signal is a function of time. Depending on your selection of the **control type**, you have to specify information to define the shape of the steering control signal.
- **machine** - Uses closed-loop lateral and longitudinal controllers to follow a path and target velocity or longitudinal acceleration profiles.

For **machine** control, you have to select the type of closed-loop target for the appropriate application area; Available options are:

- **ax_s_map** - User enters the target longitudinal acceleration as a function of traveled distance; data is input through a spline and edited using Ax Map Table Editor
- **ax_t_map** - User enters the target longitudinal acceleration as a function of time; data is input through a spline and edited using Ax Map Table Editor
- **file** - longitudinal controller is actuated using an external xml file
- **lat_accel** - longitudinal controller is defined using a constant lateral acceleration target.
- **lon_accel** - longitudinal controller is defined using a constant longitudinal acceleration target and a start time for the controller activity.
- **maintain** - target longitudinal velocity is set to a constant value.
- **speed_s_map** - User enters the target velocity as a function of path s; data is input through a spline and edited using Speed Map Table Editor
- **speed_t_map** - User enters the target longitudinal velocity as a function of time; data is input through a spline and edited using Speed Map Table Editor
- **vel_polynomial** - longitudinal velocity target is set using a polynomial.

The Driving Machine computes the speed using the following relation:

```
IF (Time < START_TIME):
    SPEED = VELOCITY
    IF ( TIME > START_TIME ):
        SPEED = VELOCITY +
        ACCELERATION*(TIME - START_TIME) +
        1/2*JERK*(TIME-START_TIME)**2
```

where START_TIME is the starting time relative to the beginning of the mini-maneuver.

User specifies the following arguments:

```
VELOCITY = value <length/time>
ACCELERATION = value <length/time2>
JERK = value <length/time3>
START_TIME = value <time>
```

Control Type

When using the open control method you must select a control type. Either one of the function based control types:

- **constant** - sets the steering at a constant value
- **step** - will change the steering demand during a given **duration** to set a **Final Value** beginning at a given **Start time**
- **ramp** - changes the steering demand at constant **Ramp Value** per second beginning at a given **Start time**
- **impulse** - impulsively changes the steering demand for a **Duration** while peaking in the **Maximum value** and beginning at a given **Start time**.
- **sine** - gives a single sinusoidal steering demand change beginning at a given **Start time** and with the given **Amplitude** and **Cycle length**.
- **swept_sine** - gives a sinusoidal steering demand beginning at a given **Start Time** with an **Amplitude**. The frequency can be changed from an **Initial Frequency** to a **Max Frequency** at a given **Frequency Rate**.

or the data map driven **Data_map** control type in which case you must specify an a data map using the specific (throttle, brake, gear, clutch) **open loop demand** editor.

Control Mode

If you set Control Method to open, you must also define the Control Mode as either absolute or relative. If Control Method is not open, Control Mode is always absolute. With all open-loop maneuvers, the final value of the preceding mini-maneuver is used as the initial value for the next mini-maneuver. This is irrespective of whether you set Control Mode to relative or absolute.

The following Control Mode options are only available in the constant and step control type during an open loop maneuver:

- **Absolute** - Indicates that the final value is absolute. For example, for a step input to the steering where the initial steering is 10 degrees and the final value is 50 degrees, the steer at the end of the step equals 50 degrees.
- **Relative** - Indicates that the final value is relative to the initial value. For example, for a step input to the steering where the initial steering is 10 degrees and the final value is 50 degrees, the steer at the end of the step equals 60 degrees.

These tabs contain the similar options as the [Steering tab](#), except that the information applies to the particular signal.

Conditions tab

Specify a number of conditions for each mini-maneuver. Conditions can be added, deleted and changed in two modes:

- **Table Editor**

Add or delete end conditions to the mini-maneuver you are creating. To display, select the Arrow button



The Table Editor mode gives you access to the complete set of parameters.

Note that the conditions do not necessarily correspond to the condition defined in your vehicle model.

This is an extensive list but the Event Builder cannot browse your model and populate the Type option menu with the condition actually defined in your model.

- **Property Editor**

Define parameters for the condition. To display, you can do either of the following:

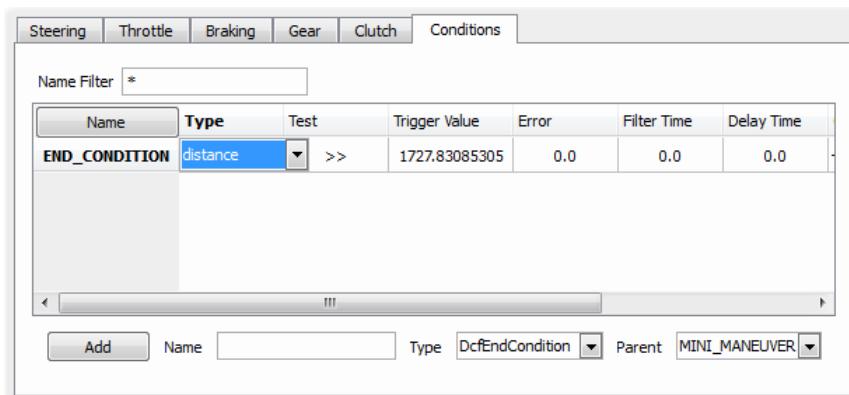
- Double-click the mini-maneuver name
- Right-click the mini-maneuver name and select Modify with Property Editor

Specifying End Conditions

Conditions specify when one mini-maneuver ends so the next one can begin. For example, you might end a mini-maneuver when the vehicle speed reaches 100 kph. You can also group end conditions together. For example, you might end a mini-maneuver when the vehicle speed reaches 100 kph and the lateral acceleration exceeds 5 m/s².

The event file supports conditions based on time, distance, velocity, acceleration, and many other vehicle control variables. Conditions reference a measure or solver variable by name to measure a given quantity in the model.

In **Table Editor mode**, you can see all defined conditions, add new conditions, modify conditions, or delete conditions. The Table Editor mode also lets you see the full set of arguments available for each condition.



In the above example, the end condition is satisfied if:

$$5 - 0.1 < \text{Lateral acceleration} < 5 + 0.1$$

- **Filter Time**

The test must be satisfied continuously over the filter time to satisfy the end condition, it must be positive.

- **Delay Time**

Once the end condition is satisfied, delay the end of the mini-maneuver by delay time.

- **Group Name**

You specify a name to group conditions together. All conditions having the same group name must be satisfied simultaneously to end a mini-maneuver. For example, you might specify two end conditions:

- Longitudinal velocity equal to 20 m/s
- Lateral acceleration greater than 5 m/s²

Then you place the specified end conditions in the group mygroup. To end the mini-maneuver, the longitudinal velocity must be 20 m/s and the lateral acceleration must be greater than 5 m/s² at the same time.

- **Condition Type**

Select a condition type:

- **abort**

When met, it causes the simulation to stop.

- **end**

When met, it causes the simulation to proceed to the next mini-maneuver, if one is defined.

VIDriver

The available events are:

- [Cornering Events](#)
- [Course Events](#)
- [File Driven](#)
- [Open Loop Steering Events](#)
- [Stability](#)
- [Straight Line Events](#)

Note: refer to [VI-Driver Theory](#) chapter to understand how VI-Driver algorithm works.

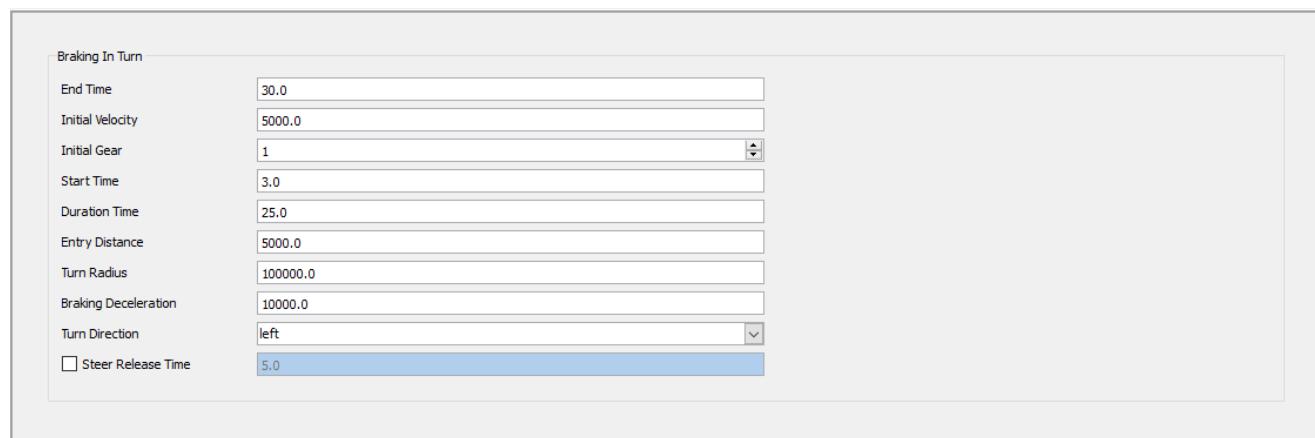
Cornering Events

This category contains predefined events to run the most common cornering maneuvers. User customizes the maneuver specifying only a predefined set of parameter. All events in this category are performed using VI-Driver.

Available events classes are:

- [Braking In Turn](#)
- [Constant Radius Cornering](#)

Braking In Turn events run dynamic vehicle simulations by using VI-Driver.



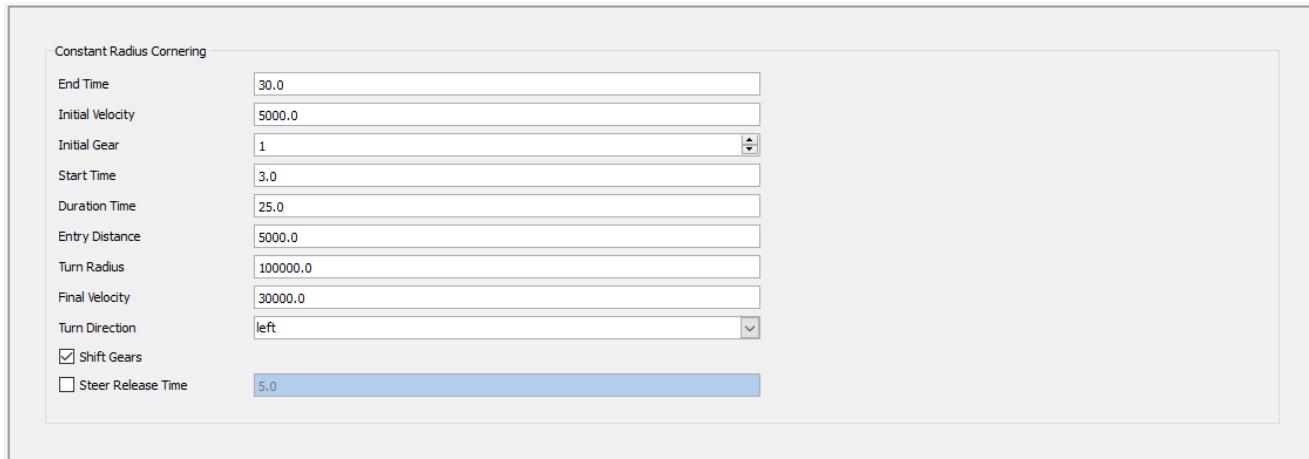
User can modify all the following parameters of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
initial time speed ramp.

VI-CarRealTime

- **Duration Time**
duration of the speed ramp.
- **Entry Distance**
beginning distance for the curved section.
- **Turn Radius**
constant radius of the curved section.
- **Braking Deceleration**
slope of the speed ramp.
- **Turn direction**
direction toward which the vehicle turns (left/right).
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

Constant Radius Cornering events run dynamic vehicle simulations using VI-Driver.



User can modify the following parameters of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Duration Time**
duration of the speed ramp.
- **Entry Distance**
beginning distance for the curved section.
- **Turn Radius**
constant radius of the curved section.
- **Final Velocity**
final value for the speed ramp.
- **Turn direction**
direction toward which the vehicle turns (left/right).

- **Shift gears**

fixed gear or machine controlled gear switch.

- **Steer Release Time**

when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

CourseEvents

This unit embeds the following events:

- [Max Performance](#)
- [Path Compensation](#)
- [Press Maneuvers](#)

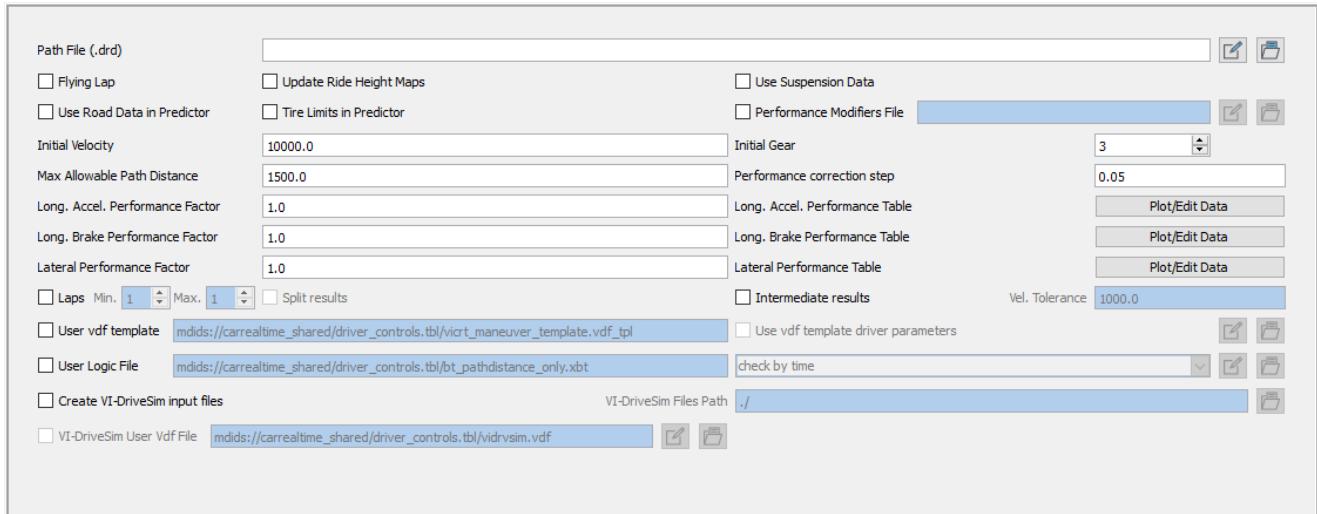
MaxPerformance event in VI-CarRealTime is used to define dynamic limit velocity profile on a given track.

The event uses a specific static solver (VI-SpeedGen) to compute a velocity profile and then it uses the dynamic solver to verify if the computed speed profile is feasible. If the velocity path is unfeasible (maximum path distance exceeded, zero velocity etc.) the solver turns back to the static and computes a new velocity profile, modifying only the portion nearby the unfeasible point. The process is iterated until a feasible velocity profile is found.

Basically the vehicle model used for static prediction has no suspensions and inherits all properties from the full VI-CarRealTime model.

The effect of aero forces is considered and the effect of suspension jounce is taken into account by the presence of ride height maps.

The figure below shows the property editor for this event class.



The available parameters for the MaxPerformance event are:

- **Path File**

specifies the path that the vehicle has to follow. It can be visualized and edited in VI-Road using the button beside the fields.

- **Initial Velocity**

initial velocity for the simulation (disabled when activating flying lap option).

- **Initial Gear**

initial gear for the simulation (disabled when activating flying lap option).

- **Flying Lap**

computes velocity profile imposing that final velocity is the same of initial velocity.

- **Update Ride Height Maps**

[ride height maps](#) are data that can be optionally stored in the vehicle model (body subsystem); they are used by the static solver to compute the dependency of Ride Height (and Aero Forces) on vehicle velocity, since the vehicle model in the static solver has no suspensions. These maps can be generated or updated before running the Max Performance event. To create them a coast down dynamic simulation is run starting from a high velocity with open clutch on a straight track. The simulation continues until the vehicle velocity, due to aero drag, reaches a given value.

Note: the parameters to control the dynamic ride height map "pre-phase" are shown in vehicle properties [System Parameters-Update ride height maps](#).

Once the pre-phase simulation is run, the maps are present in memory, but not stored into [body subsystem](#) file.

To visualize (if not present) and save them, the procedure is:

1. Go in vehicle system parameters.
2. Modify a parameter.
3. Click apply.
4. Reset the parameter to original value if needed.

Ride height map instance should now be visible in body subsystem property editor under Auxiliary Spline Data Tab.

- **Create intermediate result file**

Dump the result file at the end of each iteration.

- **Max Path Distance**

Maximum allowable path distance that makes a section and velocity profile unfeasible.

- **Correction step**

Size of the correction step used during the optimization.

- **Long. Accel. Performance Factor**

Initial guess of vehicle longitudinal performance while accelerating.

- **Long. Brake Performance Factor**

Initial guess of vehicle longitudinal performance while braking.

- **Lateral Performance Factor**

Initial guess of vehicle lateral performance.

- **Laps**

Check box enables multiple lap functionality; it is possible to set a minimum and maximum number of laps that are to be executed. Simulation will be stopped if velocity difference among current and previous lap is included in specified tolerance. Specific vdf template is needed for multiple lap simulation (example file named `vicrt_maneuver_multilap_template.vdf_tpl` is included in `carrealtime_shared` database).

- **Min**

Minimum number of laps to be executed; velocity tolerance check is disabled until minimum number of laps is exceeded.

- **Max**

Maximum number of laps to be executed during multi lap simulation.

- **Split results**

Check box enables the creation of separate result files for the single lap (multi-lap simulation).

- **Vel. tolerance**

Vehicle velocity tolerance among subsequent laps; simulation is stopped if velocity difference falls in the specified tolerance.

- **User vdf template**

A template file used to customize the driver parameters.

The template file is a standard vdf file with some specific placeholders used to the simulation parameters during the optimization:

- `>>REPLACE_INIT_SPEED<<` , used to set the initial speed;
- `>>REPLACE_INIT_GEAR<<` , used to set the initial gear;
- `>>REPLACE_ABORT_TIME<<` , used to set the maneuver abort time;
- `>>REPLACE_DRD_FILE<<` , used to set the name of the drd file containing the reference path and the calculated speed profile.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'VDF'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'

[...]

$-----STARTUP
[STARTUP]
STARTING_TIME = 0
INITIAL_SPEED = >>REPLACE_INIT_SPEED<<
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRAKING = 0
INITIAL_BRAKING2 = 0
INITIAL_GEAR = >>REPLACE_INIT_GEAR<<
INITIAL_CLUTCH = 0
INITIAL_SETUP = 'STANDARD'
$-----MANEUVERS_LIST
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'MANEUVER_1' >>REPLACE_ABORT_TIME<< 0.01
$-----MANEUVER_1
[MANEUVER_1]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'

[...]

(MACHINE)
PATH = 'MANEUVER_1_PATH'
SPEED = 'MANEUVER_1_SPEED'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'TRUE'
```

```

SPEED_MAP_ABS_TIME = 'FALSE'
SPEED_MAP_GLOBAL_S = 'FALSE'
CONNECT_PATH = 'TRUE'
END_OF_PATH_CHECK = 'TRUE'
PATH_POSITIONING = 'ABSOLUTE'
$-----MANEUVER_1_PATH
[MANEUVER_1_PATH]
TYPE = 'DRD_FILE'
INTERPOLATION = 'CUBIC_SPLINE'
FILE_NAME = '>>REPLACE_DRD_FILE<<'
$-----MANEUVER_1_SPEED
[MANEUVER_1_SPEED]
TYPE = 'DRD_FILE'
FILE_NAME = '>>REPLACE_DRD_FILE<<'
```

- **User Logic File**

A file containing the custom [correction logic](#). The feature is available when the related license key is checked-out. Please refer to [Licenses](#) topic for the details.

- **Create VI-DriveSim input file**

The creation of a VI-DriveSim input file related to the event.

- **Use Road Data File**

when the toggle button is checked, the user is allowed to enter a Road Data File ([rdf](#)) which will be used by VI-SpeedGen to compute the static speed profile. If the toggle button is unchecked, a flat road with constant unitary friction coefficient will be used by the VI-SpeedGen solver.

- **Tire Limits in Predictor**

when the flag is checked, the [Tire Limits](#) tool is used to evaluate the force envelope produced by each tire. Such envelopes are used by VI-SpeedGen solver to compute the maximum lateral and longitudinal acceleration of the simplified vehicle; otherwise an approximated force ellipsoid is used to evaluate the maximum pure longitudinal and lateral forces. In general the Tire Limits option allows VI-SpeedGen solver to estimate speed profiles closer to the dynamic simulation ones (results produced by VI-Drive using the vdf generated by the StaticLapTime event), thus reducing the overestimation of the speed profile.

- **Performance Modifiers File**

when the flag is checked, it is possible to specify a [PMF](#) file to alter aerodynamic and electric motor performance as a function of path_s.

- **Use Suspension Data**

when the flag is checked, the simulation relies on VI-SpeedGenEvo internal model to compute the static speed profile. In such case [Update Ride Height Maps](#) toggle button is disabled since the evo model doesn't use it. In such condition, before VI-SpeedGen solver is called, a suspension testrig analysis (*tr* suffix) is performed in order to extract the following suspension curves for front and rear suspension:

1. Base vs. Jounce
2. Camber vs. Jounce
3. Track vs. Jounce
4. Side View Angle vs. Jounce
5. Vertical Force vs. Jounce
6. AntiRollForce vs. Delta Jounce (left - right)

The curves are then passed to VI-SpeedGenEvo to compute the vehicle ground stiffness.

At the end of the simulation an event file is created in the working directory (event_name_mxp.vdf). That event can be used to run a standard VIDriver event.

The save and restore functionality allows the restart of the optimization process from the last complete iteration. To do this the VI-Driver/MaxPerformance tool keeps an history file with the data of the previous complete iterations; that file can be used to resume the optimization process.

VI_CRT_Demo_vdr.bat	3/21/2012 11:59 AM	Windows Batch File	1 KB
VI_CRT_Demo_vdr.mxp	3/21/2012 11:59 AM	MXP File	2 KB
VI_CRT_Demo_vdr.vdf	3/21/2012 5:17 PM	VDF File	4 KB
VI_CRT_Demo_vdr_send_svm.xml	3/21/2012 11:59 AM	XML Document	618 KB
VI_CRT_Demo_vdr_solver_svm.log	3/21/2012 5:18 PM	Text Document	2 KB
VI_CRT_Demo_vdr_work_history.xsh	3/21/2012 5:17 PM	XSH File	7 KB

ViDriver/MaxPerformance output files

It is possible to resume the optimization process from the last valid iteration, by calling from a dos shell the following command:

```
event_name_vdr.bat event_name_work_history.xsh
```

It is also possible to drag and drop the history file on the .bat file directly.

To run a co-simulation using a MaxPerformance event, the following steps are needed:

1. In the VI-CarRealTime GUI, create a MaxPerformance event in "files only" mode.
2. Open MATLAB in the current working directory.
3. Generate the files mpx_initialize.m, mpx_runmaneuver.m, mpx_terminate.m in the working directory.
4. Run the event co-simulation using the vimaxperf_mtl command.

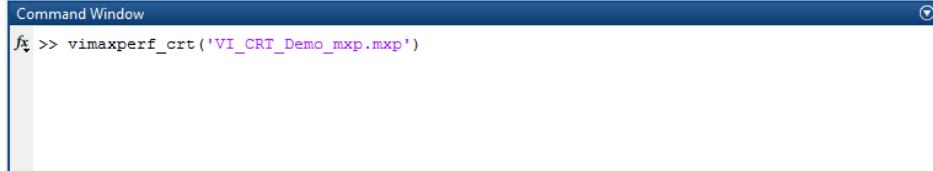
The vimaxperf_mtl command take as input the event file name (either the **event_name_mxp_send_svm.xml** or **event_name_vdr.mxp**) and optionally the result files output prefix and the partial history file name :

```
vimaxperf_mtl('event_name_mxp_send_svm.xml',[event_name_work_history.xsh])
vimaxperf_mtl({'event_name_mxp_send_svm.xml','output_files_prefix'},[event_name_work_history.xsh])
```

or

```
vimaxperf_mtl('event_name_mxp.mxp',[event_name_work_history.xsh])
vimaxperf_mtl({'event_name_mxp.mxp','output_files_prefix'},[event_name_work_history.xsh])
```

In order to use the MaxPerformance, the Simulink model must be instrumented using an interface block. The "MaxPerformance interface" block can be inserted anyway inside the model and is part of the "VI-CarRealTime library".



Run MaxPerformance from matlab

The VI-Driver/MaxPerformance optimization procedure interacts with the MATLAB/Simulink environment using three callback functions that are called at each iteration:

- `mxp_initialize.m`: called at the beginning of each iteration. The function receives the model file name and must return a success flag (0/1). The purpose of this function is to contain all the initialization commands needed by the external controllers placed in Simulink model.

```
function ret = mxp_initialize( filename )

ret = 1;

return
```

- `mxp_runmaneuver.m`: called for running the simulink model. The function receives receives the `output_prefix` and must return a success flag (0/1). This function is used to actually run the co-simulation, and must contain at least a "sim" command referencing the Simulink model.

```
function ret = mxp_runmaneuver( output_prefix )

ret = 1;

sim('active_vehicle');

return
```

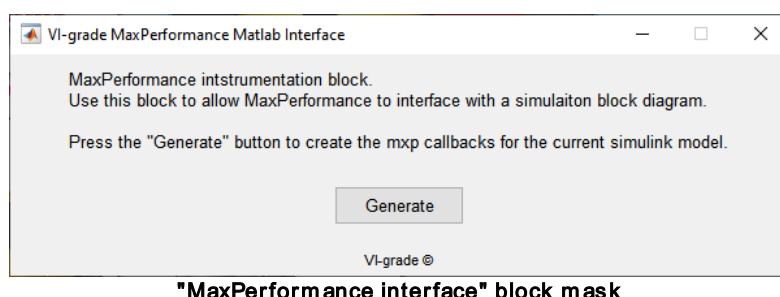
- `mxp_terminate.m`: called at the end of each iteration. The function receives a dummy value and must return a success flag (0/1). The purpose of this function is to contain all the termination commands needed by the external controllers placed in Simulink model.

```
function ret = mxp_terminate( val )

ret = 1;

return
```

The previous callback could be automatically generated for the Simulink model using the "Generate" button placed into the mask of the "MaxPerformance interface" block.

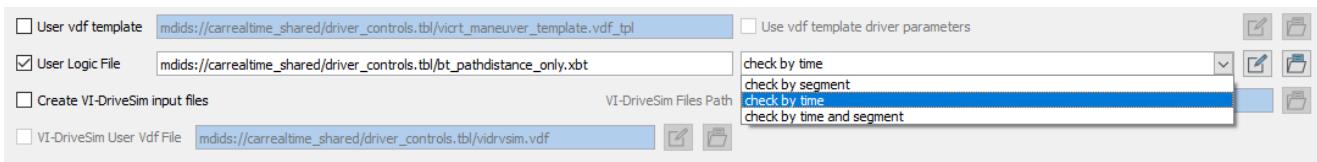


It is possible to use a custom optimization logic by checking the "User Logic file" check button. Two fields become active:

- a text box pointing to the [correction logic file](#);
- a combo box representing the type of check performed.

While simulation is running, the custom optimization logic is executed according to the combo box selection:

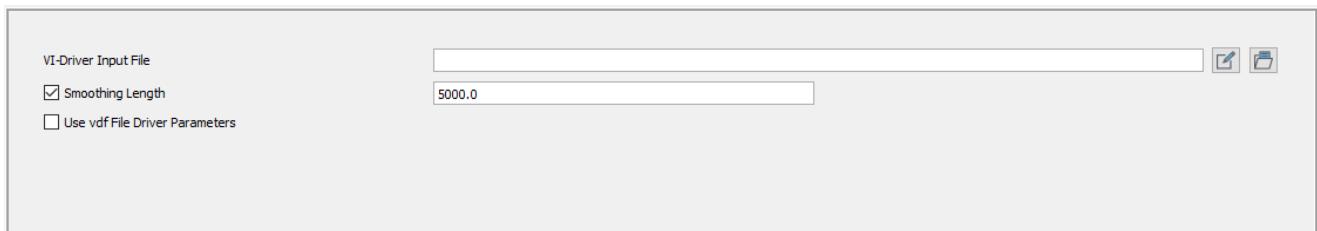
- **check by time:** check every two seconds of simulation time;
- **check by segment:** check at the end of each segment;
- **check by time and segment:** check every two seconds and at the end of each segment.



Path Compensation event is used to compensate the tracking error of a dynamic analysis (made with VI-Driver). The event uses a vdf, which must reference a drd file both for path and speed profile.

The event consists of the following steps:

- a first dynamic simulation using the selected vdf is performed when submitting the event;
- a compensated drd file with a new trajectory, called <model_name>_ptc_path_comp.drd , is created in the working directory: the internal algorithm creates a new trajectory (new drd file) in order to minimize the path distance obtained in the previous analysis;
- a new vdf, called <model_name>_ptc.drd is stored in the working directory: the vdf is the same as the one selected by the user in the GUI, except the references to the path and speed profile: it refers to the new drd file stored in your working directory;
- a new analysis with the latter vdf file is submitted.



Smoothing Length

It represents the length of the blending curve between the original path and the modified path.

Use vdf File Driver Parameters

When the flag is enabled, the driver uses the parameters embedded in the vdf file rather than the ones stored in the model.

How to use the tool at best:

- the first analysis should be done using a preview time greater than 0.5s (suggested value PT=0.7s), in order to obtain a smooth steering angle profile signal, even if the path distance might become bigger than when using the default PT: the path distance will be reduced in the second analysis (using the compensated DRD file);
- if the original DRD file doesn't start with a straight line segment, it's suggested to use a low starting speed value;
- it is recommended to set the key [STEERING_PDC_ACTIVE](#) = 'FALSE' to prevent conflicts between the path compensation routine and the VI-Driver internal path distance compensation module.

Press Maneuvers event is used to simulate different types of maneuvers.

The purpose of the event is to use VI-Driver to have the vehicle going through the cones at maximum speed available.

The event is not expected to have the vehicle following a specific trajectory precisely.

The reference driver line is simply used to create the desired history of vehicle dynamics.

Given the vehicle data and event type, cones are placed in the track and a genetic algorithm is used to generate target trajectories which will be submitted and verified during the dynamic integration, having a constant vehicle velocity as longitudinal dynamics target. For each iteration, cone interference check is used to verify if the target trajectory is feasible.

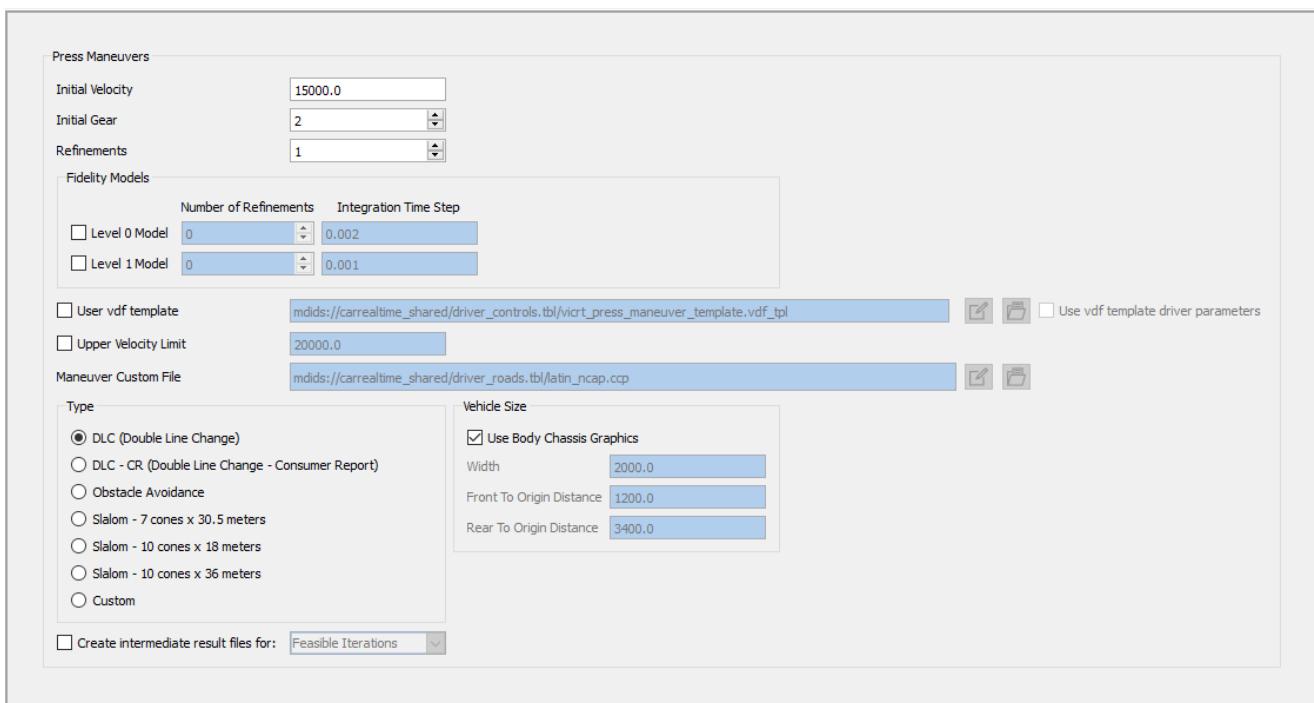
When the full set of trajectories have been verified, velocity is increased and a new cycle begins. Depending on the required refinements, a different number of generation is used by the genetic algorithm involved in the trajectory creation process.

The event is completed when a single combination of trajectory and velocity (maximum) is identified featuring a no cone hit situation.



The key aspects of the event are:

- *Learning* : collect data from vehicle dynamics.
- *Adapting* : using information coming from previous trials, updating the trajectory and pushing the speed.



The parameters of the events are:

- **Initial Velocity**

Initial velocity for first iteration.

- **Initial Gear**

Event gear for first iteration.

- **Refinements**

Specify the number of generations to be used by the genetic algorithm.

- **Fidelity Models**

It is possible to run the first n refinements of a Press Maneuver simulation by using a simplified version of the model. Activating these flags allows to speed up the course event maneuver, since the simulation with the simplified model is much faster. For each level the user must specify the **Number of Refinements** and the **Integration Time Step** to be used.

The following model level can be enabled:

- **Level 0 Model:** maximum model simplification (it corresponds to [simplified_model](#) flag equals to 1)
- **Level 1 Model:** kinematic version of the model (it corresponds to [simplified_model](#) flag equals to 2)

Note: the sum of the refinements defined in the Fidelity Models section must be always minor than the total number of Refinements, because the last refinement of the press maneuver event will be always run with the full version (no model simplification) of the VI-CarRealTime Model.

Note: a GUI warning message is thrown whenever the refinements condition is not respected.

- **User vdf template**

Toggle button which enables the usage of a vdf template file (vdf_tpl) for customizing the driver parameters and tasks. The user is allowed to tailor the vdf_tpl with his own requirements.

The template file is a standard vdf file with some specific placeholders used as simulation parameters:

- >>REPLACE_INIT_SPEED<< , used by press maneuver event to set the initial speed;
- >>REPLACE_INIT_GEAR<< , used by press maneuver event to set the initial gear;
- >>REPLACE_ABORT_TIME<< , used by press maneuver event to set the maneuver abort time;
- >>REPLACE_DRD_FILE<< , used by press maneuver event to set the name of the drd file containing the reference path and the calculated speed profile.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'VDF'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'

[...]

$-----STARTUP
[STARTUP]
STARTING_TIME = 0
INITIAL_SPEED = >>REPLACE_INIT_SPEED<<
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRAKING = 0
INITIAL_BRAKING2 = 0
INITIAL_GEAR = >>REPLACE_INIT_GEAR<<
INITIAL_CLUTCH = 0
INITIAL_SETUP = 'STANDARD'
$-----MANEUVERS_LIST
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'MANEUVER_1' >>REPLACE_ABORT_TIME<< 0.01
$-----MANEUVER_1
[MANEUVER_1]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'

[...]

(MACHINE)
PATH = 'MANEUVER_1_PATH'
SPEED = 'MANEUVER_1_SPEED'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'TRUE'
SPEED_MAP_ABS_TIME = 'FALSE'
SPEED_MAP_GLOBAL_S = 'FALSE'
CONNECT_PATH = 'TRUE'
END_OF_PATH_CHECK = 'TRUE'
PATH_POSITIONING = 'ABSOLUTE'
$-----MANEUVER_1_PATH
[MANEUVER_1_PATH]
TYPE = 'DRD_FILE'
INTERPOLATION = 'CUBIC_SPLINE'
FILE_NAME = >>REPLACE_DRD_FILE<<
$-----MANEUVER_1_SPEED
[MANEUVER_1_SPEED]
TYPE = 'DRD_FILE'
FILE_NAME = >>REPLACE_DRD_FILE<<
```

- **Upper Velocity Limit**

toggle button which allows to define an upper velocity limit in press maneuver event.

- **Use vdf template driver parameters**

toggle button with which the user choices to feed VI-Driver with the driver parameters stored in the vdf template file or with the parameters stored in the [model system tree](#).

- **Create intermediate results file for**

when the toggle button is checked, intermediate results file are saved in the working directory, otherwise they are deleted. It is possible to keep intermediate result files for:

- Feasible Iterations (only)
- All Iterations

Type

The following maneuvers are available:

- [DLC \(Double Line Change\) - ISO 3888-1](#)
- [DLC CR \(Double Line Change Consumer Report\)](#)
- [Obstacle Avoidance - ISO 3888-2](#)
- [Slalom \(7x30.5 - 10x18 - 10x36\)](#)
- [Custom \(cones defined by the user\)](#)

Vehicle Size

Check **Use Body Chassis Graphic** box to automatically compute vehicle size from the graphic shell, otherwise the following parameters can be used to define the vehicle size:

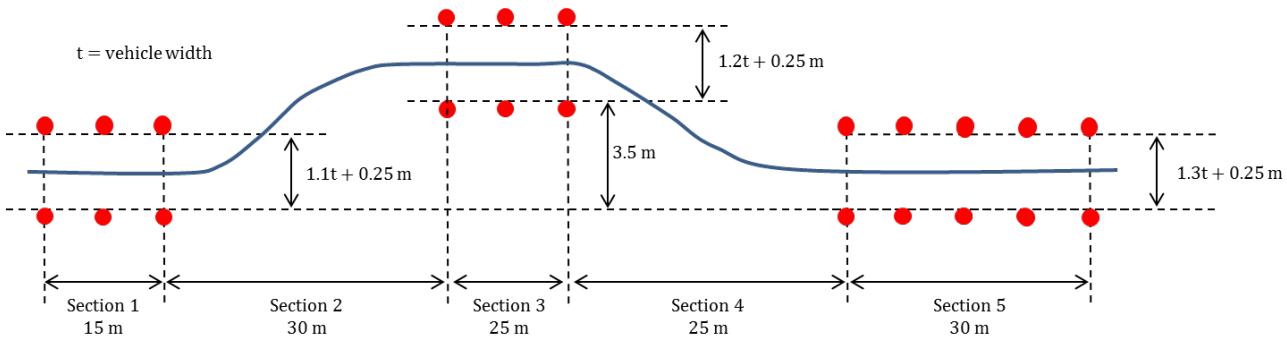
- Width
- Front To Origin Distance (distance of the front edge of the chassis measured with respect to the origin of the [Vehicle Reference System](#))
- Rear To Origin Distance (distance of the rear edge of the chassis measured with respect to the origin of the [Vehicle Reference System](#))

At the end of the simulation a VI-Driver input file is created to give the user the capability of running the final event out of iterative procedure.

The event features automatic graphical representation of cones, for simulation postprocessing (live or review mode).



In **Double Lane Change** maneuver, 11 pairs of cones are placed along the road according to ISO 3888-1:



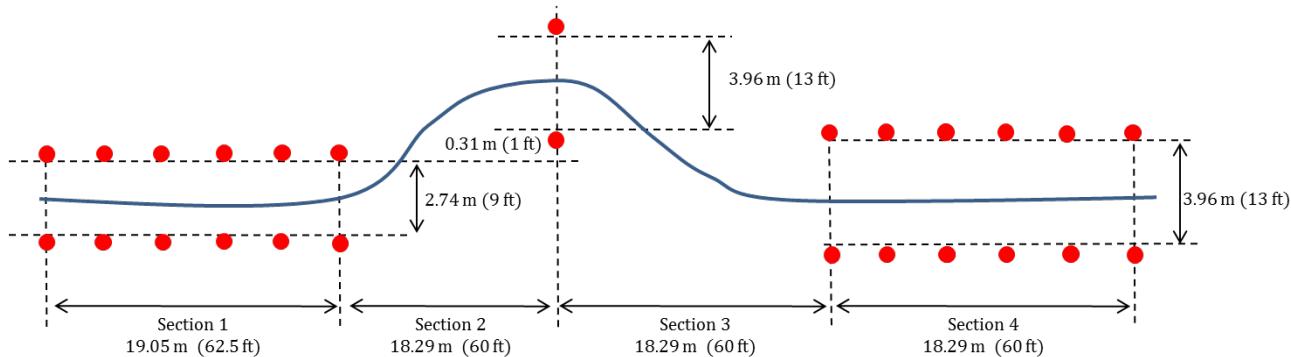
The real width between each pair of cones is corrected with the cone radius (0.092 m).

An example of cones X and Y location is shown below (assuming $V_{track} = 2$ m).

```
=====
=          PRESS MANEUVER           =
=          CONES DISPLACEMENT      =
=          MANEUVER TYPE          =
=          dlc                      =
=====
=  ID   =  CONE LEFT [m]    =  CONE RIGHT [m]   =  RADIUS [m]  =
=====
=  1    =  ( 30.000,   1.375)  =  ( 30.000, -1.375)  =  0.092  =
=====
=  2    =  ( 37.500,   1.375)  =  ( 37.500, -1.375)  =  0.092  =
```

```
= 3 = ( 45.000, 1.375) = ( 45.000, -1.375) = 0.092 =
= 4 = ( 75.000, 5.075) = ( 75.000, 2.125) = 0.092 =
= 5 = ( 87.500, 5.075) = ( 87.500, 2.125) = 0.092 =
= 6 = (100.000, 5.075) = (100.000, 2.125) = 0.092 =
= 7 = (125.000, 1.775) = (125.000, -1.375) = 0.092 =
= 8 = (140.000, 1.775) = (140.000, -1.375) = 0.092 =
= 9 = (155.000, 1.775) = (155.000, -1.375) = 0.092 =
= 10 = (170.000, 1.775) = (170.000, -1.375) = 0.092 =
= 11 = (185.000, 1.775) = (185.000, -1.375) = 0.092 =
=====
```

In **Double Lane Change - Consumer Report** maneuver, 13 pairs of cones are placed along the road as follows:

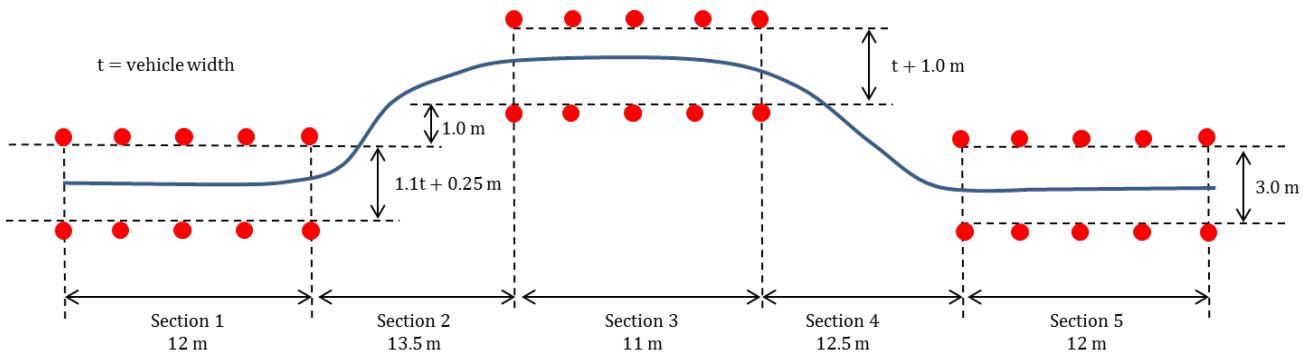


The real width between each pair of cones is corrected with the cone radius (0.092 m).

An example of cones X and Y location is shown below (assuming $V_{track} = 2$ m).

```
=====
= PRESS MANEUVER =
= CONES DISPLACEMENT =
= MANEUVER TYPE =
= dlc_consumer_reports =
=====
= ID = CONE LEFT [m] = CONE RIGHT [m] = RADIUS [m] =
=====
= 1 = ( 30.000, 1.370) = ( 30.000, -1.370) = 0.092 =
= 2 = ( 33.810, 1.370) = ( 33.810, -1.370) = 0.092 =
= 3 = ( 37.620, 1.370) = ( 37.620, -1.370) = 0.092 =
= 4 = ( 41.430, 1.370) = ( 41.430, -1.370) = 0.092 =
= 5 = ( 45.240, 1.370) = ( 45.240, -1.370) = 0.092 =
= 6 = ( 49.050, 1.370) = ( 49.050, -1.370) = 0.092 =
= 7 = ( 67.340, 5.640) = ( 67.340, 1.680) = 0.092 =
= 8 = ( 85.630, 1.980) = ( 85.630, -1.980) = 0.092 =
= 9 = ( 89.288, 1.980) = ( 89.288, -1.980) = 0.092 =
= 10 = ( 92.946, 1.980) = ( 92.946, -1.980) = 0.092 =
= 11 = ( 96.604, 1.980) = ( 96.604, -1.980) = 0.092 =
= 12 = (100.262, 1.980) = (100.262, -1.980) = 0.092 =
= 13 = (103.920, 1.980) = (103.920, -1.980) = 0.092 =
=====
```

In **Obstacle Avoidance** maneuver, 15 pairs of cones are placed along the road according to **ISO 3888-2**:



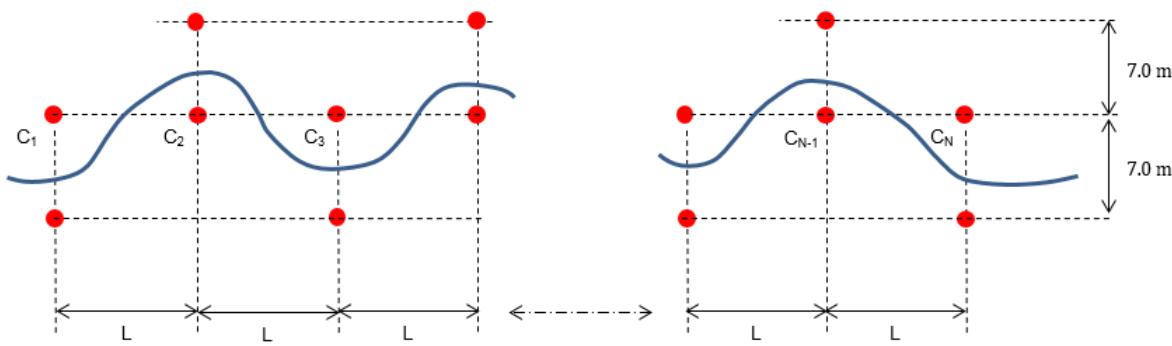
The real width between each pair of cones is corrected with the cone radius (0.092 m).

An example of cones X and Y location is shown below (assuming $V_{track} = 2 \text{ m}$).

```
=====
=           PRESS MANEUVER
=           CONES DISPLACEMENT
=           MANEUVER TYPE
=           obstacle_avoidance
=====
=   ID   =   CONE LEFT [m]   =   CONE RIGHT [m]   =   RADIUS [m] =
=====
=   1   =   ( 30.000,  1.375)   =   ( 30.000, -1.375)   =   0.092   =
=   2   =   ( 33.000,  1.375)   =   ( 33.000, -1.375)   =   0.092   =
=   3   =   ( 36.000,  1.375)   =   ( 36.000, -1.375)   =   0.092   =
=   4   =   ( 39.000,  1.375)   =   ( 39.000, -1.375)   =   0.092   =
=   5   =   ( 42.000,  1.375)   =   ( 42.000, -1.375)   =   0.092   =
=   6   =   ( 55.500,  5.375)   =   ( 55.500,  2.075)   =   0.092   =
=   7   =   ( 58.250,  5.375)   =   ( 58.250,  2.075)   =   0.092   =
=   8   =   ( 61.000,  5.375)   =   ( 61.000,  2.075)   =   0.092   =
=   9   =   ( 63.750,  5.375)   =   ( 63.750,  2.075)   =   0.092   =
=  10  =   ( 66.500,  5.375)   =   ( 66.500,  2.075)   =   0.092   =
=  11  =   ( 79.000,  1.375)   =   ( 79.000, -1.925)   =   0.092   =
=  12  =   ( 82.000,  1.375)   =   ( 82.000, -1.925)   =   0.092   =
=  13  =   ( 85.000,  1.375)   =   ( 85.000, -1.925)   =   0.092   =
=  14  =   ( 88.000,  1.375)   =   ( 88.000, -1.925)   =   0.092   =
=  15  =   ( 91.000,  1.375)   =   ( 91.000, -1.925)   =   0.092   =
=====
```

Three **Slalom** types are available, they have different number of cones and different longitudinal distance between two consecutive cones (see the figure below):

- 7 cones with distance 30.5 meters ($N = 7, L = 30.5$)
- 10 cones with distance 18 meters ($N = 10, L = 18$)
- 10 cones with distance 36 meters ($N = 10, L = 36$)



An example of cones X and Y location for the slalom with 7 cones is shown below (assuming $V_{track} = 2 \text{ m}$).

```
=====
=           PRESS MANEUVER          =
=           CONES DISPLACEMENT      =
=           MANEUVER TYPE          =
=           slalom                  =
=====
=   ID   =   CONE LEFT [m]   =   CONE RIGHT [m]   =   RADIUS [m] =
=====
=   1   =   ( 40.000,  1.550)   =   ( 40.000, -5.450)   =   0.092   =
=   2   =   ( 70.500,  8.550)   =   ( 70.500,  1.550)   =   0.092   =
=   3   =   (101.000,  1.550)   =   (101.000, -5.450)   =   0.092   =
=   4   =   (131.500,  8.550)   =   (131.500,  1.550)   =   0.092   =
=   5   =   (162.000,  1.550)   =   (162.000, -5.450)   =   0.092   =
=   6   =   (192.500,  8.550)   =   (192.500,  1.550)   =   0.092   =
=   7   =   (223.000,  1.550)   =   (223.000, -5.450)   =   0.092   =
=====
```

The **Custom Maneuver** can be set through a specific property file (see *latin_ncap.cpp* in the VI-CarRealTime database as example).

The content of the file includes the position of the cones, the list of the control points which determine the base trajectory, some data used to build the control points grid and other auxiliary parameters.

- **CONES** table

It contains the centers of the cones ordered along the x-axis (vehicle advancing direction). The y-coordinates must be entered first for the left cones and then for the right ones, considering that the y-axis is oriented towards the left side.

- **CONTROL_POINTS** table

Adopting the same coordinate reference of the cones, the table is used to list the x-y pairs of the control points that define the shape of the base trajectory. Starting from this base set of control points, a grid is constructed to determine the full set of perturbed trajectories.

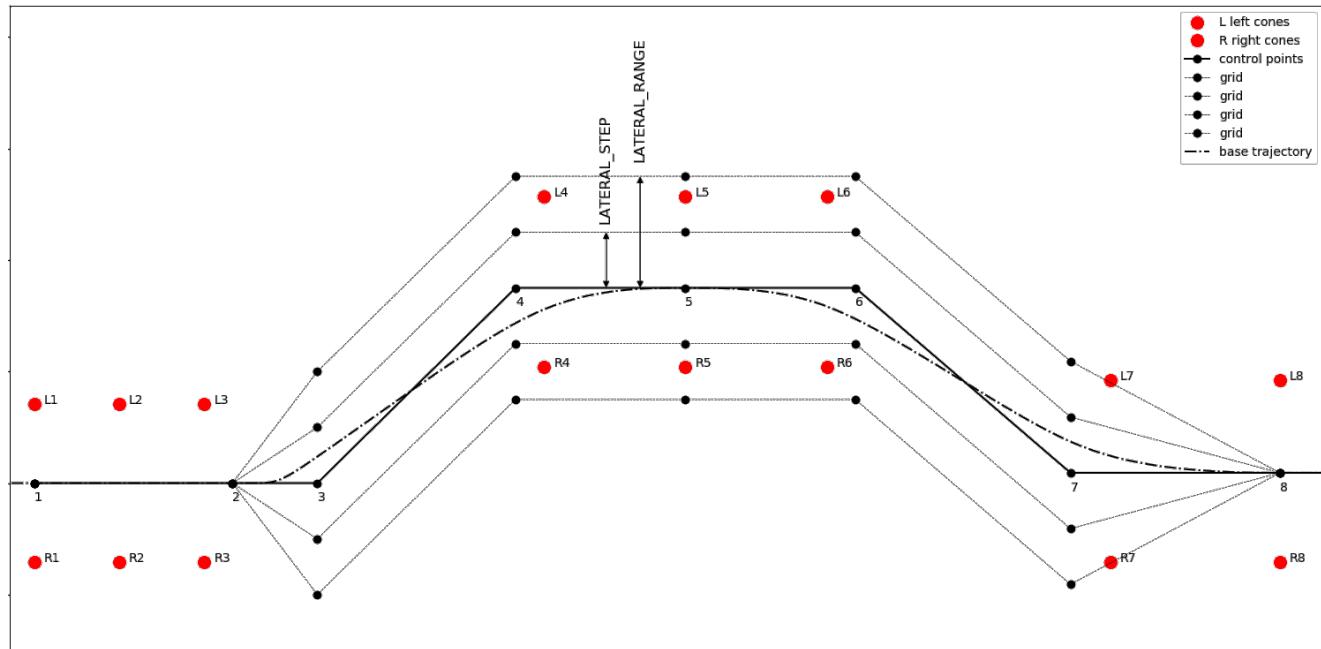
- **GENERAL** data

- **LATERAL_RANGE**: maximum lateral distance of the perturbed control point (default 2 meters)
- **LATERAL_STEP** : lateral step of the grid (default 1 m)
- **SPEED_STEP** : velocity increment after a feasible trajectory (default 0.5 m/s)
- **BEGIN_CONTROL_POINT** : index of the first control point to be perturbed (default 1)
- **END_CONTROL_POINT** : index of the last control point to be perturbed (default is the last control point of the list)
- **SETTLING_DISTANCE** : longitudinal distance that the vehicle must travel before the first cone (default 30 m)

VI-CarRealTime

- *ABORT_LENGTH* : longitudinal length that determines the end of the maneuver (default is the position of the last cone)
- *ABORT_TIME* : progressive time that determines the end of the maneuver (default 100 s)

The figure below illustrates the concept of base trajectory and grid generation (*BEGIN_CONTROL_POINT* = 3 and *END_CONTROL_POINT* = 7) and highlights the meaning of *LATERAL_RANGE* and *LATERAL_STEP*.



To run a co-simulation using a PressManeuvers event, the following steps are needed:

1. In the VI-CarRealTime GUI, create a PressManeuvers event in "files only" mode.
2. Open MATLAB in the current working directory.
3. Generate the files *isolc_initialize.m*, *isolc_runmaneuver.m*, *isolc_terminate.m* in the working directory.
4. Run the event co-simulation using the *vipressmaneuvers_crt* command.

The *vipressmaneuvers_crt* command take as input the event file name (**event_name_pm_send_svm.xml**) and optionally the result files output prefix:

```
vipressmaneuvers_crt('event_name_pm_send_svm.xml')
vipressmaneuvers_crt({'event_name_pm_send_svm.xml','output_files_prefix'})
```

In order to use the PressManeuvers, the Simulink model must be instrumented using an interface block. The "PressManeuvers interface" block can be inserted anyway inside the model and is part of the "VI-CarRealTime library".



Run PressManeuvers from matlab

The VI-Driver/PressManeuvers optimization procedure interacts with the MATLAB/Simulink environment using three callback functions that are called at each iteration:

- isolc_initialize.m: called at the beginning of each iteration. The function receives the model file name and must return a success flag (0/1). The purpose of this function is to contain all the initialization commands needed by the external controllers placed in Simulink model.

```
function ret = isolc_initialize( filename )  
  
ret = 1;  
  
return
```

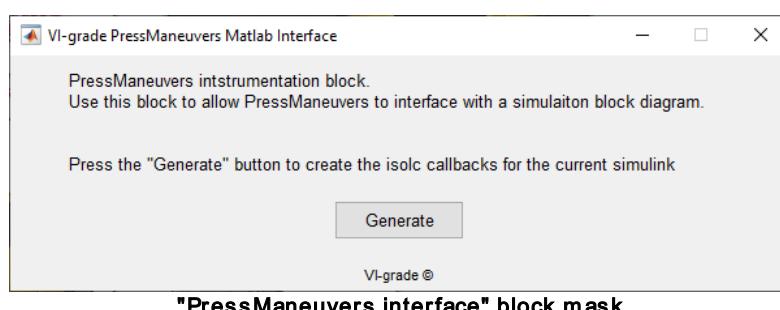
- isolc_runmaneuver.m: called for running the simulink model. The function receives the output_prefix and must return a success flag (0/1). This function is used to actually run the co-simulation, and must contain at least a "sim" command referencing the Simulink model.

```
function ret = isolc_runmaneuver( output_prefix )  
  
ret = 1;  
  
sim('active_vehicle');  
  
return
```

- isolc_terminate.m: called at the end of each iteration. The function receives a dummy value and must return a success flag (0/1). The purpose of this function is to contain all the termination commands needed by the external controllers placed in Simulink model.

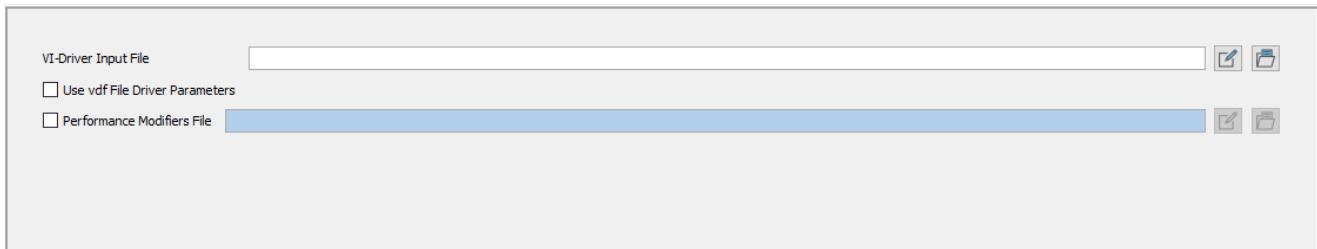
```
function ret = isolc_terminate( val )  
  
ret = 1;  
  
return
```

The previous callback could be automatically generated for the Simulink model using the "Generate" button placed into the mask of the "PressManeuvers interface" block.



FileDriven

FileDriven events run dynamic vehicle simulations using [VI-Driver](#).



The event is controlled by a [vdf file](#), which is typically stored in the `driver_controls` table of databases.

Use vdf File Driver Parameters

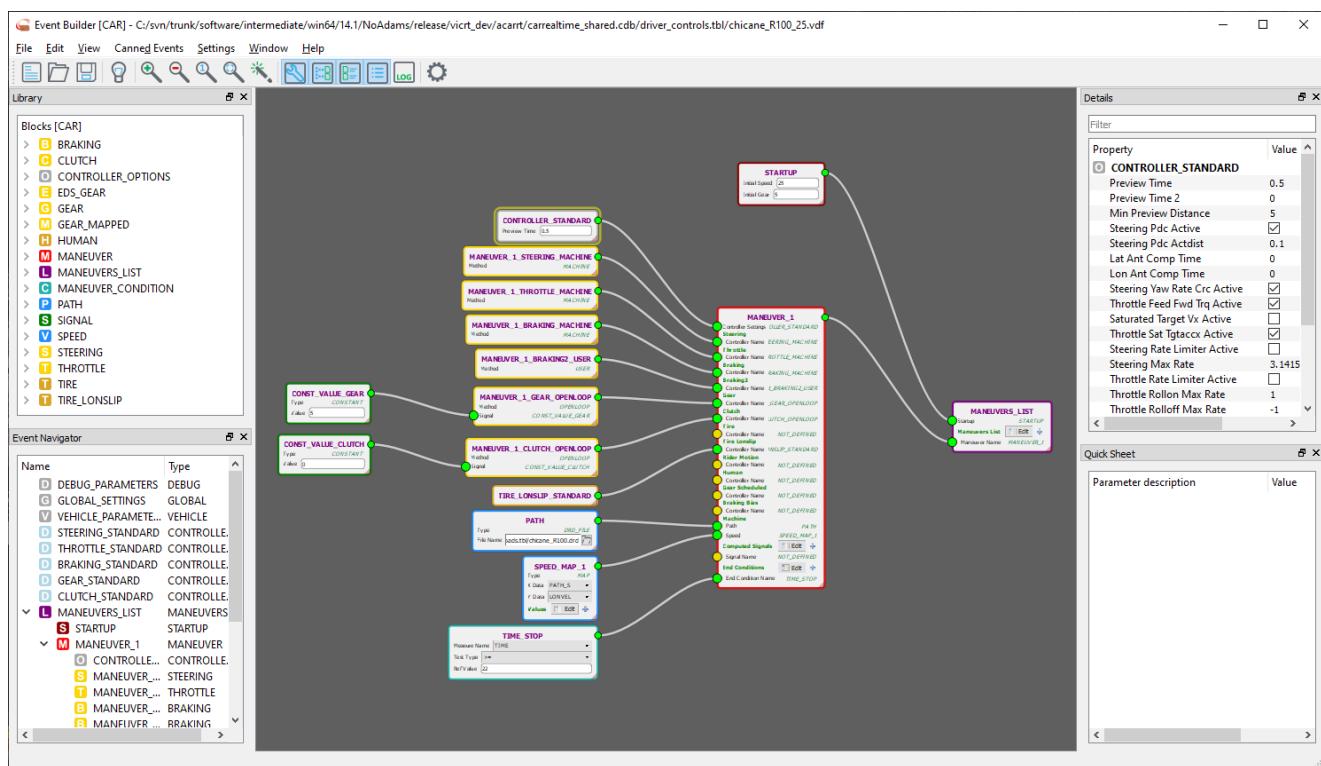
When the flag is enabled, the driver uses the parameters embedded in the vdf file rather than the ones stored in the model. In particular, the parameters are:

- preview time
- anticipatory compensation
- DT1
- DT2
- gear shifting time
- clutch raise time
- clutch fall time
- throttle raise time
- throttle fall time
- upshift rpm for each gear
- downshift rpm for each gear

Performance Modifiers File

When the flag is enabled, it is possible to specify a [PMF](#) file to alter aerodynamic and electric motor performance as a function of path_s.

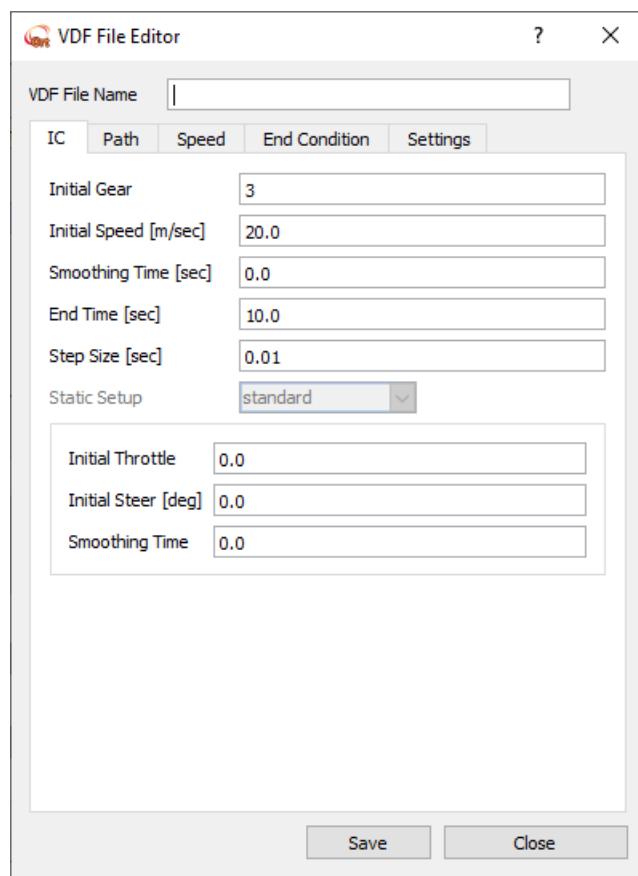
By pushing the  button, it is possible to open VI-EventBuilder in order to modify the selected vdf or to create one from scratch (if VI-Driver Input File field is empty).



Note: for further information about VI-EventBuilder please refer to its documentation.

Learn more about [VI-Driver input file](#).

The old panel used to generate vdf file can be opened by selecting *Utilities -> Create vdf file ...* in VI-CarRealTime main GUI.



Open Loop Steering Events

This category contains predefined events to run the most common open loop steering maneuvers. User customize the maneuver specifying only a predefined set of parameter. All events in this category are performed using VI-Driver.

Available events classes are:

- [Impulse Steer](#)
- [Ramp Steer](#)
- [Sine Steer](#)
- [Step Steer](#)
- [Swept Steer](#)

Impulse Steer events run dynamic vehicle simulations using VI-Driver.

Impulse Steer	
End Time	10.0
Initial Velocity	5000.0
Initial Gear	1
Start Time	1.0
Impulse Duration	1.0
Initial Steer	0.0
Impulse Amplitude	30.0
<input checked="" type="checkbox"/> Cruise control	
<input type="checkbox"/> Steer Release Time	5.0

User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Impulse Duration**
duration of the steering impulse.
- **Initial Steer**
initial value of the steering wheel angle.
- **Impulse amplitude**
constant radius of the curved section.
- **Cruise controls**
toggle activity of cruise control.
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

Ramp Steer events run dynamic vehicle simulations using VI-Driver.

Ramp Steer	
End Time	10.0
Initial Velocity	5000.0
Initial Gear	1
Start Time	1.0
Ramp Duration	1.0
Initial Steer	0.0
Slope	10.0
<input checked="" type="checkbox"/> Cruise control	
<input type="checkbox"/> Steer Release Time	5.0

User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Ramp Duration**
duration of the ramp action.
- **Initial Steer**
initial value of the steering wheel angle at the beginning of the step.
- **Slope**
Steering wheel angle rate.
- **Cruise controls**
toggle activity of cruise control.
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

Sine Steer events run dynamic vehicle simulations using VI-Driver.

The screenshot shows a configuration dialog for a "Sine Steer" event. It contains the following fields and their values:

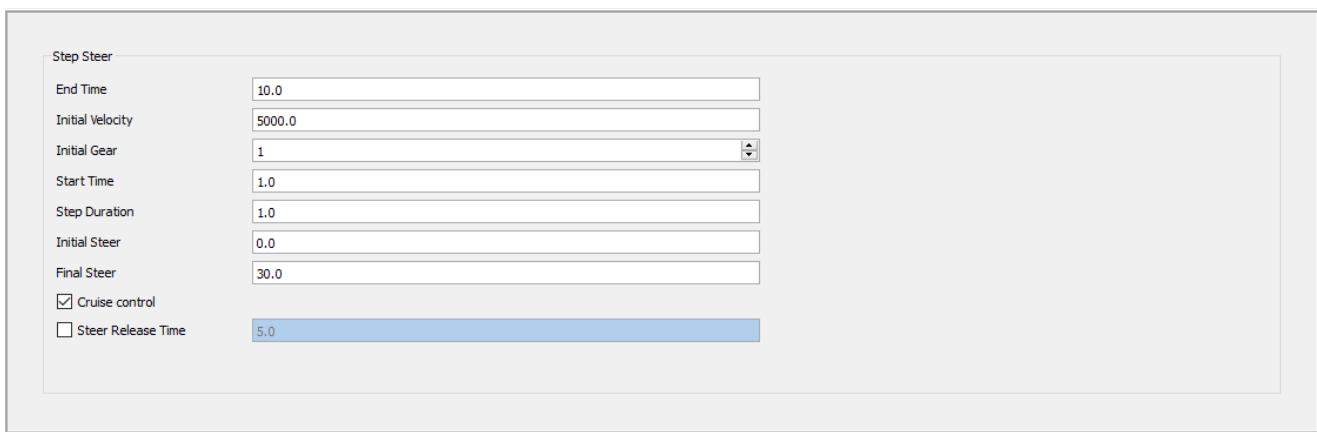
End Time	10.0
Initial Velocity	5000.0
Initial Gear	1
Start Time	1.0
Initial Steer	0.0
Sine Amplitude	30.0
Sine Frequency	0.5
Sine Phase	0.0
<input checked="" type="checkbox"/> Cruise control	
<input type="checkbox"/> Steer Release Time	5.0

User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Initial Steer**
initial value of the steering wheel angle.
- **Sine Amplitude**
amplitude of the sine shape of the steering wheel.

- **Sine Frequency**
frequency of the sine function.
- **Sine Phase**
phase of the sine function.
- **Cruise controls**
toggle activity of cruise control.
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

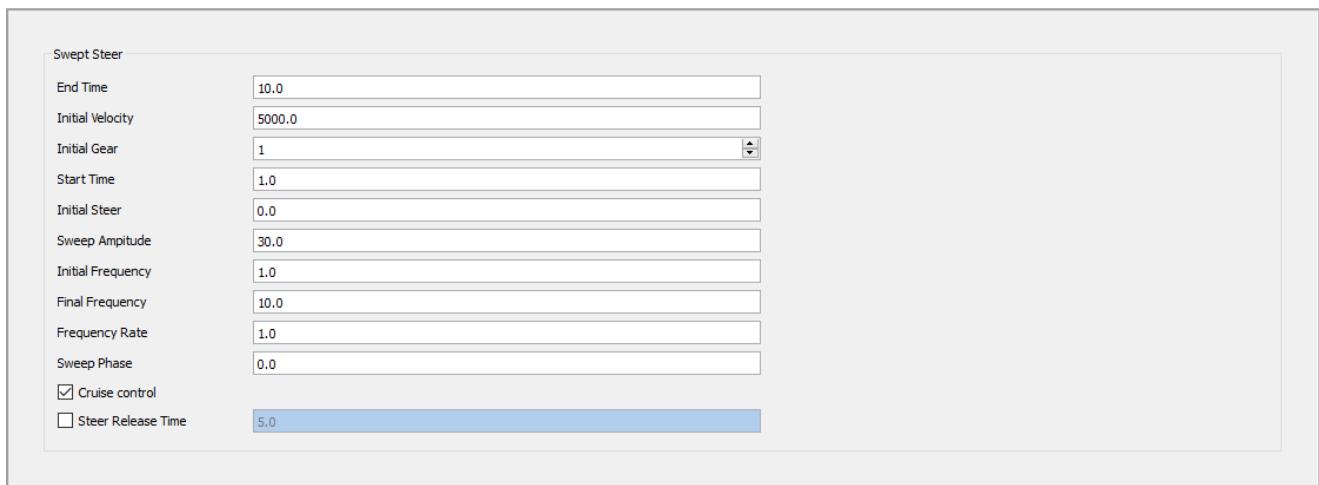
Step Steer events run dynamic vehicle simulations using VI-Driver.



User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Step Duration**
duration of the step action.
- **Initial Steer**
initial value of the steering wheel angle at the beginning of the step.
- **Final Steer**
final value of the steering wheel angle at the end of the step.
- **Cruise controls**
toggle activity of cruise control.
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

Swept Steer events run dynamic vehicle simulations using VI-Driver.



User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Start Time**
time at which the steering action begin.
- **Initial Steer**
initial value of the steering wheel angle at the beginning of the sweep action.
- **Sweep Amplitude**
amplitude of the sine shapes of the steering wheel.
- **Initial Frequency**
initial frequency (Hz) of the sine shape of the steering wheel.
- **Final Frequency**
final frequency (Hz) of the sine shape of the steering wheel.
- **Frequency Rate**
rate of frequency of the sine shapes of the steering wheel.
- **Sweep Phase**
phase of the sine shapes of the steering wheel.
- **Cruise controls**
toggle activity of cruise control.
- **Steer Release Time**
when toggle is checked, this field sets the global simulation time at which VI-Driver releases the steering wheel angle.

Stability

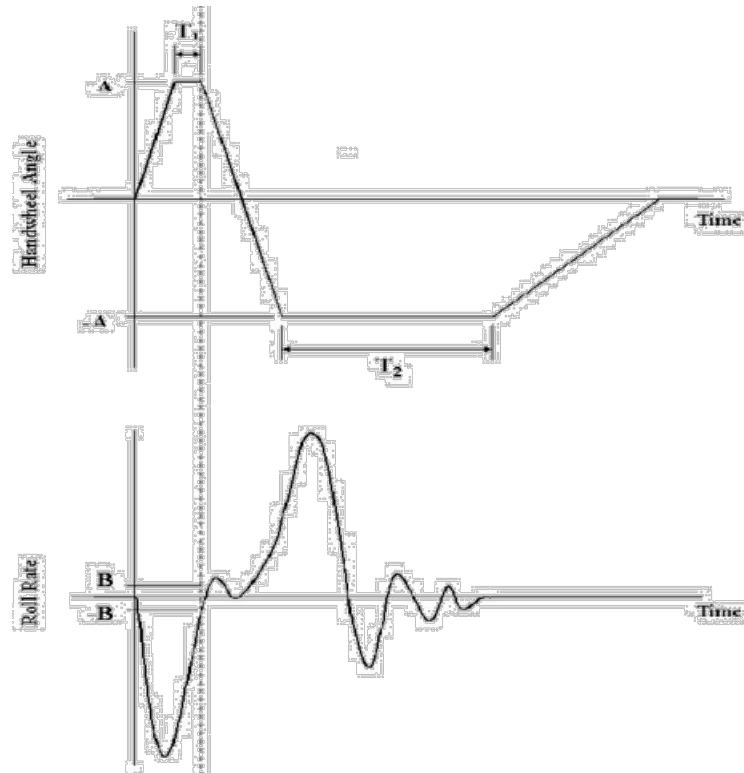
The following events are available:

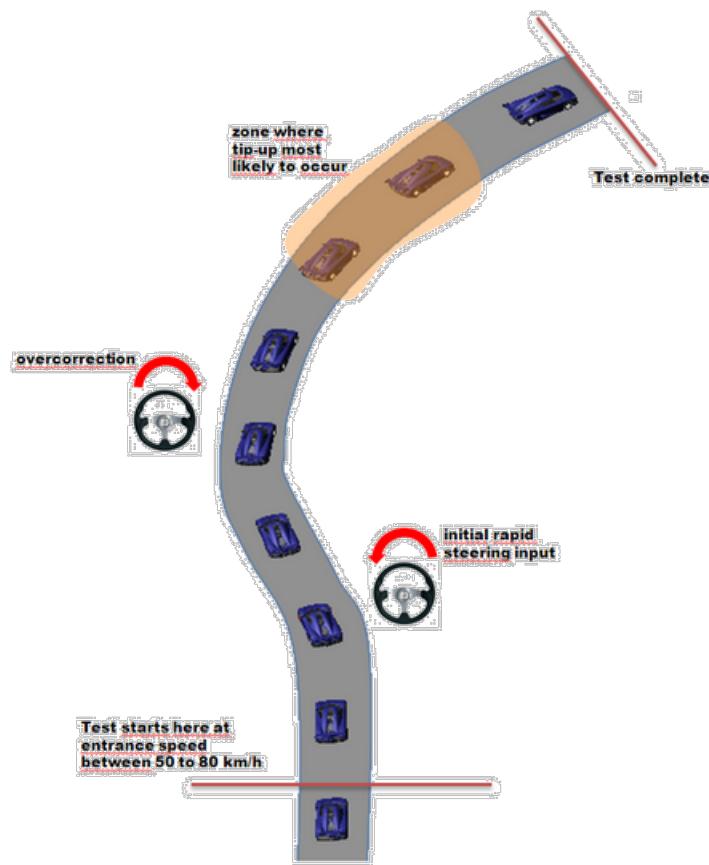
- [Fishhook](#)
- [JTurn](#)
- [SineWithDwell](#)

The Fishhook maneuver is a test that stresses the rollover tendency of vehicles by inducing it with a typical emergency maneuver.

The name of the event is given by the shape that the trajectory assumes during the test.

The following picture describes the maneuver.





The maneuver consists of the following steps:

- initial steering input: the peak value is the steady-state steering angle which gives a lateral acceleration of 0.3g multiplied by a factor of 8;
- maintain peak value of steering angle for 0.25 seconds or maintain peak value of steering angle until the roll rate falls below 1.5 deg/sec depending on whether Fixed Time or Roll Rate Feedback field is selected in Steering Actuation Method text box;
- steering in the opposite direction until the opposite value of peak steering angle is reached;
- maintain the latter peak value of steering angle for 3 seconds;
- steering back until the steering angle brings back to 0 degrees.

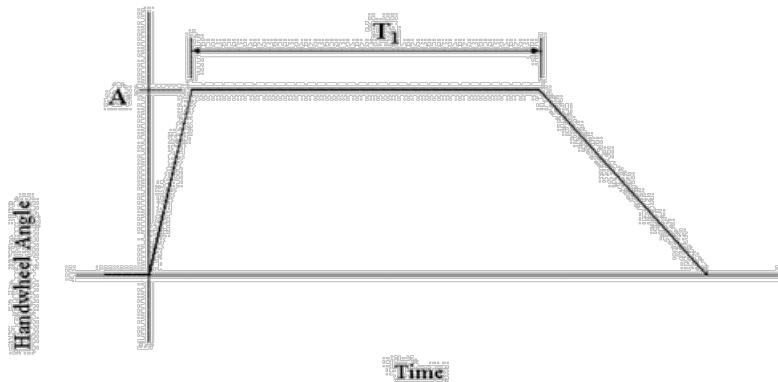
The user can set the following parameters:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Steering Actuation Method**
The user can select Fixed Time or Roll Rate Feedback.

NHTSA Fishhook	
End Time	<input type="text" value="30.0"/>
Initial Velocity	<input type="text" value="22500.0"/>
Initial Gear	<input type="text" value="1"/>
Steering Actuation Method	<input type="text" value="Roll Rate Feedback"/>

The JTurn event is performed to test the lateral transient response of the vehicle.
The name of this maneuver derives from the shape assumed by the trajectory of the vehicle.

The following pictures reports the steering angle vs. time:



The steering rate in the increasing part is equal to 17.45 rad/sec and the peak value A is defined as the product between a coefficient (8.0) and the steering wheel angle corresponding to a steady-state lateral acceleration of 0.3g.

When this peak is reached, the steering angle is maintained for a time $T_1 = 4$ seconds and during this time, a full brake for 0.5 seconds is performed.

Finally a counter steering with a rate of 8.25 rad/sec is done until the steering wheel angle reaches 0 degrees.

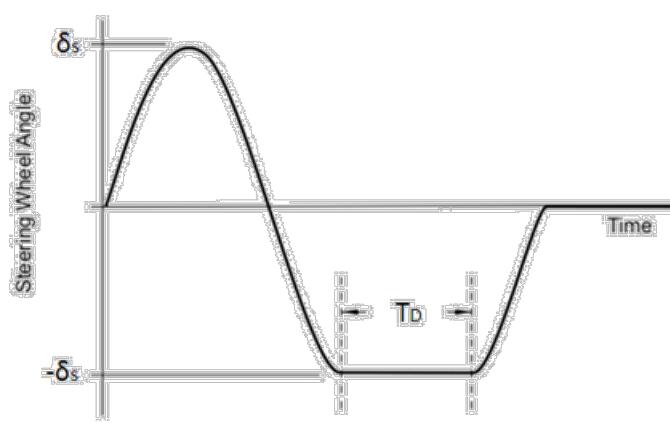
The user can set the following parameters:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.

NHTSA J-Turn	
End Time	<input type="text" value="30.0"/>
Initial Velocity	<input type="text" value="22500.0"/>
Initial Gear	<input type="text" value="1"/>

The Sine With Dwell maneuver is a stability test which stresses the lateral dynamic of the vehicle.

The following pictures reports the steering angle vs. time:



As first an increasing steer test intended to determine the steering wheel angle associated with a lateral acceleration of 0.3g for a speed of 80 km/h is submitted.. Then the sine with dwell maneuver is performed, consisting in a steering wheel angle sine wave at 0.7 Hz (one with initially positive steer and one with the sign reversed) with a 500 ms delay beginning at the second peak amplitude as shown in the figure above.

The user can set the following parameters:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Frequency**
It represents the frequency of the steering wheel angle sine wave.
- **Nominal Lateral Acceleration (g's)**
It's the acceleration used to detect the maximum steering angle δ_s .
- **Dwell Duration**
It is defined as the time, after the second peak amplitude in steering sine wave, in which the steering angle is maintained at the constant value $-\delta_s$.

- **Nominal Steering Wheel Angle Gain**

It's the coefficient multiplying the maximum steering wheel angle δ_s .

Sine With Dwell	
End Time	30.0
Initial Velocity	22500.0
Initial Gear	1
Frequency	0.6
Nominal Lateral Acceleration (g's)	0.3
Dwell Duration	0.5
Nominal Steering Wheel Angle Gain	1.0

Straight Line Events

This category contains predefined events to run the most common longitudinal dynamic maneuvers. User customize the maneuver specifying only a predefined set of parameter. All events in this category are performed using VI-Driver.

Available events classes are:

- [Straight Acceleration](#)
- [Straight Braking](#)

Straight Acceleration events run dynamic vehicle simulations using VI-Driver.

Straight Acceleration	
End Time	120.0
Initial Velocity	5000.0
Initial Gear	1
Open Loop Throttle	no
Target Acceleration	10000.0
Straight Line	yes
Steering Demand	0.0
<input checked="" type="checkbox"/> Shift Gears	

User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Open Loop Throttle**
if "Yes" a throttle ramp is applied, if "No" a target acceleration is specified.
- **Start Time**
time at which the acceleration begin.

- **Ramp Up Time**
ramp duration in case of Open Loop Throttle.
- **Throttle Demand**
ramp amplitude in case of Open Loop Throttle.
- **Target Acceleration**
desired acceleration, in case of Closed Loop Throttle.
- **Straight Line**
toggle between straight or curved path.
- **Steering Demand**
constant steering for non straight maneuver.
- **Shift gears**
fixed gear or machine controlled gear switch.

Straight Braking events run dynamic vehicle simulations using VI-Driver.

Straight Braking	
End Time	10.0
Initial Velocity	25000.0
Initial Gear	3
Open Loop Brake	yes
Start Time	1.0
Ramp Up Time	1.0
Brake Demand	1.0
Straight Line	yes
Steering Demand	0.0

User can modify all the most important parameter of the maneuver:

- **End Time**
duration of the event.
- **Initial Velocity**
initial velocity of the vehicle.
- **Initial Gear**
initial gear of the vehicle.
- **Open Loop Throttle**
if "Yes" a throttle ramp is applied, if "No" a target acceleration is specified.
- **Start Time**
time at which the acceleration begin.
- **Ramp Up Time**
ramp duration in case of Open Loop Brake.
- **Brake Demand**
ramp amplitude in case of Open Loop Brake.
- **Target Acceleration**
desired acceleration, in case of Closed Loop Brake.
- **Straight Line**
toggle between straight or curved path.
- **Steering Demand**
constant steering for non straight maneuver.

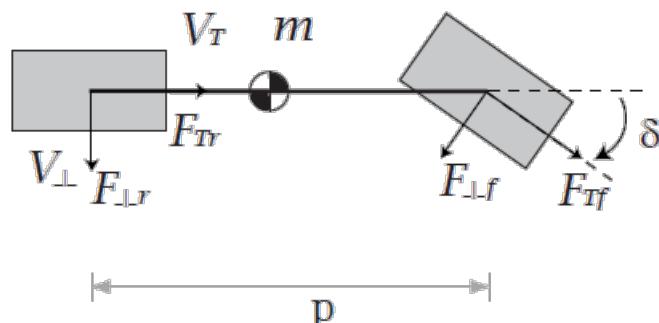
VI-Driver Theory

VI-Driver is a control system designed to drive a vehicle model in a very simple and efficient way.

The driver model has been developed with the idea that the "driver" must be:

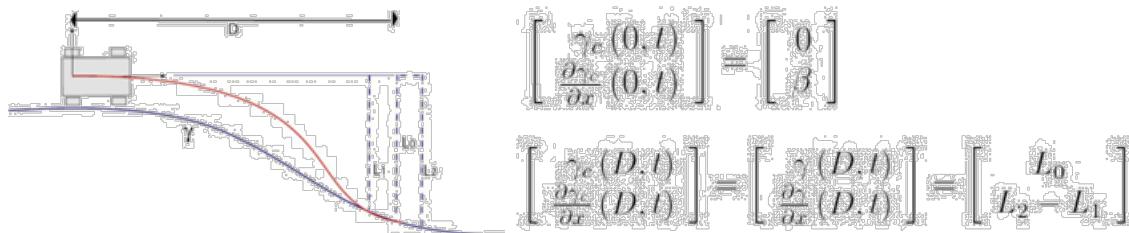
- robust enough to adapt to a wide range of vehicles characteristics
- simple to tune, and self-adapting whenever possible
- capable of driving on both limit and sub-limit maneuvers

To fulfill those requirements, the simplest vehicle model that captures the dynamical effects of interest has been chosen. The vehicle model used as the basis for a model based predictive controller is the classical bicycle model, shown in the following picture:



VI-Driver's lateral controller uses a model base predictive control technique, calculating at each instant in time the required control action.

For the vehicle model shown above, given a target curve which represents the trajectory that must be followed, it is possible to define a connecting contour (compliant to the differential flatness principle) obtained by resolving the equation below (assuming a proper order polynomial is used for interpolation):



The vehicle moving frame is used as reference (Frenet Frame), and the connecting contour constraints taken into account are:

1. contour initial position (compatible with the vehicle position)
2. contour initial orientation (compatible with the vehicle speed and heading)
3. contour final position (the reference driverline evaluated at the preview distance)
4. contour final orientation (smoothly joined with the reference driverline)

Those four constraints require at least a polynomial of order four: a cubic polynomial has been chosen, and its coefficients calculated using the relations shown in the picture above.

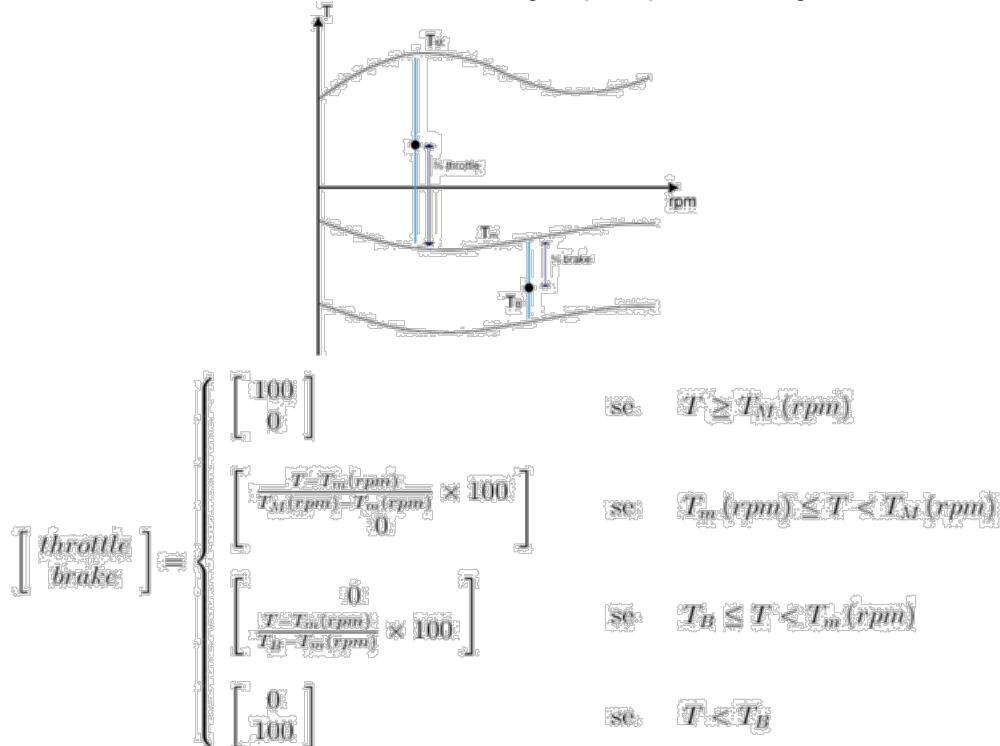
Using the differential flatness property, the connecting contour is used as dynamic vehicle trajectory to be applied as control action. The same property allows inverting that trajectory and calculating the appropriate steering control action.

This procedure is repeated at each instant in time, following the Model Predictive Control Paradigm.

Given the vehicle speed v , the side slip angle, the preview time pt and the preview distance D (computed as $v \cdot pt$), the principle used is able to offer a very good approximation of the required steering angle, necessary to bring back the vehicle on the target path, when a tracking error occurs.

A final stage has been implemented, which compensates for the unmodeled lateral dynamics. The compensation, called "yaw rate controller", uses the reference path curvature and the actual vehicle yaw rate, and corrects the steering action with the yaw rate error to bring the vehicle instantaneous curvature as close as possible to the reference path curvature.

As for longitudinal dynamics, a model based feedforward/feedback scheme is used to compute the needed torques to accelerate or brake the vehicle, based on the target speed profile coming from the stationary prediction:



the following scheme is used to compute the necessary final feed-forward torque:

$$T_{FF} = T_{engine} + T_m + (D_A + \Delta_{DRAG}) R_{eq} = T_{engine} + T_m + D_R R_{eq}$$

Because of unmodeled dynamics, disturbances and numerical integration noise, the simulated vehicle almost always diverges from the global optimal trajectory target. The combination of the control actions takes the vehicle back on it in some sub-optimal fashion: the motion control acts on the steering, the throttle and/or the brake in order to reduce the longitudinal speed tracking error to acceptable limits. The controller is not acting in a combined way though, the longitudinal and the lateral controller algorithms are separated: any longitudinal controller action would certainly have influence on the lateral dynamics of the car, especially at the limit. Nevertheless the simplicity and the natural robustness of the lateral controller invokes correct actions, and it has been observed to behave very realistically in all the cases where the target trajectory is not properly followed, due to excessive side slip. Those cases include limit over and understeer situations, which typically occur to a real driver when trying to reach the limit vehicle performance.

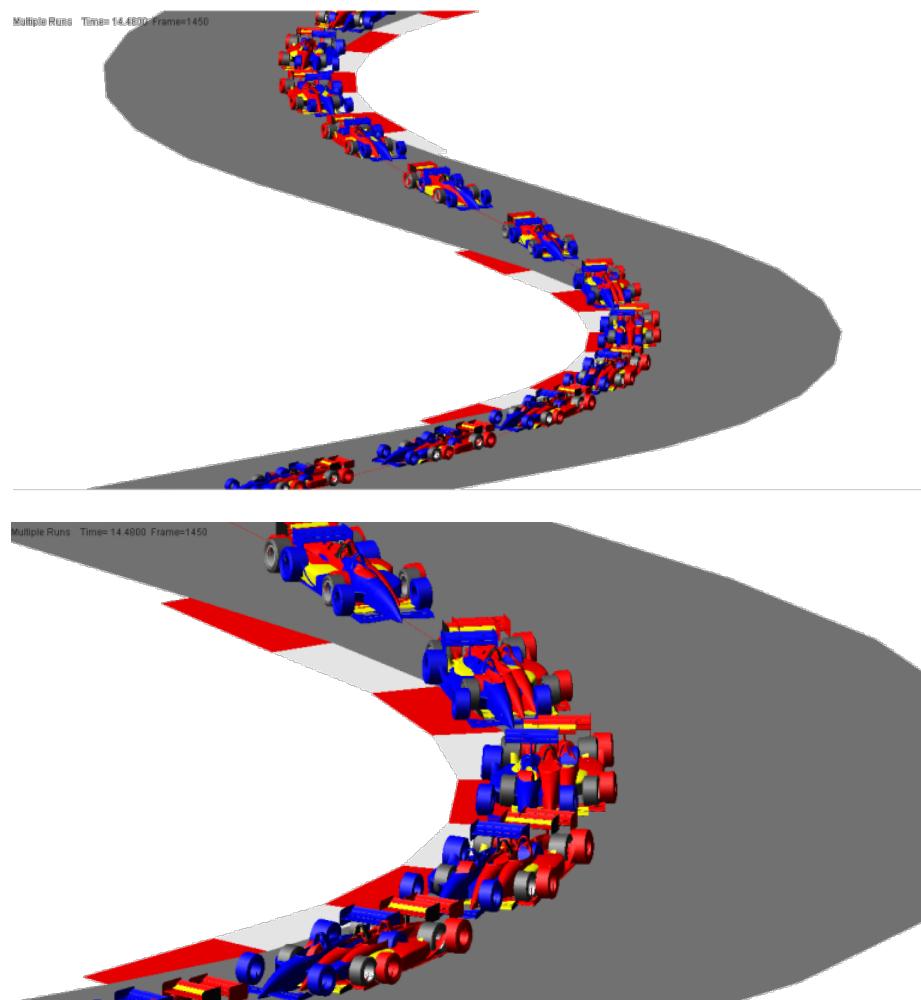
The preview time parameter (pls check at the [VDF File Format](#) documentation for a detailed description) is used to calculate the preview distance and then the connecting contour.

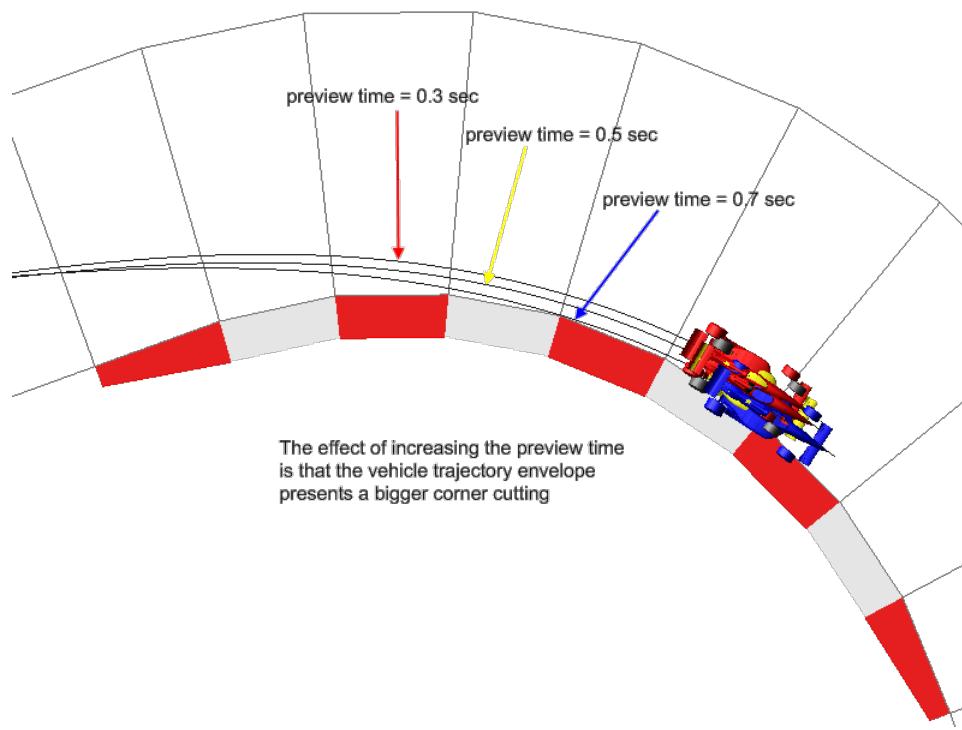
The key concept behind the connecting contour technique, is that *the vehicle is moving on the envelope of the connecting contours* initial point.

The following pictures shows the same maneuver simulated with 3 values of preview time:

- preview time = 0.3 sec for the red vehicle
- preview time = 0.5 sec for the yellow vehicle
- preview time = 0.7 sec for the blue vehicle

Because of the model predictive algorithm used to calculate the steering actions, the preview distance D (calculated as $PT \cdot vel$) is the distance between the vehicle reference system and the look-ahead point projected onto the driverline, where the vehicle is planned to join the reference trajectory. The main consequence of this approach is that increasing the preview time a more stable control action is generated, with the disadvantage of a bigger "corner cutting" effect, as shown in the following picture:





Knowing this, the user should choose the preview time value taking into account the following hints:

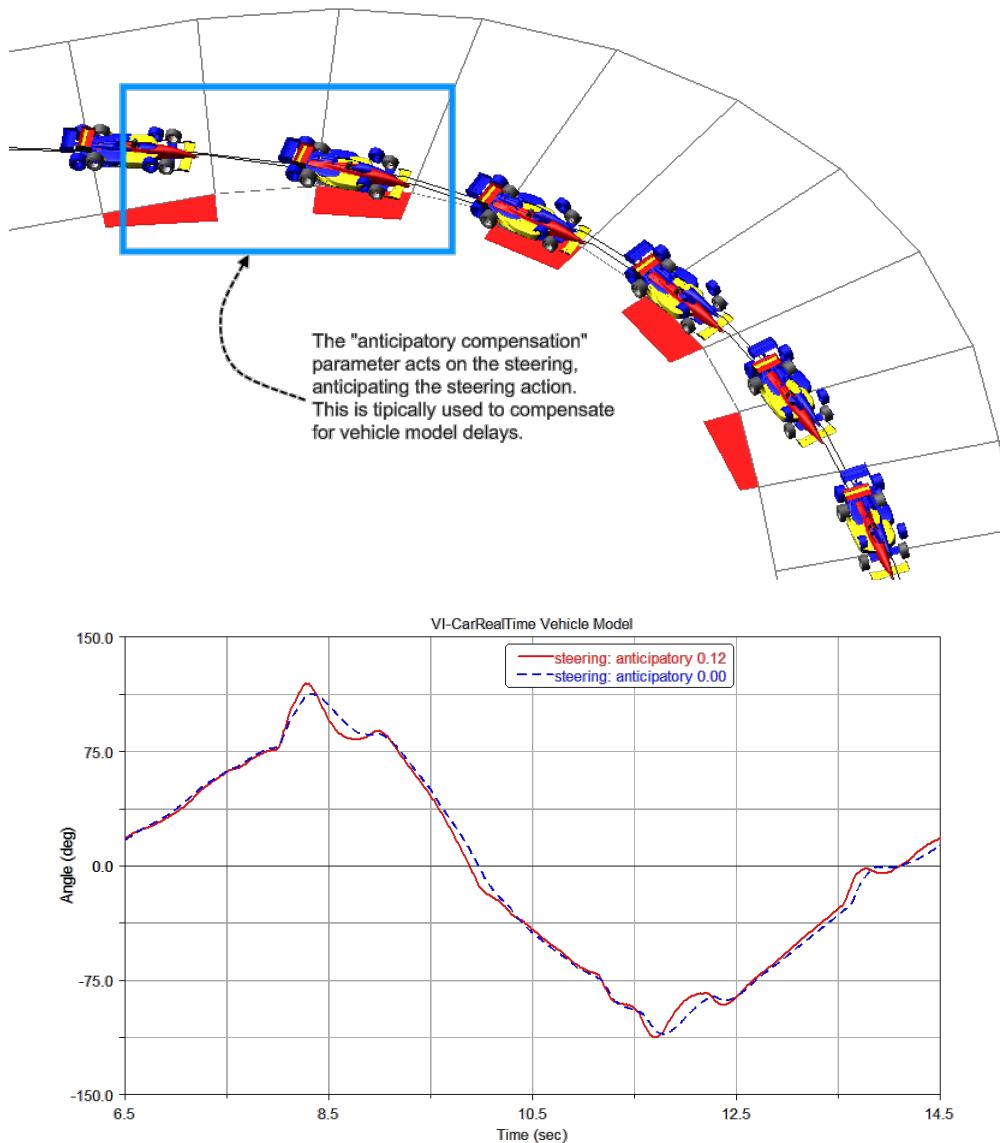
- if the steering action in the maneuver is too nervous, or the vehicle unstable, the preview time should be increased, to generate a smoother control action (but, on the opposite, relaxing a little the tracking error performance)
- if the vehicle is too far from the reference trajectory, the preview time should be decreased, to generate a stronger control action (this could lead to a potentially more nervous steering signal)
- if there road unevenesses or any other external "noise" source is present, and the steering signal is reflecting such noise, then the preview time should be increased, to reduce the steering sensitivity to the noise

In summary, a good compromise should always be reached between tracking accuracy and stability of the lateral control action.

The anticipatory compensation (pls check the [VDF File Format](#) documentation for a detailed description) parameter is used to compensate for vehicle delays due to its inertia and compliance, tire time lag and/or suspension compliance.

Its action is to move ahead the reference point used for the calculation of the connecting contour, by an amount $A=AC*vel$.

The effect of the anticipatory compensation parameter is to *anticipate* the steering action, and thus the vehicle insertion into the turn, as it appears evident by the following two pictures.



The value of anticipatory compensation can also be negative, to compensate for vehicle lateral dynamics overreactions.

Knowing this, the user should choose the anticipatory compensation value taking into account the following hints:

- if there are important chassis, tires, suspension and/or steering line compliance present, the AC should be set approximately to a value that can be measured in an open loop steering manouevre (at the typical target speed) between the two highest response peaks (for example, obtained from the lateral acceleration signal) of an equivalent kinematic vehicle (obtained stiffening the compliances) and the target vehicle
- if the vehicle is unstable because it shows delays while negotiating a turn, then the anticipatory compensation should be increased.
- if the steering wheel angle is operated too much in advance with respect to the turn apex, then the anticipatory compensation should be reduced

In general, it is recommended to change the default AC value (=0) to a different from zero one, depending on compliance effects.

VI-CarRealTime

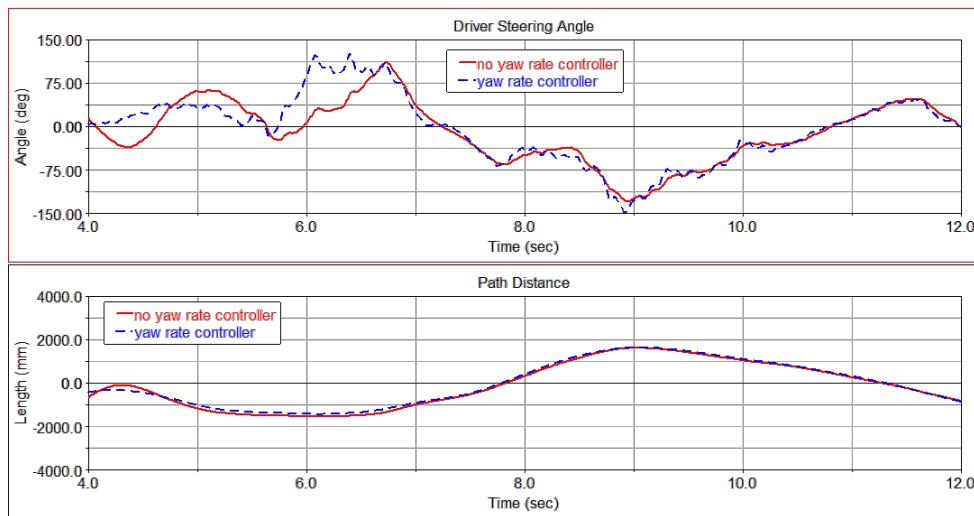
The yaw rate controller is meant to compensate for the unmodeled vehicle lateral dynamics.

Its input signals are the reference path curvature and the actual vehicle yaw rate, and the output is a steering compensation value meant to bring the vehicle instantaneous curvature to match the reference.

Nevertheless, if the maneuver is simulated with a big amount of road disturbances, or there are any other external noise sources, this controller will try to compensate the disturbances if their frequency is similar to the driver calculation frequency.

If this is the case, deactivating the yaw rate controller can lead to a more realistic control action (while, of course, getting the tracking a bit worse).

The following picture shows the driver steering signal and path distance on a maneuver performed with road disturbances. With the yaw rate controller disabled, the steering action is smoother, while the tracking performances are slightly worse.

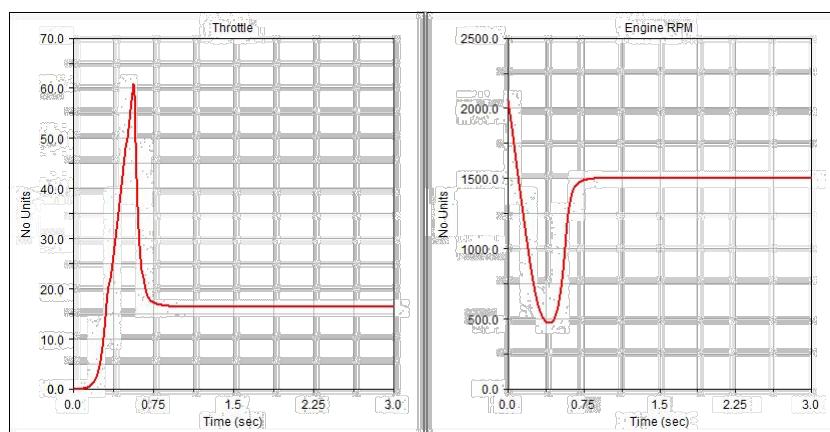


Please refer to the [VDF File Format](#) documentation for detailed information on how to enable or disable the yaw rate controller.

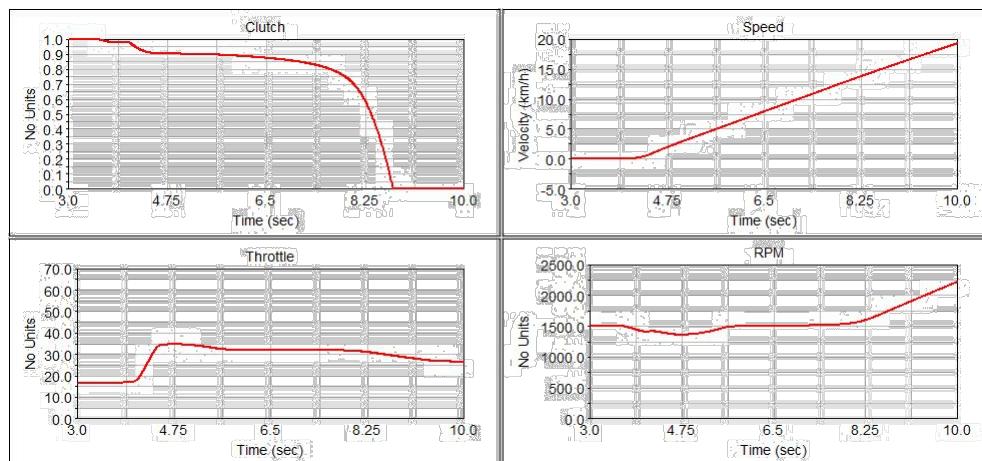
This chapter describes some of the additional functionalities featured by VI-Driver. In almost all cases an extended functionality is implemented as an additional control module that can be enabled using the proper keys in the event file. For a complete review of the supported keys please refer to the [VDF File Format](#) chapter.

In order to simulate urban driving condition or simply for running detailed transmission studies, it is required to slow down the vehicle to zero speed and then being able to restart it. Different restart strategies are needed to properly reproduce a mild driving style or a very aggressing racing like startup. VI-Driver features a specific startup module able to properly actuate throttle and clutch of both manual and automatic transmission in order to accelerate the vehicle from standstill condition with different startup strategies.

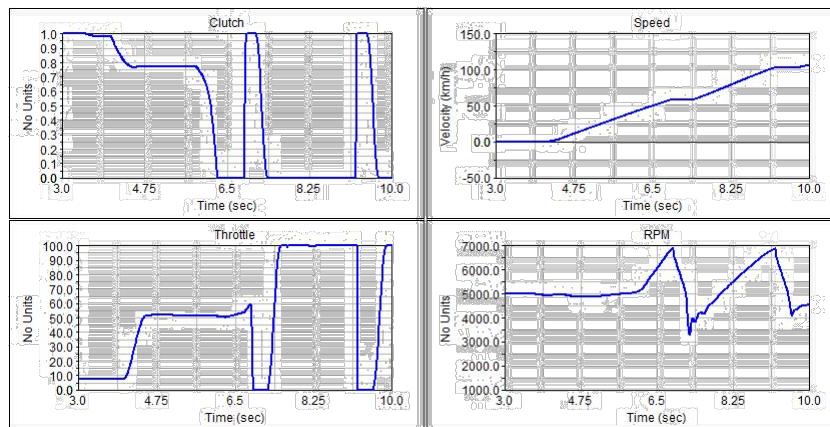
Once the vehicle is in idle conditions with clutch open, the startup process can begin increasing the throttle demand in order to accelerate the engine to the desired speed. The target of this preliminary step is to get enough engine torque for accelerating the vehicle:



Once the engine speed is stable, VI-Driver moves to the next step in which it controls the clutch channel in order to startup the vehicle maintaining the engine speed as constant as possible:



When finally the clutch is closed, the startup process is completed and the standard VI-Driver longitudinal controller is used to continue the maneuver. Different startup strategies are defined adjusting the desired initial engine speed and the target acceleration to track during the startup process. The plots above are typical for a mild restart with a very soft clutch actuation and a very low engine speed. The following plots shows the case of a quicker startup with the same vehicle:



The relevant block of the VDF file used for the startup maneuver are the following:

```
[CLUTCH_STANDARD]
STARTUP_START_RPM = 1500
STARTUP_START_TIME = 3.5
```

VI-CarRealTime

```
STARTUP_TARGET_AX = 1
```

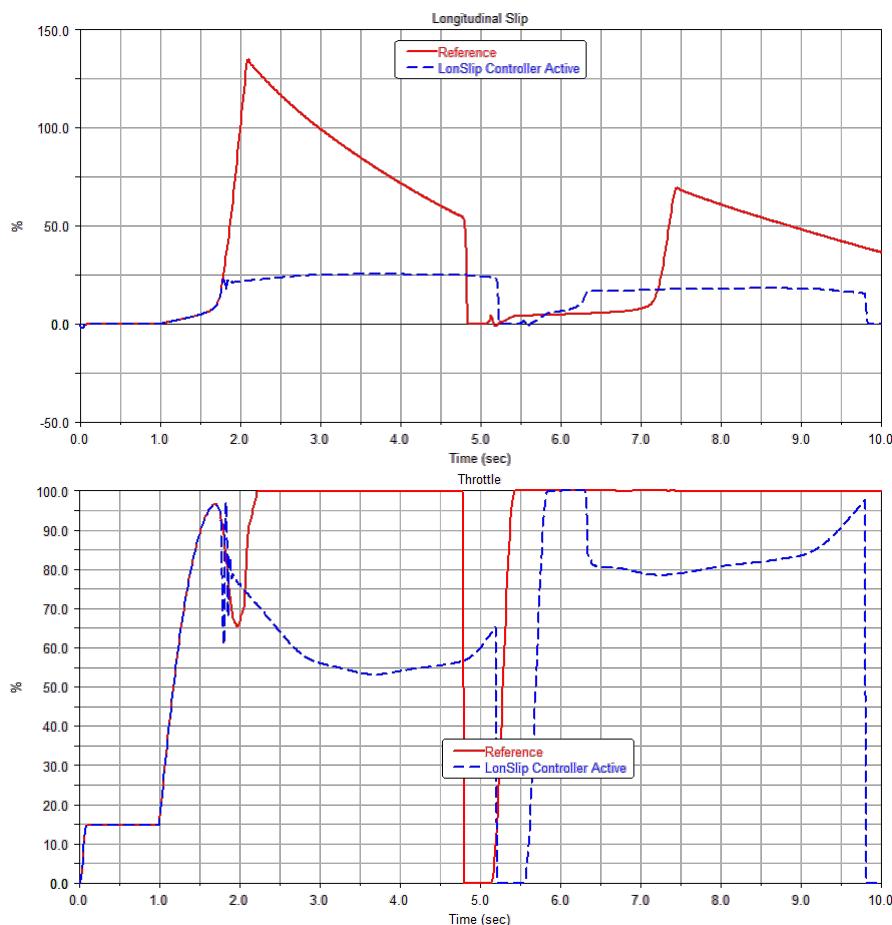
In addition the maneuver block should declare the `TASK = 'STARTUP'` instead of the usual `TASK='NORMAL'`. Note that the maneuver should have throttle and clutch set to machine. Please refer to the [VDF File Format](#) chapter for all information related to the event file syntax

Please note that this control works properly only when the vehicle model has an independent state for the engine speed.

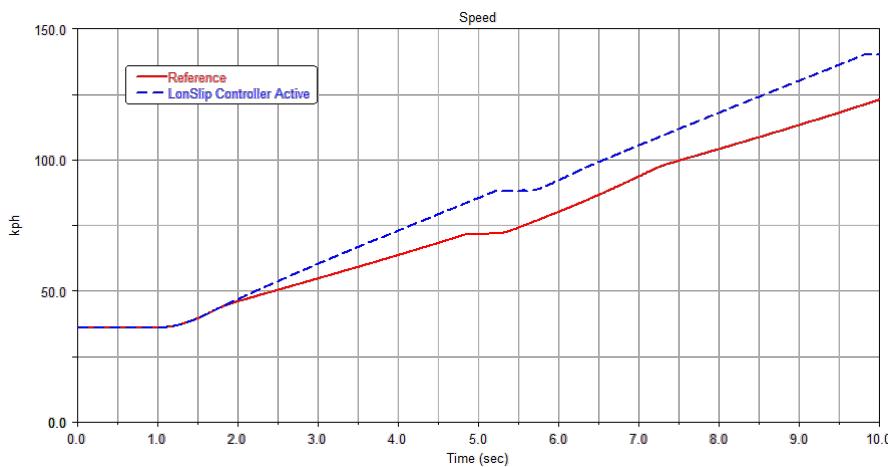
VI-Driver can be instructed to modulate throttle demand for preventing wheel spinning condition. This functionality is designed to handle cases in which a traction control module is not implemented in the vehicle plant.

Please refer to the [TIRE_LONSLIP_STANDARD](#) section of the VDF file format chapter for detailed description of the parameters affecting the longitudinal slip controller action and the data required to activate it.

The following example shows the action of the controller during an acceleration maneuver on a low friction surface:



Thanks to the throttle reduction, the slip ratio of the driving tires is reduced and the vehicle speed increases faster:



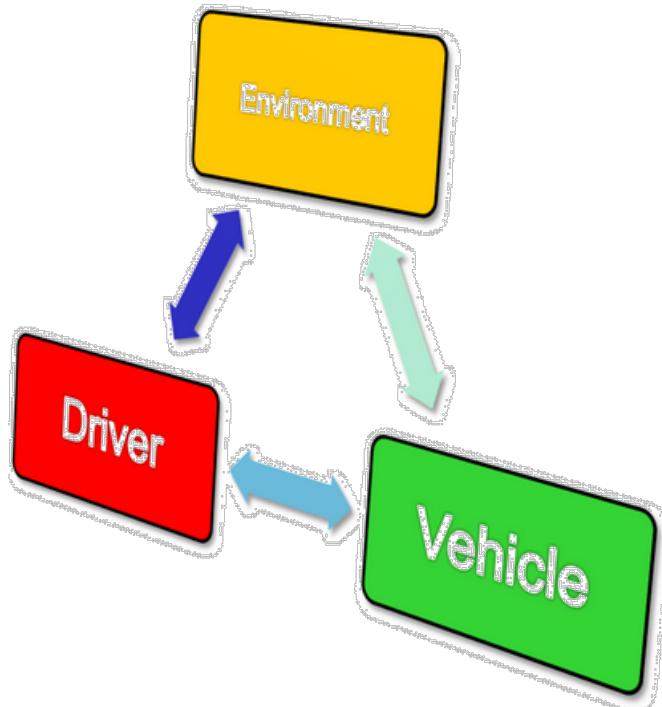
Please note that this control module requires tire states to operate properly. These information may not be available in all implementation of VI-Driver.

During the development activity, it is often needed to drive the vehicle in a way that is comparable to a real driver. For example in the development of driving aid, security and stability systems the vehicle controller must behave like a human being, with the same skills and limitations.

The "Human Driver" model developed by VI-grade it's based on the idea that the simulating the reality means modeling the continuous exchange of information among three main actors:

- the [Environment](#)
- the [Vehicle](#)
- the [Driver](#)

These entities are connected with each other, providing different kind of information, stimuli, forces.



Environment

The environment interacts with the vehicle providing reaction forces and friction through the road surface, air resistance to the chassis, external forces such as gravity etc.

It also provides information about the position and actual velocity to the driver (through the optical flow perceived by the driver eyes), and collateral information needed to interact with external entities, like other vehicles, obstacles, etc.

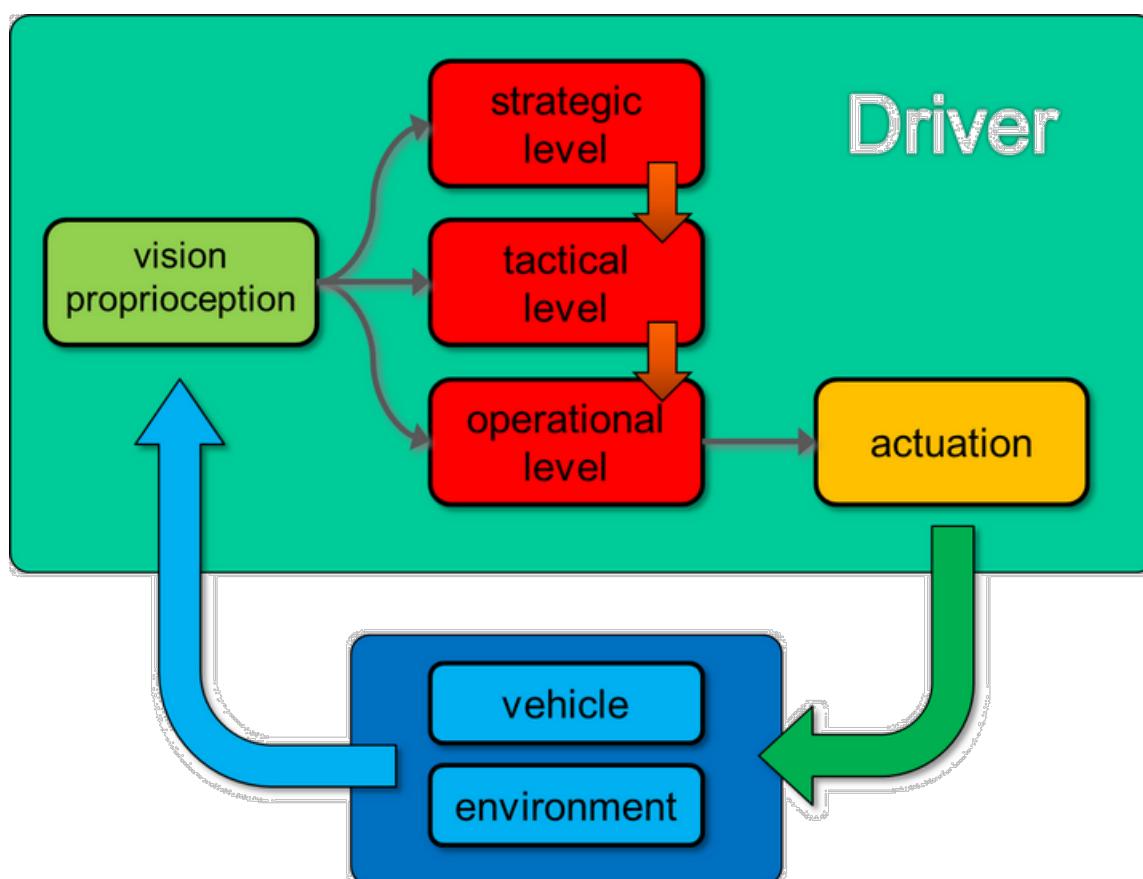
Vehicle

The vehicle is the 'interface' the driver uses to interact with the environment. It has a set of well-defined controls that can be used (steering wheel, throttle pedal, etc.) to change its state, in response of some driving strategy. The vehicle movement provides to the driver all the information about the perceived accelerations, the steering feedback, and optionally additional data coming from on-board systems.

Driver

The driver is the main entity responsible for collecting all the inputs coming from the environment and from the vehicle, defining a strategy and a set of actions for fulfill the given task. The cognitive model will be explained in the next section.

The following picture show the internal structure of the Human Driver cognitive model:



The driver collects information from the environment and from the vehicle through the *perception layer* (the green block), defines the control action in the *logical layer* (the red blocks) and operate the vehicle through the *actuation layer* (the orange block).

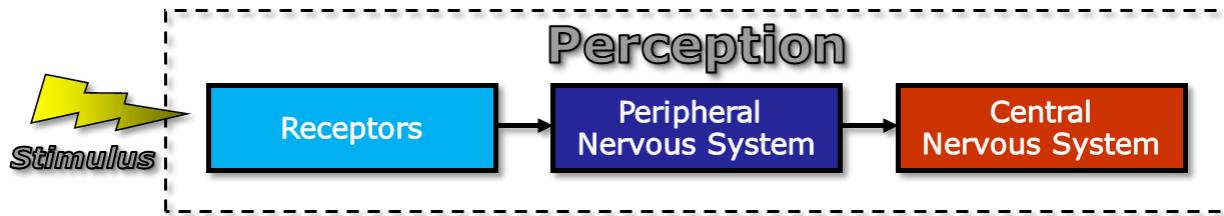
Perception Layer

This layer is responsible for the information retrieval from the environment, the vehicle and any additional information source available. The various subsystem have different sensitivities, signal to noise ratios, bandwidth, delays, etc.

The human perception works as follows:

1. a stimulus is retrieved by a specific structure called 'receptor'. There are many kind of receptors, spread all around the body; for example within the inner ear there are housed the semicircular canals, that control the sense of steadiness, balance, and also provide information on the gravity orientation;

2. the receptors convert the stimulus in electric signals, and send those signals to the Peripheral Nervous System;
3. the peripheral nervous system conveys the signals to the Central Nervous System (the Brain);
4. the brain receive all the signals, and produces a reduced set of signals using the redundant channels to compensate for lack of precision and sensing errors.



The two main subsystems are:

- the Vision subsystem: it collects spatial information from the environment, data about tracking, obstacles and in general external events. Using the optical flow it's also responsible for estimating the vehicle velocity, the side slip angle. The typical precision is in the range of 0.1 circular degrees, spanning approximately 150 degrees; the frequency band is vastly subjective, but usually in the range 10 - 20 Hz.
- the Proprioception subsystem: it is composed by a broad range of different receptors, distributed all around the body. Some examples are the semicircular channels in the inner ear, the pressure and temperature receptors in the skin, etc.

Logical Layer

The logical layer is a simplification of the elaborations made in the brain. Its main purpose is to start from a task definition and an history of sense data and calculate the actions to take. It can be subdivided in three levels:

- *Strategic Level* : high level decisions
Example: pass through the cones with no hits in a double lane change maneuver.
- *Tactical Level* : maneuver definition
Example: define a trajectory and a speed profile for the lane change maneuver.
- *Operational Level* : vehicle control
Example: Drive the vehicle to follow the given trajectory.

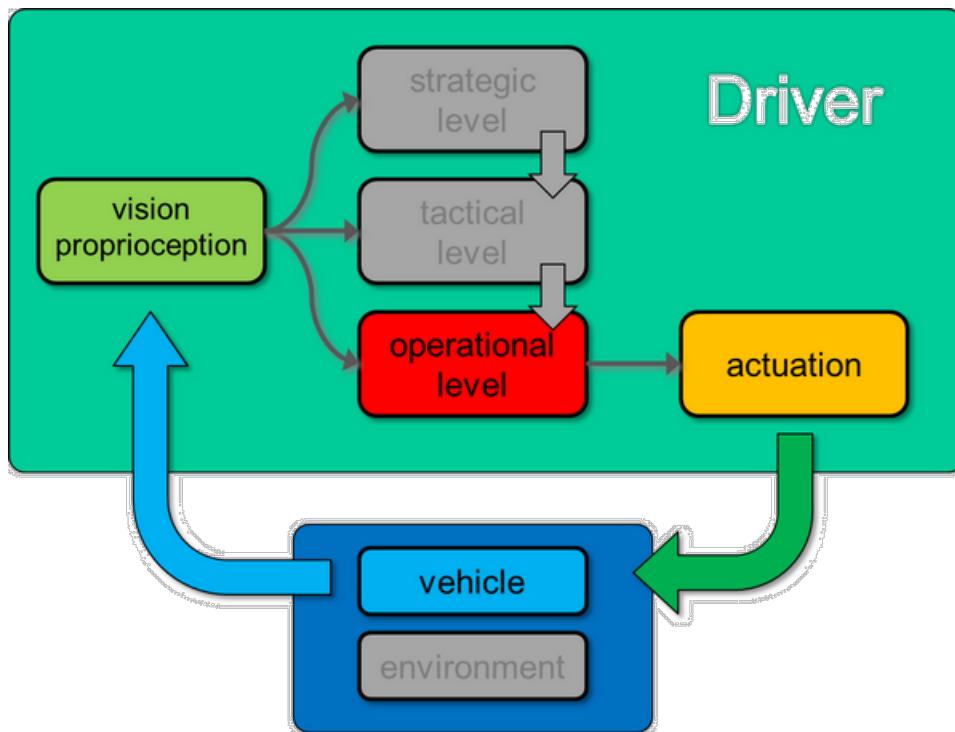
This layer is also responsible for compensating the delays and inaccuracies introduced by the actuation layer.

The strategic, tactic and operational levels can be associated to the driver skill, because they use an internal representation of the vehicle, the interaction with the environment and the precision of the actuation system.

Actuation Layer

The actuation layer converts the definition of the action in the action itself. Muscles, articulations and the feedback sensors needed to measure the actual movement are part of this subsystem. Maximum allowed power, cut frequency, delay, etc. are specific of the particular organ or sensor they refer to.

In VI-Driver release 16, a subset of the previous human model has been implemented. In the following picture, the functionalities not yet implemented are represented by grey blocks.



The actual model has the following limitations:

- No strategic and tactical levels have been implemented
- The information come entirely from the vehicle model

Despite the missing blocks, the actual implementation already allow to model different driver skills, and to provide more consistent results especially with respect to the rejection to high frequency content coming from the vehicle, the road surface etc.

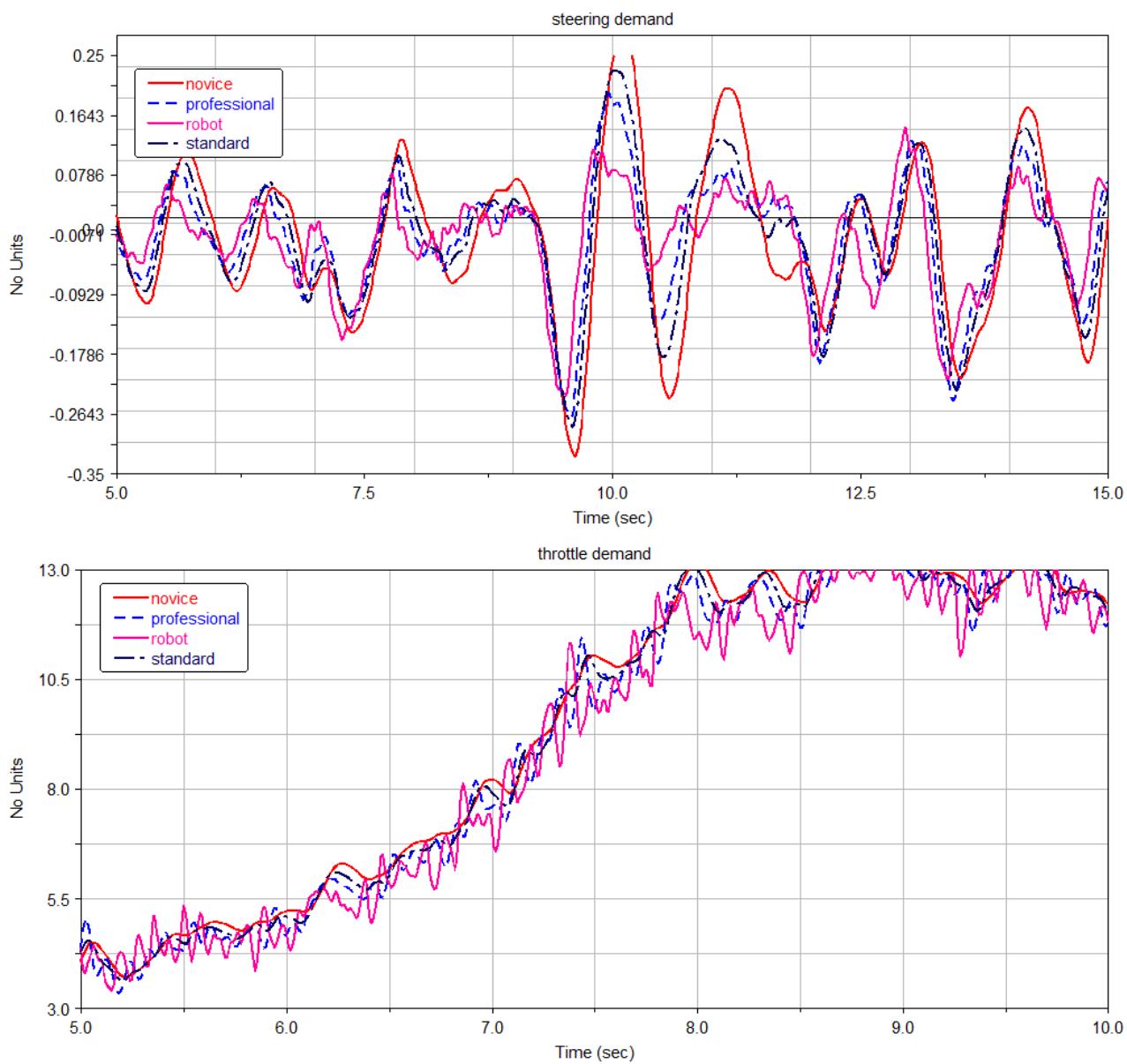
Four different skill levels have been exposed:

- ROBOT: it is equivalent to a standard VI-Driver event, with the Human Driver disabled
- PROFESSIONAL: it's designed to behave as a fast reacting and precise professional driver
- STANDARD: this is comparable to a normal driver
- NOVICE: unexperienced and slow driver

Please refer to the [VDF File Format - Human Driver](#) documentation for detailed information on how to enable the Human Driver in a driver event.

When the Human Driver controller is activated, the anticipatory compensation value is ignored, and the preview time is clamped to a minimum value of 1.0 sec.

In the following picture an example of the steering and throttle signals of the four skill levels. The simulation represent a durability event on a very detailed belgian-block road surface; it is possible to notice how the frequency in both the steering and throttle signals are different, and also the different behavior in terms of oscillation and responsiveness.



VISafety

The VI-Safety toolkit for VI-CarRealTime has been designed in order to enable users to setup and perform the following types of simulations:

- [Misuse](#) (straight line)
- [Curb trip rollover](#) (on sled)

In order to exploit the full set of functionalities, VI-CarRealTime models can be instrumented using a specific set of sensors, generating the relevant outputs that can be used to evaluate event results.

Misuse

Misuse event is featuring several possible testcases.

Each test-case is characterized by a specific driver schedule and associated to a road model.

Suitable road model can be selected from a drop-down menu among all the road data file stored into registered databases; road data files are associated to the specific test-case by file name prefix.

Testcases

Implemented testcases are:

- **Screwed Rollover**

No driving machine is active, steering fixed, clutch disengaged, engine idle, brake disabled. Vehicle is 'shot' with initial velocity.

Road prefix is 'screw_ '.

- **Crossing Plank**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Road prefix is 'plank_ '.

- **Drive-up 90deg Curb**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Optionally braking maneuver starting at a user specified distance in front of the obstacle can be overlayed.

Road prefix is 'curb90_ '.

- **Drive-up 35deg Curb**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Road prefix is 'curb35_ '.

- **Passing both-sided Drain**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Optionally braking maneuver starting at a user specified distance in front of the obstacle can be overlayed.

Road prefix is 'drain_bs_ '.

- **Passing one-sided Drain**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Optionally braking maneuver starting at a user specified distance in front of the obstacle can be overlayed.

Road prefix is 'drain_os_ '.

- **Run over two side-shifted Ramps**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Road prefix is 'ramp_bs_ '.

- **Run over a single Ramp**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Road prefix is 'ramp_os_ '.

- **Jump down from platform**

Driving machine is active, keeps vehicle on straight line with constant velocity.

Road prefix is 'jump_ '.

Road Data Files

New road data files can be added by placing them into the directory '*/roads.tpl*' of registered databases. Their names must start with the testcase-prefix described above.

The screenshot shows the 'Misuse event' configuration window in the VI-CarRealTime software. It includes the following sections:

- Testcase:**
 - Testcase Type: Screwed Rollover
 - End Time: 10.0
- Vehicle Conditions:**
 - Initial Velocity: 20000.0
 - Initial Gear: 3
- Road Setup:**
 - Road Data File: mdids://visafety_shared/roads.tbl/screw_ramp_left.rdf
- Brakes Setup:**
 - Active:
 - Target Deceleration: 6000.0
 - Obstacle Braking Distance: 2000.0
- End Conditions:**
 - Maximum Roll Angle: 75.0
 - Travelled Distance: 100000.0
 - Minimum Long. Velocity: 3000.0

Testcase

Parameter	Description
Testcase Type	Select type of simulation to be performed.
End Time	Specify simulation end time

Vehicle Conditions

Parameter	Description
Initial Velocity	Enter the vehicle forward velocity at begin of simulation.
Initial Gear	Select gear position used at begin of simulation. Will be kept fix during the simulation. Needs to be entered even for the 'screwed rollover' simulation, although the clutch will be disengaged here (engine off is not possible).

Road Setup

Parameter	Description
Road Data File	Select a road data file used for simulation. Dependent on the ' Testcase ' selected above, all road_data_files valid for this testcase are listed here.

Brake Setup (this block is not available for all testcases)

Parameter	Description
Braking active	Select if a braking maneuver should be performed.
Target Deceleration	Enter value of maximum target deceleration used for braking maneuver
Obstacle Braking Distance	Enter distance in front of obstacle to start the braking maneuver.

Additional End Conditions (not all options are available for every testcase)

Parameter	Description
<i>maximum Rollangle</i>	Select to activate the roll-angle end sensor. Then enter the maximum roll-angle at which the simulation will be stopped.
<i>traveled Distance</i>	Select to activate the maximum distance traveled end sensor. Then enter the maximum traveled distance at which the simulation will be stopped.
<i>minimum Vehicle Velocity</i>	Select to activate the minimum vehicle velocity end sensor. Then enter the minimum vehicle forward velocity at which the simulation will be stopped.

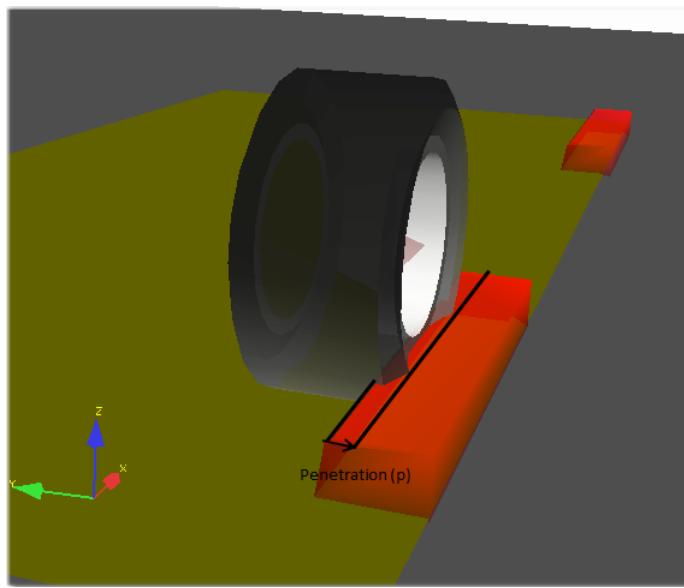
Curb Trip Rollover

Event description:

- The vehicle is placed on a sled which is accelerated from zero to a constant velocity. After a certain distance (which is automatically calculated by the input made into the 'Crash Setup' block of the Dialog_Box) the sled undergoes two curbs fixed on ground. The vehicle hits the curbs with both its left wheels at the same distance and TIME.
- The distance for each wheel outer side to its curb is the same for front and rear, so both front and rear track and the width of front and rear wheel are taken into account.
- Solid cylinders, representing wheel and rim, are automatically generated starting from tire property files and they have the following geometries:
`tire_radius` : tire unloaded radius parameter;
`tire_width` : tire width parameter;
`rim_radius` : rim radius parameter;
`rim_width` : rim width parameter.
If tire property file does not collect information about rim, rim solid cylinder will be created starting from tire geometry using `tire_aspect_ratio` parameter:
`rim_radius = tire_radius - tire_width*tire_aspect_ratio`
`rim_width = tire_width - 0.02 (m)`
- The shape of both curbs is:
tilting angle = 8.5 deg;
upper radius = 15 mm;
height = 120 mm.

Impact model description

Interactions between impact geometries (wheel+rim and curb) are modeled using impact forces. Such impact force becomes active when the distance (i.e. penetration) between impact geometries rises above 0, that is, when the two parts collide.



[Bullet 2.81 Physics Library](#) is the routine used to compute the intersection volumes between impact solids. Once the contact is established, bullet library finds the centroid of the intersection volume, the center of mass of the intersection volume, and then it finds the closest point on each solid to the centroid. The distance between these two points is the **penetration depth**.

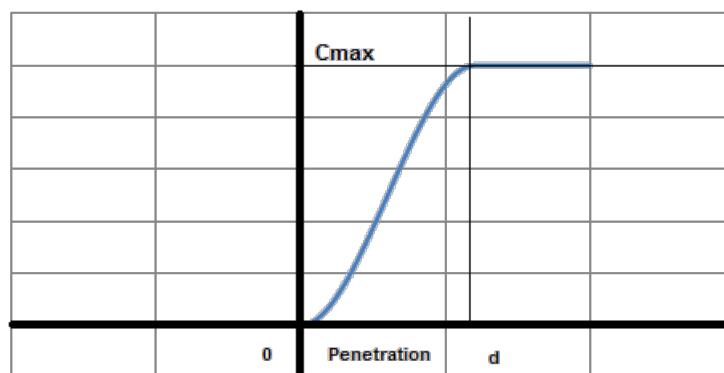
The impact force is zero as long as the penetration between the wheel and rim cylinders and curb is negative. The force has two components, a *spring* or *stiffness* component, and a *damping* or *viscous* component. Stiffness component is proportional to contact stiffness coefficient (k), and it is a function of the solids penetration. The stiffness component opposes the penetration. The damping component of the force is a function of the speed of penetration. The damping opposes the direction of relative motion. In order to prevent a discontinuity in the damping force at contact, the damping coefficient is, by definition, a cubic step function of the penetration. According to this model, at zero penetration, the damping coefficient is always zero and it achieves its maximum (c_{\max}) at a defined penetration, $d = 5e-4$ meter.

$$\text{Force} = \begin{cases} 0, & p < 0 \\ \text{Max}(0, k * p^e - c * \frac{dp}{dt}), & p \geq 0 \end{cases}$$

where:

$e = 1.5$;
 c = step function of penetration.

Damping Coefficient vs Penetration Depth



Moreover a friction force is applied in the plane normal to impact force (static/dynamic friction coefficient = 0.3/0.1).

As contact is a discontinuous event, when two geometries come into contact it could happen that a large normal force or an impulse is generated. As consequence, this numerical spike could generate solver instability if integrator time step is not small enough. In order to avoid this when simulation is close to impact event, solver resample automatically this time step (new time step becomes 1/5 of the standard one).

Sled Curb Trip Rollover

Crash Setup

Crash Velocity	3000.0
Pre Crash Simulation Time (Constant Velocity)	0.5
Post Crash Simulation Time	2.0
Sled Initial Acceleration	750.0

Steering and Wheels Contact

Stiffness		Damping
<input type="checkbox"/> Steering Wheel Lock	5000.0	
Rim To Curb Contact	1000.0	100.0
Tire To Curb Contact	150.0	1.5

End Conditions

<input checked="" type="checkbox"/> Maximum Roll Angle	75.0
--	------

Crash Setup

Parameter	Description
Crash Velocity	Enter the sled velocity at begin of impact (crash) and select the appropriate units.
Pre Crash Simulation Time (Constant Velocity)	Enter simulation TIME the sled should drive with constant velocity before impact occurs.
Simulation Duration after Impact	Enter duration of simulation TIME after impact occurs.
Sled Initial Acceleration	Enter linear sled acceleration to rise sled velocity from zero to full.

Steering and Wheels Contact

Parameter	Description
Steering Wheel Lock	Check to activate steering wheel lock.
Torque on Steering Wheel	Enter linear stiffness and damping coefficients for torque spring on steering wheel (steering lock inactive)
Contact Rim to Curb	Enter parameters for rim to curb contact: stiffness coefficient and damping coefficient
Contact Tire to Curb	Enter parameters for tire to curb contact: stiffness coefficient and damping coefficient

Additional End Conditions

Parameter	Description
maximum Rollangle	Select to activate the roll-angle end sensor. Then enter the maximum roll-angle at which the simulation will be stopped.

VI-SpeedGen

The event embeds in this unit permits the calculation of the lap time on a given track through a simplified model.

[StaticLapTime](#)

Note: refer to VI-SpeedGen Theory chapter to understand how VI-SpeedGen algorithm works.

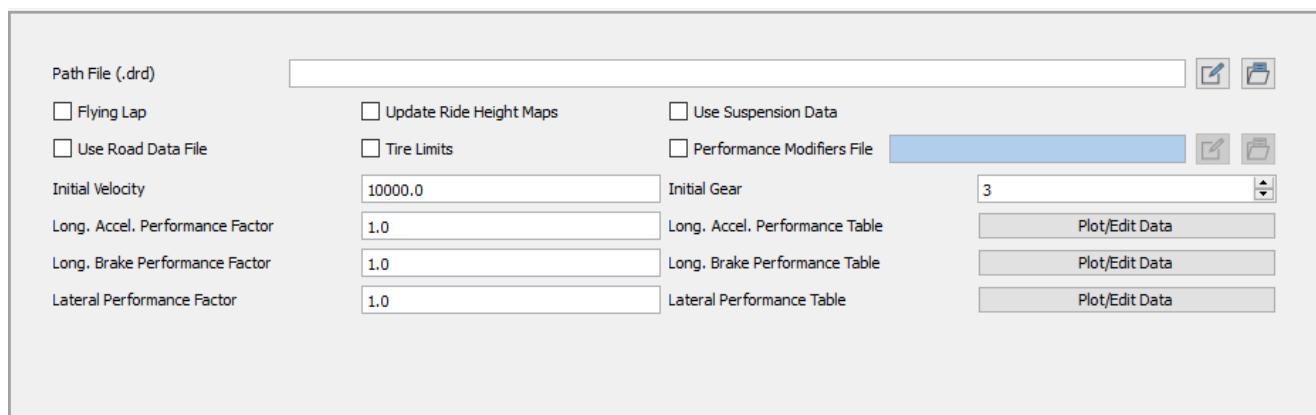
StaticLapTime

StaticLapTime event in VI-CarRealTime is used to define static limit velocity profile on a given track. The event uses a specific static solver (SpeedGen) and a simplified model w.r.t. the one used in other events. Basically, the vehicle has no suspensions and inherits all the properties from the full VI-CarRealTime model.

The effect of aero forces is considered and the effect of suspension jounce is taken into account by the presence of ride height maps.

The solver, based on vehicle model and specified lateral and longitudinal grip, computes the limit velocity profile and produces it as output tabular data for simplified vehicle state and a Driving Machine event file which can be used to verify if the vehicle can reach the limit set by VI-SpeedGen static solver in a dynamic simulation.

The figure below shows the property editor for this event class.



The available parameters for the Quasi Static event are:

- **Path File**

specifies the path that the vehicle has to follow. It can be visualized and edited in VI-Road using the button beside the fields.

- **Initial Velocity**

initial velocity for the simulation (disabled when activating flying lap option).

- **Initial Gear**

initial gear for the simulation (disabled when activating flying lap option).

- **Flying Lap**

computes velocity profile imposing that final velocity is the same as initial velocity.

- **Update Ride Height Maps**

[ride height maps](#) are data that can be optionally stored in the vehicle model (body subsystem); they are used by the static solver to compute the dependency of Ride Height (and Aero Forces) on vehicle velocity, since the vehicle model in the static solver has no suspensions. These maps can be generated or updated before running the Max Performance event. To create them a coast down dynamic simulation is run starting from a high velocity with open clutch on a straight track. The simulation continues until the vehicle velocity, due to aero drag, reaches a given value.

Note: the parameters to control the dynamic ride height map "pre-phase" are shown in vehicle properties [System Parameters-Update ride height maps](#).

Once the pre-phase simulation is run, the maps are present in memory, but not stored into [body subsystem](#) file.

To visualize (if not present) and eventually save them, the user has to follow these steps:

1. go in vehicle system parameters.
2. modify a parameter.
3. click apply.
4. eventually reset the parameter to original value.

Ride height map instance should now be visible in the body subsystem property editor under Auxiliary Spline Data Tab.

- **Long. Accel. Performance Factor**

Vehicle longitudinal performance while accelerating.

- **Long. Brake Performance Factor**

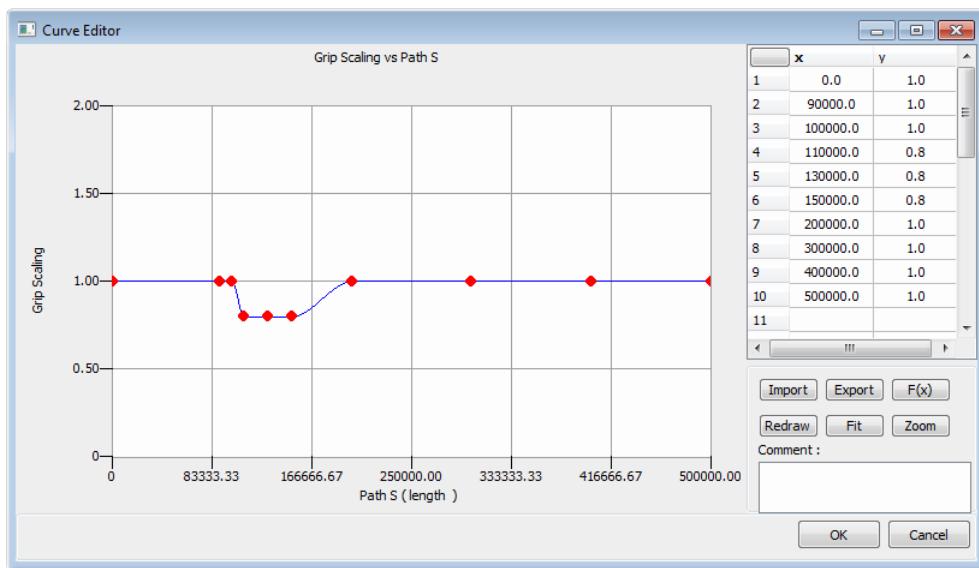
Vehicle longitudinal performance while braking.

- **Lateral Performance Factor**

Vehicle lateral performance.

- **Lateral Performance Table**

The user has the option to scale the grip in some regions of the track when verifying that the performance computed by VI-SpeedGen cannot be reached in dynamics. A 2D spline function of path S is used to scale the grip.



- **Long. Accel. Performance Table**

The user has the option to scale the grip in some regions of the track when verifying that the performance computed by VI-SpeedGen cannot be reached in dynamics. A 2D spline function of path S is used to scale the grip.

- **Long. Brake Performance Table**

The user has the option to scale the grip in some regions of the track when verifying that the performance computed by VI-SpeedGen cannot be reached in dynamics. A 2D spline function of path S is used to scale the grip.

- **Use Road Data File**

when the toggle button is checked, the user is allowed to enter a Road Data File ([rdf](#)) which will be used by VI-SpeedGen to compute the static speed profile. If the toggle button is unchecked, a flat road with constant unitary friction coefficient will be used by the VI-SpeedGen solver.

- **Tire Limits**

when the flag is checked, the [Tire Limits](#) tool is used to evaluate the force envelope produced by each tire. Such envelopes are used by VI-SpeedGen solver to compute the maximum lateral and longitudinal acceleration of the simplified vehicle; otherwise an approximated force ellipsoid is used to evaluate the maximum pure longitudinal and lateral forces. In general the Tire Limits option allows VI-SpeedGen solver to estimate speed profiles closer to the dynamic simulation ones (results produced by VI-Driver using the vdf generated by the StaticLapTime event), thus reducing the overestimation of the speed profile.

- **Performance Modifiers File**

when the flag is checked, it is possible to specify a [PMF](#) file to alter aerodynamic and electric motor performance as a function of path_s.

- **Use Suspension Data**

when the flag is checked, the simulation relies on VI-SpeedGenEvo internal model to compute the static speed profile. In such case [Update Ride Height Maps](#) toggle button is disabled since the evo model doesn't use it. In such condition, before VI-SpeedGen solver is called, a suspension testrig analysis (tr suffix) is performed in order to extract the following suspension curves for front and rear suspension:

1. Base vs. Jounce
2. Camber vs. Jounce
3. Track vs. Jounce
4. Side View Angle vs. Jounce

5. Vertical Force vs. Jounce
6. AntiRollForce vs. Delta Jounce (left - right)

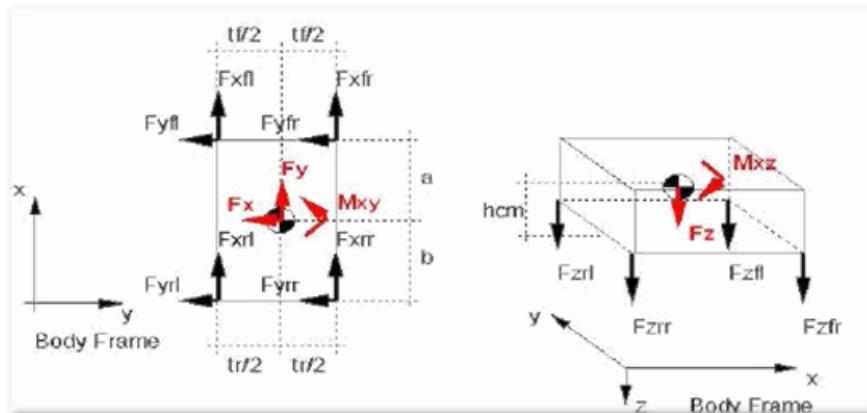
The curves are then passed to VI-SpeedGenEvo to compute the vehicle ground stiffness.

The events created in the working directory (event name + _vdd_event.vdf) can be used to analyze the computed velocity profile and to test it in a dynamic simulation.

VI-SpeedGen Theory

VI-SpeedGen algorithm is used to compute a static speed profile on a given trajectory.

VI-SpeedGen uses a simplified model with respect to the one used in standard dynamic events. The simplified vehicle has no suspensions and inherits all the properties from the full VI-CarRealTime model.



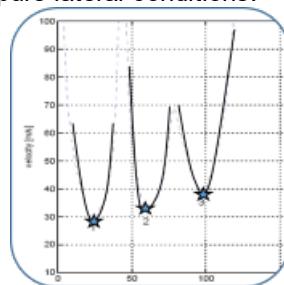
The effect of aero forces is considered and the effect of suspension jounce is taken into account by the presence of ride height maps.

The solver, based on vehicle model and specified lateral and longitudinal grip, computes the limit velocity profile and produces it as output tabular data for simplified vehicle state and a VDF file which can be used to verify if the vehicle can reach the limit set by VI-SpeedGen static solver in a dynamic simulation.

The procedure consists of the following steps:

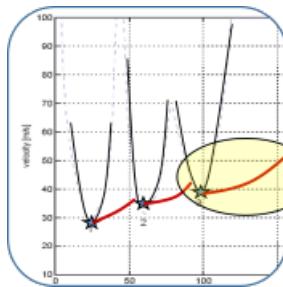
- **Step 1: Speed profile limit on Apex**

Starting from the input trajectory, usually provided with a DRD file, the algorithm recognizes the apex of all the curves and computes the static limit speed for each apex assuming that the vehicle is in pure lateral (the vehicle is assumed to have zero longitudinal acceleration). From the local curvature and after computing the maximum lateral force (acceleration) the vehicle can sustain, it's possible to compute the maximum speed in pure lateral conditions.



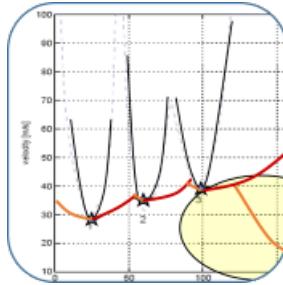
- **Step 2: Forward Integration**

Once the speed in each apex has been computed, the forward integration phase computes the maximum acceleration the vehicle can sustain from each apex forward. Since the apex is the point with minimum speed, after the apex the vehicle must accelerate and the acceleration performance of the vehicle are retrieved from tire and powertrain of the main vehicle. With such assumption, it's possible to draw the speed curve (red curve in the picture below) from each apex forward.



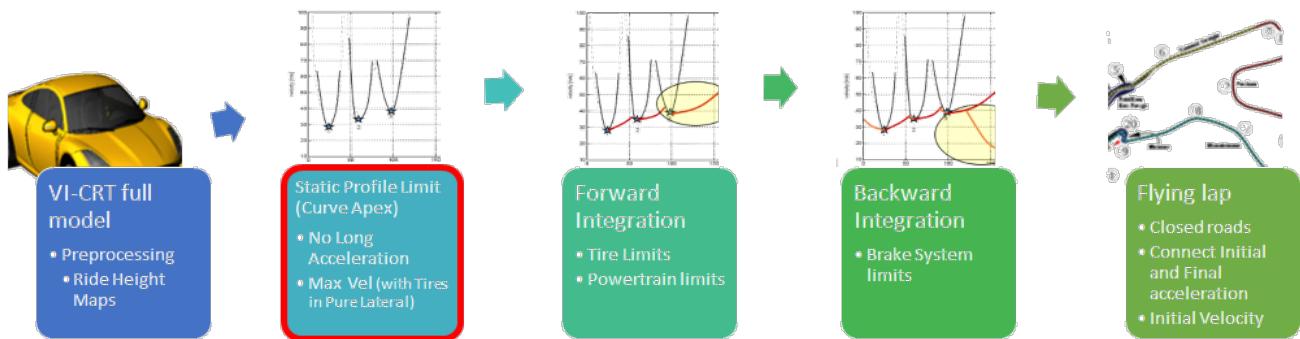
- **Step 3: Backward Integration**

The backward integration phase computes the maximum deceleration the vehicle can sustain from each apex backward. Since the apex is the point with minimum speed, before the apex the vehicle must be in deceleration and the deceleration performance of the vehicle are retrieved from tire and brake system of the main vehicle. With such assumption, it's possible to draw the speed curve (orange curve in the picture below) from each apex backward.



- **Step 4: Final Smoothing**

Red curves and orange curves are connected and smoothed in order to obtain the final static speed profile.



The procedure used to compute the static speed profile is an iterative procedure which uses the bisection method. It's composed by two successive refinements:

1. [speed profile based on pure lateral equilibrium](#)
2. [forward/backward](#) integration starting from speed profile computed at step 1

Speed profile based on pure lateral equilibrium

Starting from an initial guess speed (Vel), computed as the mean between a minimum ($VelMin$) and a maximum ($VelMax$) velocity, the speed is used to enter the *vehicle equilibrium* block: depending on the trajectory position

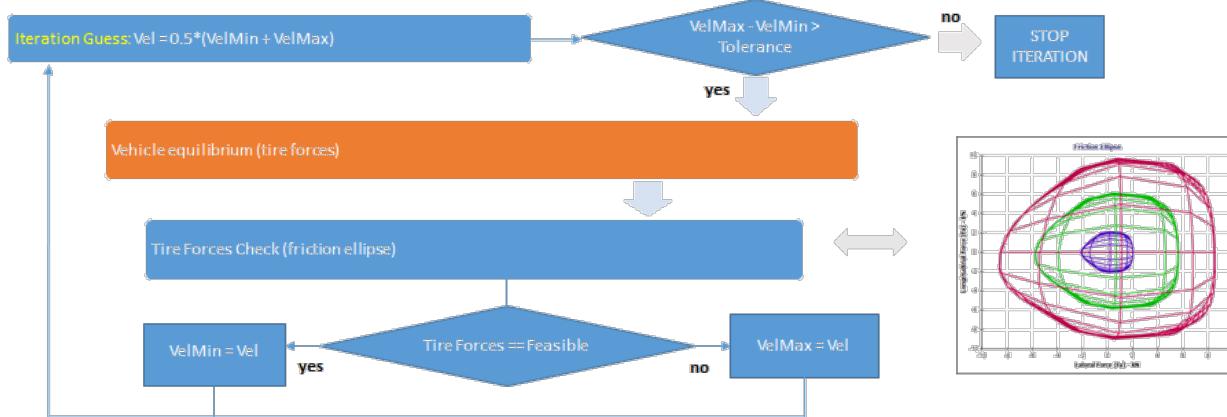
VI-CarRealTime

(apex, acceleration, deceleration), VI-SpeedGen computes the tire forces of the simplified model. Such forces are compared with the *friction ellipses* in order to check if the forces are feasible or not.

If they are feasible, it means that it's possible to go faster, so for the next iteration *VelMin* is set equal to *Vel*.

If they are not feasible, it means that the vehicle must go slower, so for the next iteration *VelMax* is set equal to *Vel*.

The iterative procedure is interrupted when *VelMin* and *VelMax* converge within the tolerance.

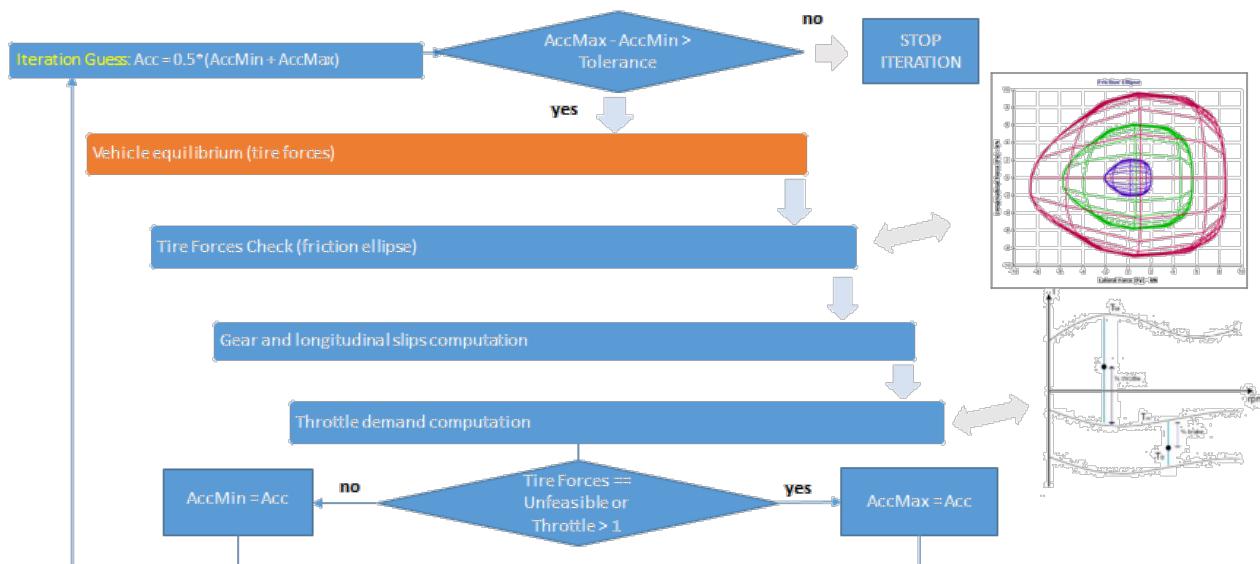


Note: the speed profile computed at this level doesn't take into account of the powertrain/brake limits of the vehicle. So in general, the speed profile computed here might not be feasible except for the apex where the vehicle is supposed to be in full lateral (no longitudinal dynamic).

The speed profile obtained is then submitted to [forward integration](#) or [backward integration](#) depending on the path location (before or after an apex).

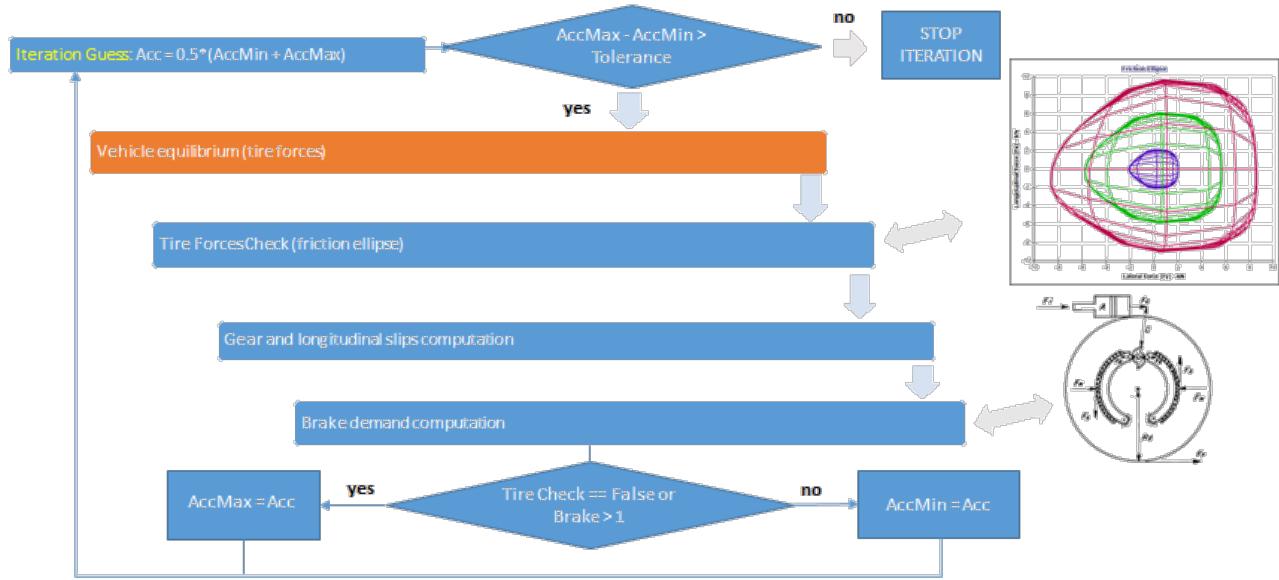
Forward integration

The forward integration uses the bisection method for the accelerations. The work-flow is the same but this time also the powertrain characteristics are taken into account: a guessed acceleration is feasible if the resulting tire forces are inside the friction ellipse and the computed throttle demand (based on powertrain characteristics) is lower than 1 (100%).



Backward integration

The backward integration uses the bisection method for the accelerations. The work-flow is the same but this time also the brake characteristics are taken into account: a guessed acceleration is feasible if the resulting tire forces are inside the friction ellipse and the computed brake demand (based on brake characteristics) is lower than 1 (100%).



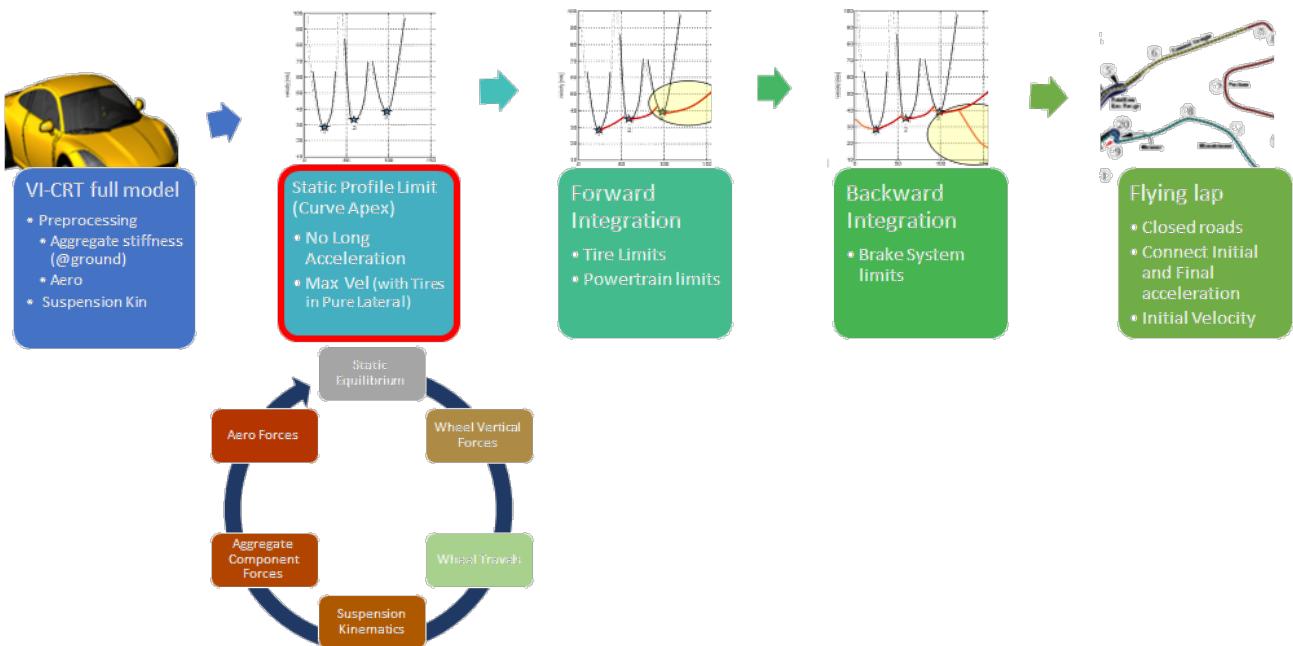
VI-SpeedGenEvo is an advanced version of VI-SpeedGen.

The working mode is exactly the same as the standard [VI-SpeedGen](#), but the **internal vehicle model** used as reference to compute the static speed profile is more detailed.

The internal vehicle model is composed by 5 rigid bodies (1 sprung mass and 4 unsprung masses).

The relative movement between sprung mass and unsprung masses is computed by using the suspension kinematic splines of the full VI-CarRealTime model.

Moreover, a pre-analysis of the full VI-CarRealTime model is performed in order to characterize the vehicle ground stiffness: parallel and opposite travel analysis using the suspension testrig.



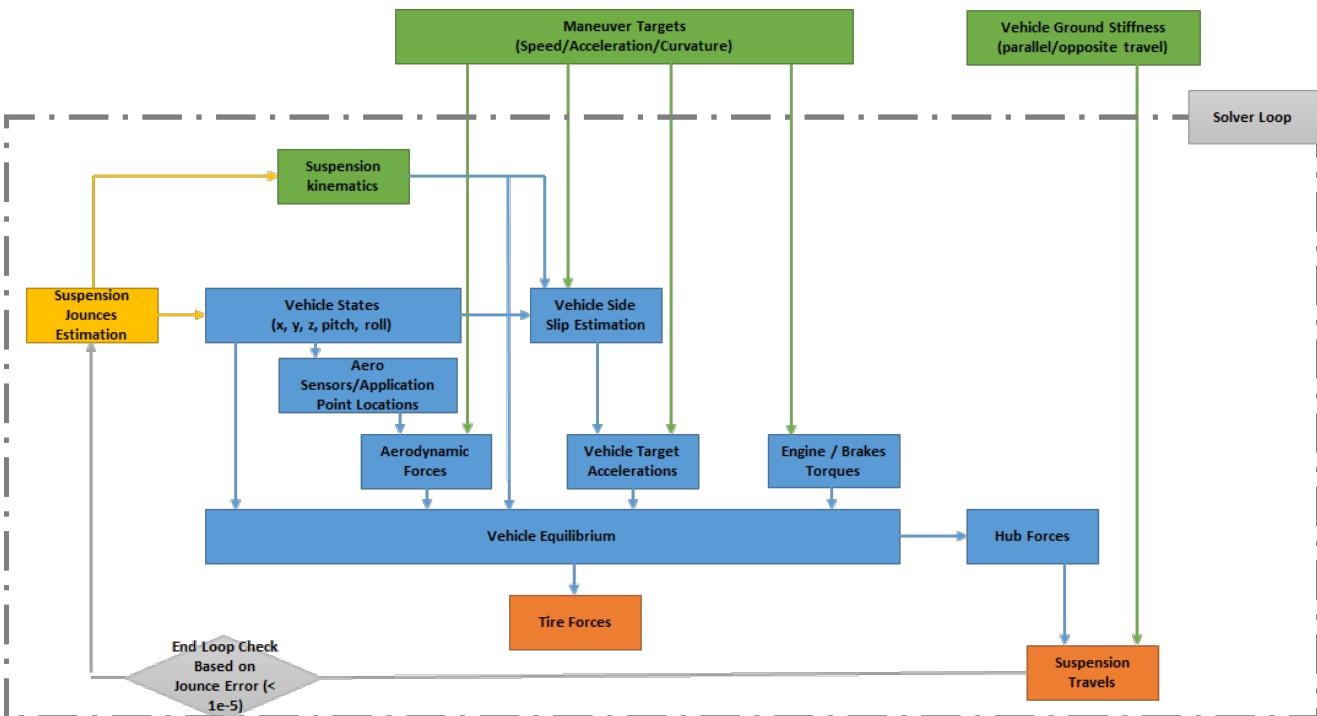
The static equilibrium of the vehicle is computed by means of [iterative statics](#).

This section describes the vehicle equilibrium work-flow of VI-SpeedGenEvo module.

Starting from the following inputs (*maneuver targets*, *vehicle ground stiffness* and *suspension kinematics characteristics*), VI-SpeedGenEvo estimates an initial suspension jounce value and uses it to compute the Vehicle Equilibrium.

The iterative procedure lasts until the suspension jounce error is lower than a threshold (default 1e-5 m).

The tire forces obtained at the end of the solver loop are compared with the friction ellipses in order to check the feasibility.



VI-TestRig

This unit embeds two common analyses for investigating the behaviour and the response of a vehicle due to its suspension system.

- [SevenPostRig Events](#)
- [Suspension TestRig](#)

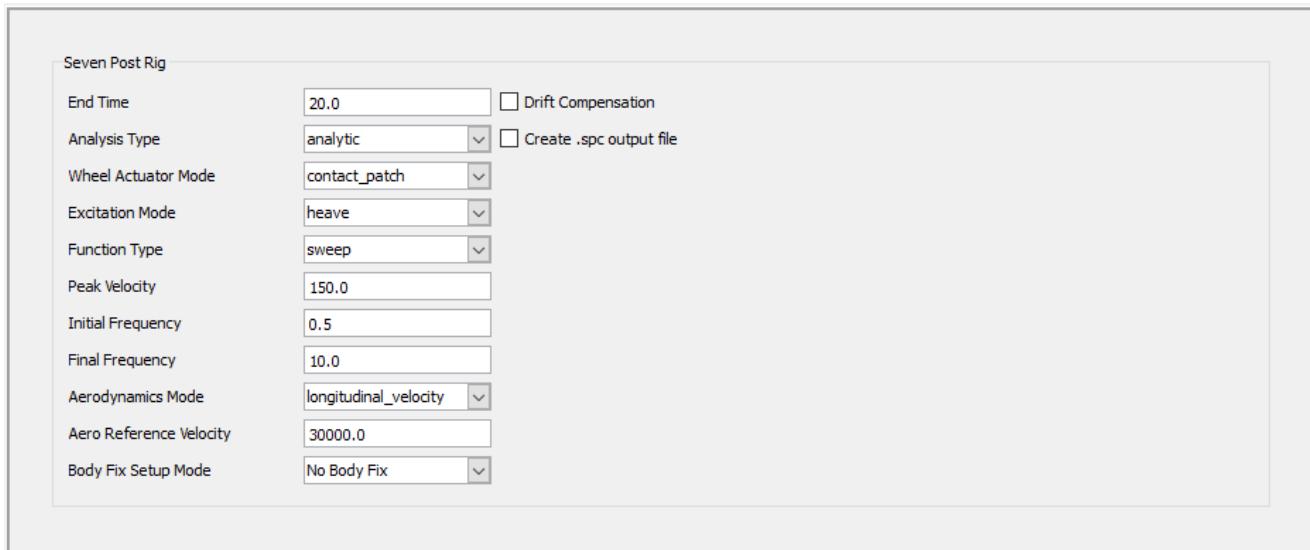
Sevenpostrig Events

The objective of this kind of analysis is to investigate the vertical dynamics of the full vehicle model by exciting its suspension subsystems. The test results can be post-processed in the frequency domain to study the damped natural frequencies of various ride "modes" and their respective damping levels.

Additional insights can also be gathered into the influences of the vehicle's vertical dynamics' effects on handling behaviour by gaining a further understanding of system dynamic responses including:

- **Front to rear modal balance**
- **Suspension to body transfer function gain and phase**
- **Suspension to tire transfer function gain and phase**
- **Tire contact patch vertical load variation**

The test is conducted by assembling a standard full vehicle model to a special 7-Post Testrig.



Two different kind of full vehicle assemblies can be used for this analysis:

- **with wheel subsystems** (Wheel Actuator Mode to **contact_patch**)
in this case the actuator input is applied to the tire contact patch. The analysis results will be affected by the tire characteristics.
- **without wheel subsystems** (Wheel Actuator Mode to **hub**)
actuators are rigidly connected to wheel hubs.

The Sevenpostrig supports two different analysis type:

- [analytic](#)
- [data file input](#)

The common parameters for the two analyses are:

- **End Time**
Simulation end time.
- **Wheel Actuator Mode**
Specify the actuators connection with the vehicle: available options are: *contact patch* and *hub*.
- **Create .spc output file**
Enable the creation of the [SPC](#) file in which all the simulation parameters and actuation data are stored.
- **Body Fix Setup Mode**
It allows the user to choose how to constraint the sprung part of the vehicle with respect to the testrig structure. Available options are:
 - **No Body Fix**: the sprung part is free (no constraints with the testrig structure);
 - **Fix Body at Design**: the sprung part is fixed to the testrig structure using the location the sprung mass has in design condition;
 - **Fix Body at Setup**: the sprung part is fixed to the testrig structure using the location the sprung mass has at the end of the [setup phase](#).

Standard Results Set

Every sevenpostrig analysis has some specific results set:

- **sevenpostrig**

These results set contains the following items:

- *FL/FR/RL/RR_target*
contain the reference signals. Depending on the tracking mode (open, hub acceleration, spring length) the signals can be an acceleration, a spring extension or any displacement, velocity or acceleration given as openloop signals.
- *F/RL/RR_aero_target*
contain the reference aerodynamic forces on the front, rear left and rear right aero actuators.
- *FL/FR/RL/RR_preprocessed*
if the tracking mode is "hub acceleration" or "spring extension", these results set contain the reference signal used internally by the tracking routines; These signals always represents hub displacements (relative to the ground in case of hub acceleration tracking and relative to the vehicle chassis in case of spring extension tracking). If the tracking mode is "open" the signals are exactly the same as the target.
- *F/RL/RR_aero*
contain the aerodynamic forces applied to the chassis. If the user activates the longitudinal and lateral acceleration compensation (see [SPC file format](#)), these requests contain those contributions.

- **global_upright_disp/vel/acc**

These results sets contain the wheel hub positions, velocities and accelerations.

- **global_pad_disp/vel/acc**

These results sets contain the pad actuators positions, velocities and accelerations.

In Analytic mode, the sevenpostrig applies the pad excitation defined by the following options (selectable from **Function Type**):

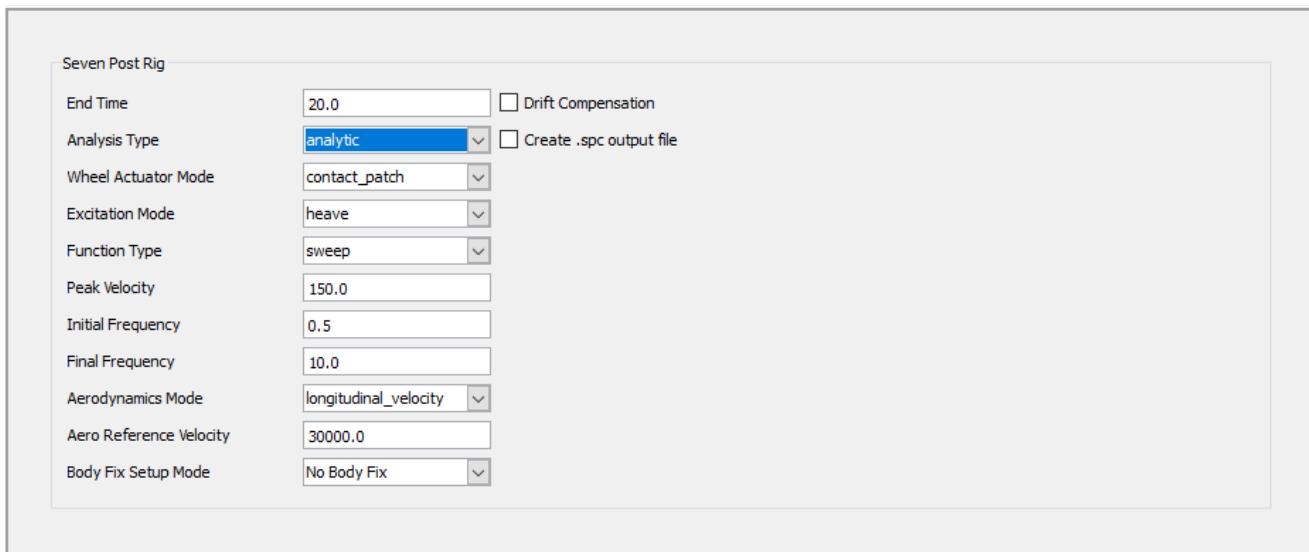
- **sine**

A velocity input is generated from the main actuators. The signal frequency is constant as the signal amplitude.

- **sweep**

The signal frequency is increased in a logarithmic way starting from the specified initial frequency. The signal amplitude is reduced in order to maintain constant the actuator maximum velocity.

Drift Compensation: enabling this flag the input signal is corrected in order to avoid a drifting of pads displacements at high frequencies.



The phase of the actuator input is defined by the **Excitation Mode** option:

- **Heave**

All tire pads move vertically in phase.

- **Pitch**

The front tire pads move 180 degrees out of phase with the rear tire pads.

- **Roll**

The left tire pads move 180 degrees out of phase with the right tire pads.

- **Warp**

The left-front and right-rear tire pads move 180 degrees out of phase with the right-front and left-rear pads.

The downforce actuators can be set to (selectable from **Aerodynamics Mode**):

- **no_aerodynamics**

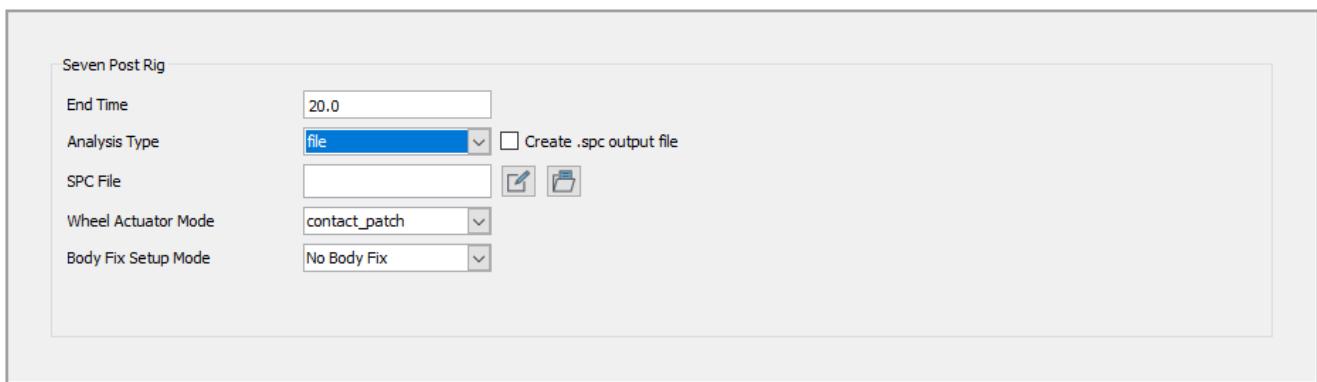
- **longitudinal_velocity**

in this case the `aero_reference_velocity` parameters is used to calculate the aerodynamic force using VI-CarRealTime standard routines.

In the data file input the actuator signals are retrieved from a configuration file, which contains the information needed to run a simulation starting from experimental input data.

In this case acceleration, velocity and displacement data are supported.

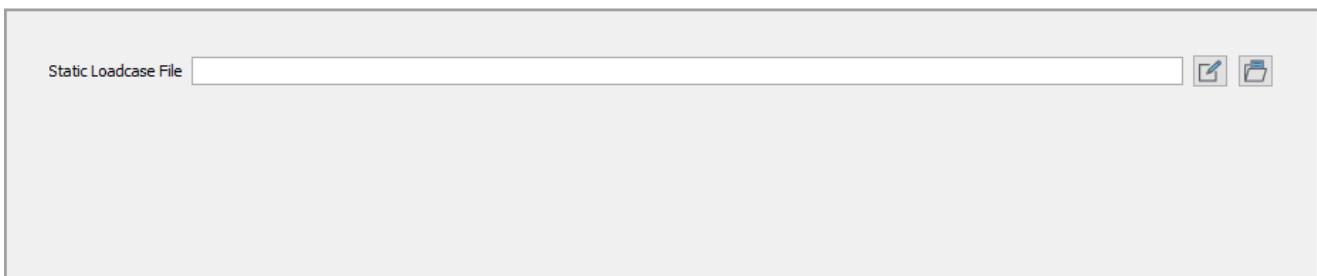
Please refer to [SPC file format](#) for all the details about how to properly format the SPC file and regarding the different options available.



Suspension Testrig

A Suspension Testrig event in VI-CarRealTime is used to test and validate suspension curves.

The Model is the full vehicle in which sprung mass is locked to the ground.
Loads are applied on wheels and eventually wheel jounce can be described.



The Simulation is driven by a loadcase file which contains the list of the inputs applied to the model which are (for each vehicle corner):

- **Jounce**

target suspension jounce. The jounce position is controlled with a PID controller, whose parameters can be set in [suspension_testrig_parameters](#) section.

The suspension jounce controller is used only when **Use Jounce Actuator** flag is set to **yes**.

- **Fz (contact patch)**

input vertical force applied at the corner contact patch. The vertical force is applied only when **Use Jounce Actuator** flag is set to **no**.

- **Fx (braking)**

input longitudinal braking force applied at the corner contact patch.

- **Fx (traction)**

input longitudinal traction force applied at the corner wheel center.

- **Fy (contact patch)**

input lateral force applied at the corner contact patch.

- **Mx**

input overturning moment.

- **My**

input rolling resistance moment.

- **Mz**

input aligning moment.

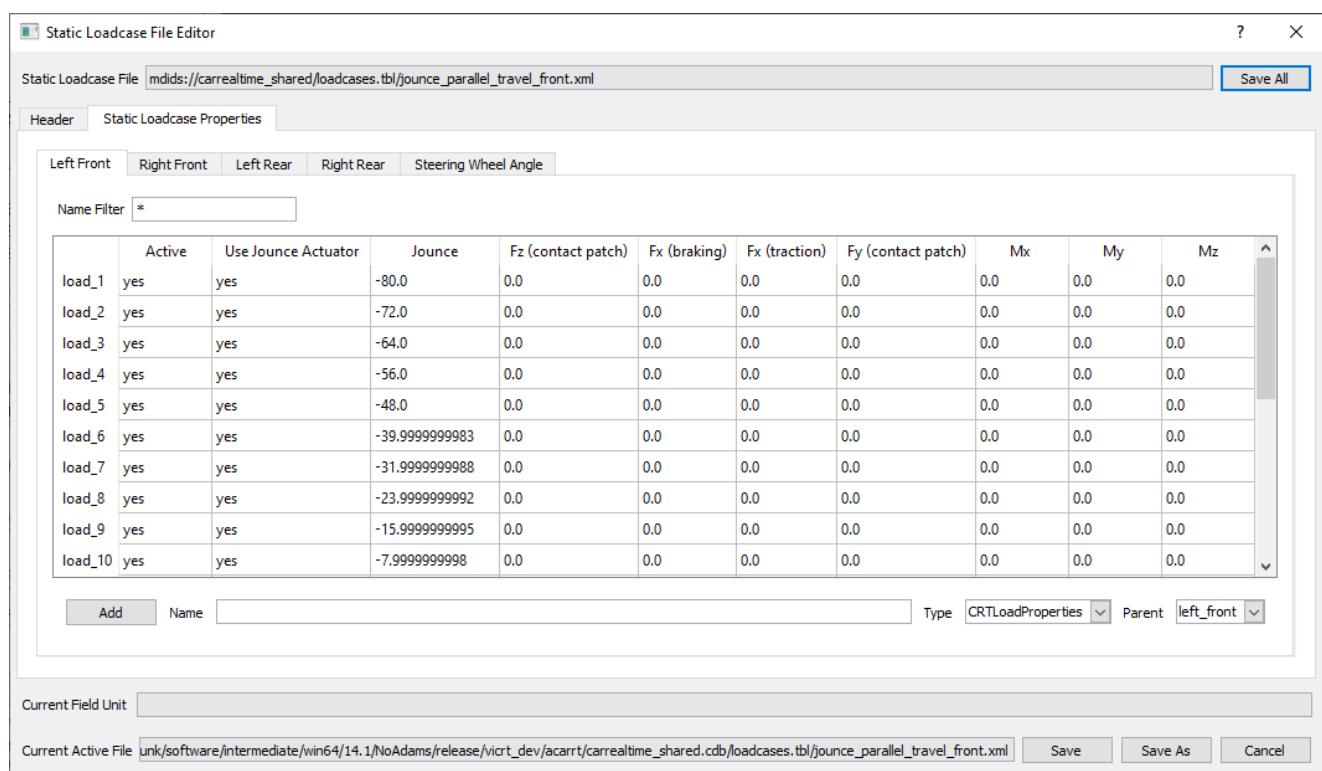
- **Steering wheel angle**

input steering wheel angle.

Forces and moments are applied in the [wheel location](#) reference frame.

The *contact patch* is defined considering the wheel rigid (no tire deformation is taken into account): it is located, with respect to the wheel center, at a distance equal to the unloaded radius of the tire and the direction is given by the camber/toe of the hub.

Loadcase file (xml format) is stored in loadcases.tbl table of databases; it is possible to edit the loadcase file by using a specific property editor.



For each load step a static analysis is performed.

Static analysis is done by means of a dynamic simulation which continues until the kinetic energy in the model falls below a given threshold or a time end condition is reached.

The thresholds are controlled in the system parameter tree of the vehicle ([Build Mode](#)).
See [Static Parameters](#) for further info.

Xternal

The events in this unit are used to generate input files needed to run VI-CarRealTime coupled with external applications (e.g. Driving Simulators).

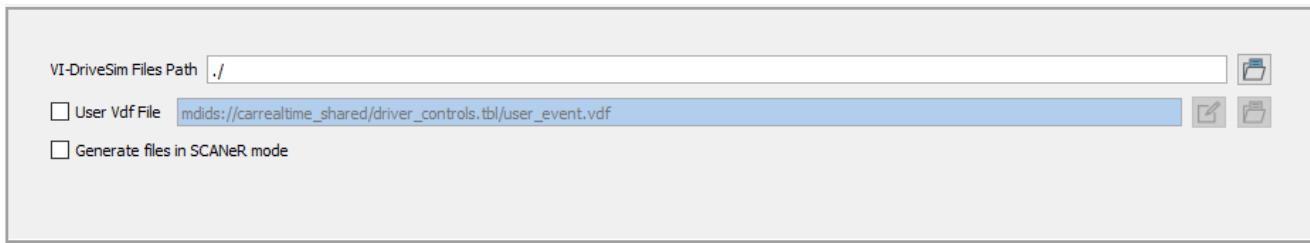
- [VI-DriveSim](#)
- [VirtualTestDrive](#)
- [SCAneR](#)

VI-DriveSim

VI-DriveSim event is used to generate the input files for the VI-DriveSim driving simulator.

The user will need to specify the target directory for the generated VI-DriveSim input file. It is also required to specify the road data file for the simulation that will be also used for retrieving the correct road Graphics in VI-DriveSim.

VI-CarRealTime



In order to avoid the direct editing of the generated file it is required that both the computer on which is running VI-CarRealTime and the one running the solver session of VI-DriveSim will have as a shared directory in which input file will have to be targeted.

It is also required that VI-DriveSim and VI-CarRealTime have the same set of registered databases.

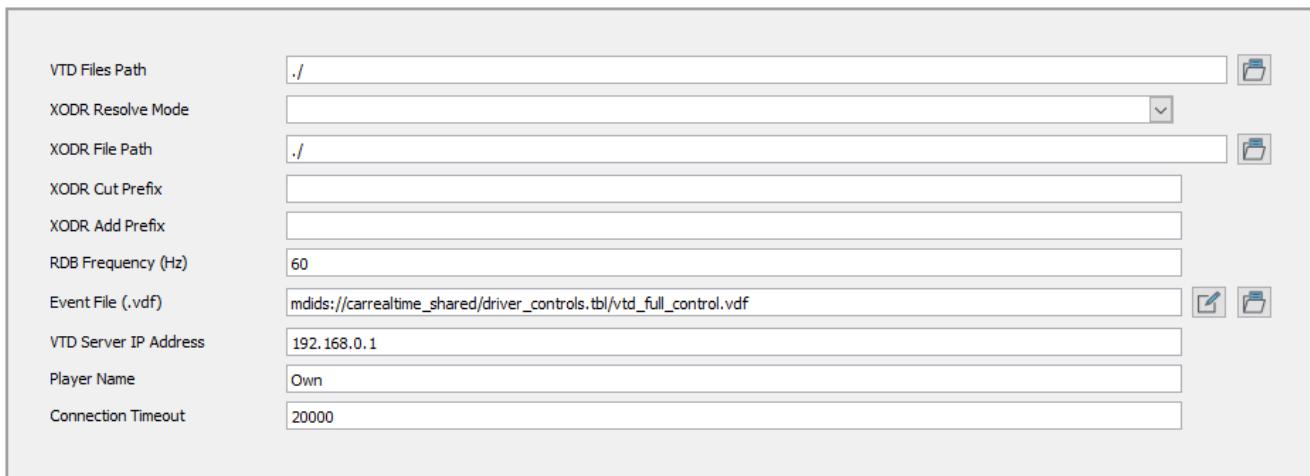
Note: when submitting an example event to test/use VI-DriveSim, it's strongly recommended to use one of the shared models (CityCar, CompactCar, RaceCar, SedanCar, SportCar, SUV).

When VI-DriveSim operates in combination with SCANEr Studio, please make sure to check the toggle **Generate files in SCANEr Mode**. This option forces SCANEr compliant settings for the [std_tire_ref](#) parameter as well as the generation of the SCANEr interface file (.vict) for VI-CarRealTime models.

VirtualTestDrive

VirtualTestDrive event is used to generate the input files for the VI-CarRealTime interface with Vires VirtualTestDrive.

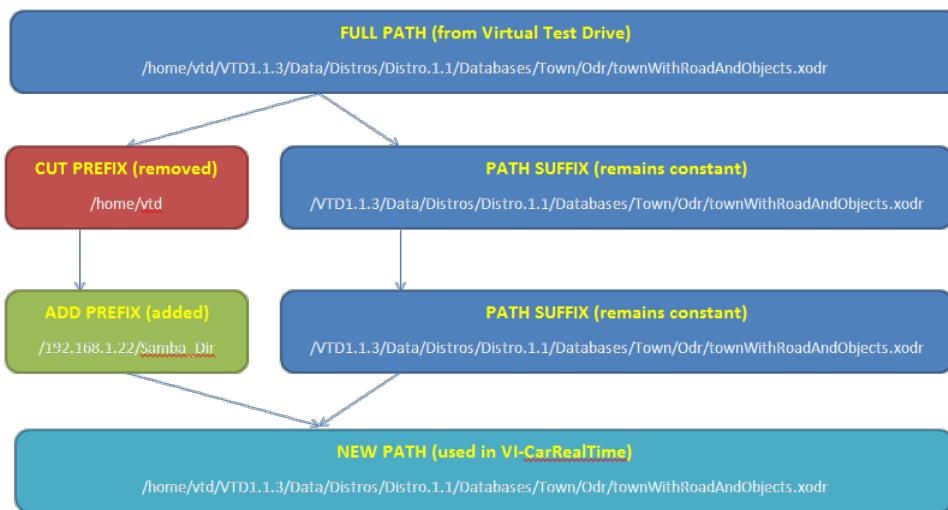
The user will need to specify the target directory for the generated input files. It is also required to specify a few additional information for the configuration of the socket communication between VirtualTestDrive and the VI-CarRealTime solver.



The available parameters for the VirtualTestDrive event are:

- **VTD Files Path**
specifies the path where the simulation input files will be written.
- **XODR Resolve Mode**
specifies the method to resolve the path to reach the road file (*.xodr). It has three options
 - vtd_path* : The xodr file will be located using the full path specified by VirtualTestDrive.
 - local_path* : The xodr file will be located using the local path specified by the user in the field 'XODR File Path'

`custom_path` : The xodr is located using a path generated modifying the full path specified by Virtual Test Drive. The portion of path specified in the 'XODR Cut Prefix' is replaced by the path specified in 'XODR Add Prefix'. Procedure used to build the xodr location is shown in picture below:



- **XODR File Path**

Specifies the path on the local Windows machine, where the road file (*.xodr) will be searched. It is only available when 'XODR Resolve Mode' is set to 'local_path'

- **XODR Cut Prefix**

Used to build the path where the road file (*.xodr) will be searched, as explained above. It is only available when 'XODR Resolve Mode' is set to 'custom_path'

- **XODR Add Prefix**

Used to build the path where the road file (*.xodr) will be searched, as explained above. It is only available when 'XODR Resolve Mode' is set to 'custom_path'

- **Event File (.vdf)**

specifies the VI-Driver file used to control the event. VI-CarRealTime provides different vdf files examples to be used in conjunction with VirtualTestDrive:

`vtd_full_control`: this file is setup to control the VI-CarRealTime vehicle using longitudinal controller inputs and lateral controller inputs coming from VirtualTestDrive

`vtd_lat_ctrl`: this file is setup to control the VI-CarRealTime vehicle using only lateral controller inputs coming from VirtualTestDrive

`vtd_long_control`: this file is setup to control the VI-CarRealTime vehicle using only longitudinal controller inputs coming from VirtualTestDrive

- **VTD Server IP Address**

specifies the address where VirtualTestDrive is running

- **Player Name**

specifies the name of the VirtualTestDrive player that will be animated by the VI-CarRealTime dynamics.

- **Connection Timeout**

specifies timeout in milliseconds after which VI-CarRealTime Solver stops waiting for informations from VirtualTestDrive. It is used to avoid the application to hang due to problems in socket communication.

When running VirtualTestDrive event VI-CarRealTime runs in co-simulation with Vires Virtual Test Drive. In order for the simulation to run properly, there are few model parameters that must be properly specified, mostly located in the [Vehicle Model Properties](#) parameter tree.

Plugin

- **active**

VI-CarRealTime

specifies that a user library must be used for this event. Must be set to 1.

- **library**

specifies the name of the custom solver library to be used (when active is set to 1). Must be set to `crt_vtdll`.

Circular_buffer

- **activity_flag**

specifies whether circular buffer should be enabled or not. Since the duration of the run is unknown when the event starts, circular buffer must be enabled, to prevent allocation of a huge memory area. Must be set to 1.

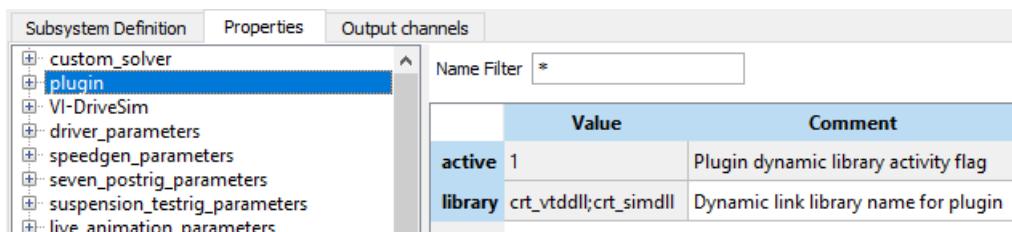
- **dimension**

specifies the size of the buffer, in seconds. This is up to the user, it may range from a few seconds to several minutes. The higher the length of the buffer, the higher will be the section of simulation that will be stored on output files for post processing.

Run VirtualTestDrive with VI-DriveSim

It is possible to run a VirtualTestDrive event in combination with VI-DriveSim. In such case both VirtualTestDrive library (`crt_vtdll.dll`) and VI-DriveSim library (`crt_simdll.dll`) must be set in plugin library field before submitting the VirtualTestDrive event. In order for both libraries to be correctly read it's important that VirtualTestDrive library is specified before VI-DriveSim one.

The two libraries must be separated by a semicolon (;) and **active** field must be set to 1, as shown in the picture below:



The **modes of simulation** available for the VirtualTestDrive event are:

- **files only**
- **interactive (only available to run VI-CarRealTime solver on windows)**

To run the simulation on **Windows**:

- **Interactive Mode**

After loading components on Virtual Test Drive environment, the simulation is started as for any other event. If the model is properly configured as explained in the [Model Setup](#) section the communication between VI-CarRealTime standalone solver and Virtual Test Drive will start automatically.

- **Files Only Mode (Standalone and Matlab modes)**

After loading components on Virtual Test Drive environment, you can start VI-CarRealTime standalone solver from command prompt, typing the command:

```
vigXX acreal ru-so <vicrt_inputfile>
```

where XX must be replaced by the installed VI-CarRealTime version, and `<vicrt_inputfile>` is the `*_send_svm.xml` generated when submitting the event in VI-CarRealTime.

If instead you want to run the Matlab interface of the VI-CarRealTime solver, you only need to press the "Play" button in the Simulink model, once it is properly setup as explained in the [Matlab Interface](#) section.

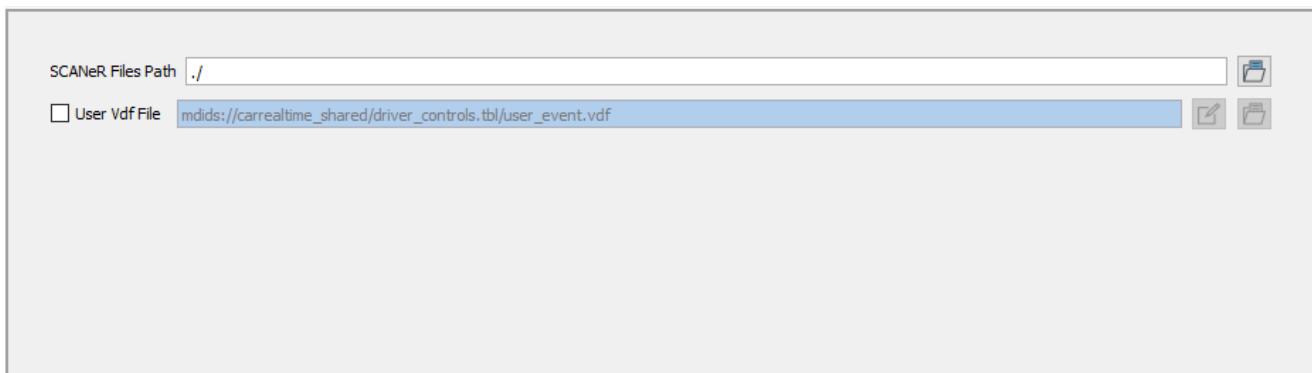
To run the simulation on **Linux**:

- **Files Only Mode**

VI-CarRealTime is automatically loaded as a component (if the Virtual Test Drive simulation environment is properly setup).

SCANeR

VI-CarRealTime supports a direct interface to the [SCANeR Studio](#) for implementing ADAS and traffic simulations.



Please note that it is recommended to enable the clutch model in the VI-CarRealTime model before submitting the event.

The SCANE R event does not require parameters.

Once the event is executed, 2 files are generated:

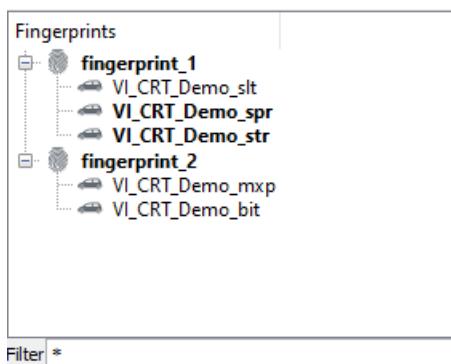
- VI-CarRealTime solver input file in the working directory
- SCANE R external vehicle descriptor (.vict) in the SCANE R Files Path

The latter should be loaded in SCANE R Studio for completing the coupling with VI-CarRealTime.

Fingerprints

Fingerprints are used in VI-CarRealTime to group a set of event instances.

Event instances are shown together in fingerprint treeview and can be saved to external file (XML format).



The following actions, related to fingerprint and events management, are allowed in Test Mode:

- **Load Fingerprint**

Allows to Load in VI-CarRealTime session a saved fingerprint file.

- **New Fingerprint**

Creates a new fingerprint object in the session.

- **Copy Selected Fingerprint**

Make a copy in session of the selected fingerprint. All the events will be copied in the new fingerprint, whose name will have a _copy suffix.

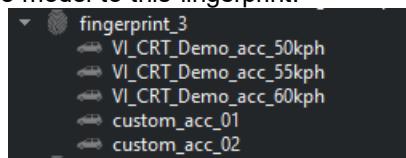
- **Replace Model for Full Vehicle Events**

Allows to replace, in all the events of the selected fingerprint, the model that will be used to run each event. The event name will be updated based on the system name and on the event type.

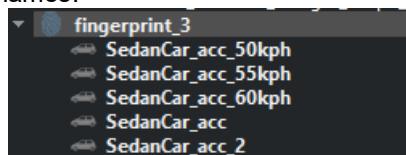
In particular, if the event name is in the form: <model_name>_<user_defined_suffix>, only the model name is replaced.

For any other name in which the model name is not present, the standard convention is used and the event name is: <model_name>_<event_id>

As an example, replacing the model to this fingerprint:



leads to the following event names:



where the first three events retain the original name (except of course for <model_name>, being changed from VI_CRT_Demo to SedanCar), and the last two have been recreated as if they were new events.

- **Run Selected Events**

Run all the event of the selected fingerprint if the fingerprint object is selected. If only specific events have been selected, only these ones will be run.

- **Delete Selected Events/Fingerprints**

Remove the selected fingerprint or events from VI-CarRealTime session. The action does not remove a fingerprint file already saved in the local disk.

- **Save Selected Fingerprints**

Save the fingerprint as a XML file in VI-CarRealTime current working directory. The file name will be equal to the fingerprint name in session.

- **Save Fingerprint As**

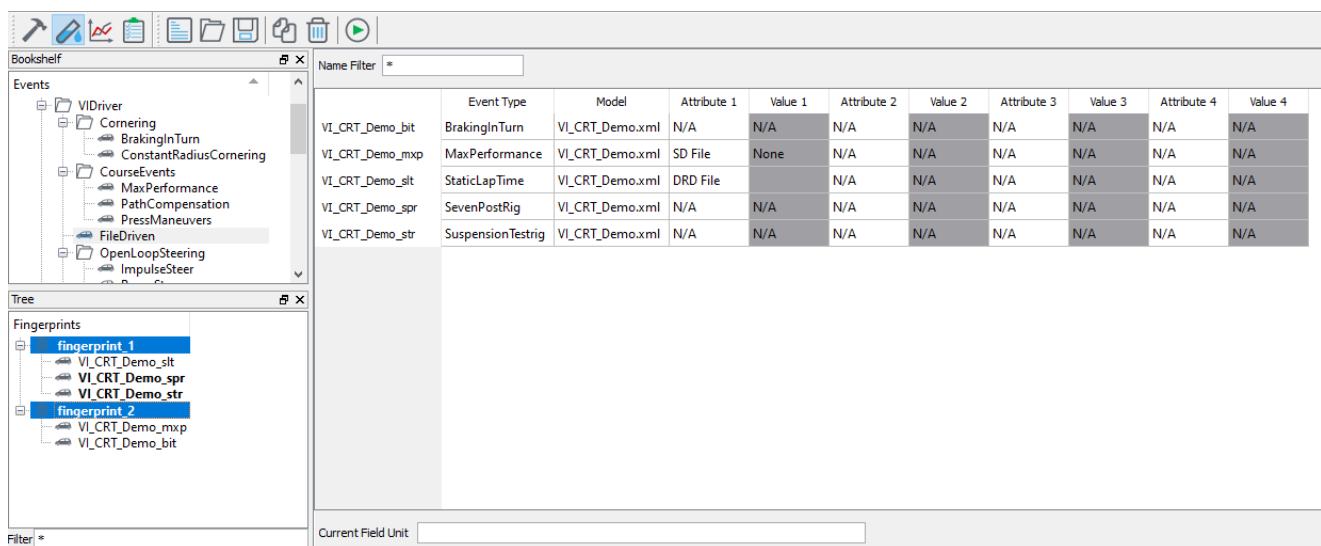
Save the fingerprint in the directory selected by the user and with the file name specified by the user.

- **Rename**

Change the name of the selected fingerprint or event.

Sets of multiple simulations can be run by the starting solution process when multiple event instances are selected.

When multiple events are selected to run they will be processed in the order that they appear in the fingerprint tree top to bottom.



Run Analysis In Batch Mode

Run VI-CarRealTime simulation in batch mode (standalone)

It is possible to run VI-CarRealTime in batch mode by typing this command from a dos shell:

```
vig20 acreal ru-st b <fingerprint_file_name>
```

where:

- **<fingerprint_file_name>** is a valid name for an existing [fingerprint](#) xml file.

Note: In order to successfully run the simulations all the databases referred in the fingerprint must be registered and stored in `vicrt_20_defaultprefs.xml` file.

Executing the simulations of a fingerprint using this approach is equivalent to run them from the VI-CarRealTime GUI. The referenced models are loaded from the database, simulation files are created and executed in batch mode.

Run VI-CarRealTime solvers in batch mode

It is also possible to execute single events launching one of the VI-CarRealTime solver and providing a beforehand created input file.

- VI-CarRealTime Standalone Solver, for [Driving Machine](#), [VI-Driver](#), and [Suspension Testrig](#) events:

```
vig20 acreal ru-so <input file>
```

where **<input_file>** is a valid input send file (*.send_svm.xml) that is generated using the `files_only` option in VI-CarRealTime [Test Mode](#).

- VI-CarRealTime [Static Lap Time](#) events:

```
vig20 acreal ru-slt <xsg_input_file>
```

where **<xsg_input_file>** is a valid Static Lap Time input send file (*.xsg) that is generated using the `files_only` option in VI-CarRealTime [Test Mode](#).

- VI-CarRealTime [Press Maneuver](#) events:

```
vig20 acreal ru-pm <input_file>
```

where **<input_file>** is a valid press maneuver input send file (*send_svm.xml) that is generated using the files_only option in VI-CarRealTime [Test Mode](#).

- VI-CarRealTime [Max Performance](#) events:

```
vig20 acreal ru-mxp <mxp_input_file>
```

where **<mxp_input_file>** is a valid Max Performance input file (*.mpx) that is generated using the files_only option in VI-CarRealTime [Test Mode](#).

To submit a Max Performance event in conjunction with a prerecorded history file (*.xsh):

```
<mxp_batch> <xsh_input_file>
```

where **<mxp_batch>** is Max Performance batch file generated in the working directory and **<xsh_input_file>** is the history file.

Note: In order to successfully run the simulations all the databases referenced in the input files must be registered and stored in a [*vicrt_cdb.cfg*](#) file available in the working dir.

Run VI-CarRealTime in batch mode (MATLAB/Simulink)

The batch call to VI-CarRealTime solver is also possible from MATLAB/Simulink calling the function from Matlab Command Window or from a Matlab script file:

```
runcrt(<fpCellVar>,<mdlName>)
```

where:

- **<fpCellVar>** is a cell variable containing the string with the path of a valid fingerprint file. As an example, if a fingerprint named *fing_1.xml* is stored in the working directory, in Matlab **<fpCellVar>** variable must be defined as *fpCellVar = {'fing_1.xml'}*.
- **<mdlName>** is an optional argument in which the user can enter the string name of the Simulink Model File to use for running the simulation. If such argument is not entered, the function uses *vicrt_passive* model (stored in *\$VI-CarRealTimeInstallationdir\carrt\examples\Simulink*). As an example, for using *active_vehicle.mdl* as Simulink model, in Matlab **<mdlName>** variable must be defined as *mdlName = 'active_vehicle.mdl'* .

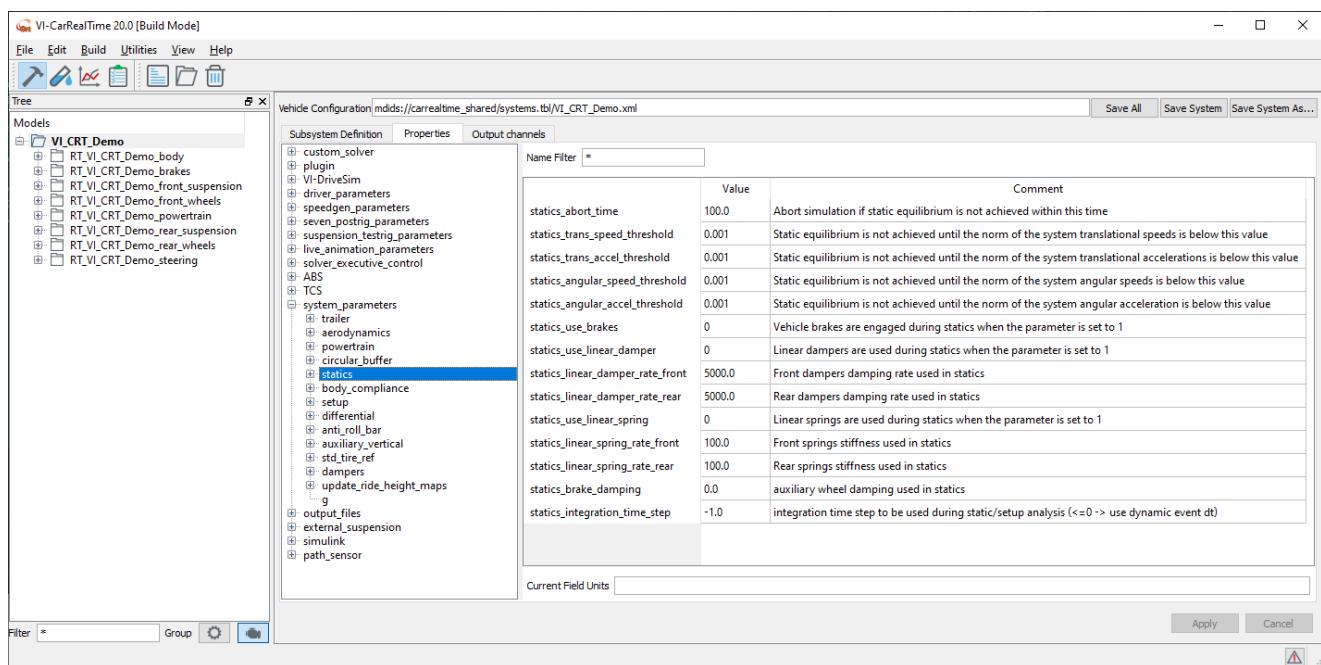
Note: in VI-CarRealTime [Test Mode](#), when defining the events to be stored in the fingerprint, choose *files_only* as [Mode of Simulation](#).

Note: *runcrt* function needs *vicrt_inputfile* variable to be defined as global in Matlab Command Window. *vicrt_inputfile* variable must be a string variable referring to **send_svm.xml* file.

Static Analysis And Vehicle Setup

The present topic is meant to give a detailed description of the statics and setup procedure performed on the vehicle model in VI-CarRealTime.

In VI-CarRealTime the static analysis is done by mean of a special dynamic analysis in which the contributes of aerodynamic drag and vehicle driveline are deactivated. The dynamic analysis is run until the kinetic energy of the system drops below a given threshold.



During the statics phase the tires only react along the road normal direction, behaving as a vertical spring damper (`USE_MODE = 0`); for this reason they permit to achieve a successful static on a flat road.

In order to obtain statics convergence on a 3D road, the solver establishes at runtime the conditions existing for a flat road.

At each integration step the solver computes the normal direction for the instantaneous road plane passing for three tires's contact patches.

The computed direction is used as unit vector for the gravitational field.

The vehicle can reach the static solution (also including setup) under the newly defined gravity, by simply using the tire normal forces.

Before the vehicle can perform a dynamic analysis the gravitational field is restored to the original one.

In order to accomplish convergence under standard conditions the vehicle model runs an additional static analysis during which three PID controlled forces lock the degrees of freedom that can't be reacted by the tires:

- chassis translations along x y plane;
- chassis rotation around z direction (yaw angle).

The controlled forces will be deactivated before the actual dynamic analysis starts.

VI-CarRealTime Vehicle Set-up

VI-CarRealTime features the capability of adjusting suspension and vehicle characteristics before a dynamic analysis. The practice, usual in racing applications, is implemented in VI-CarRealTime by modifying the values that the adjustable quantities assume after static equilibrium of the vehicle and it is achieved by using different methods depending on vehicle and suspension architecture.

The available setup adjustments are:

- [Ride height](#)
- [Cross weight](#)
- [Toe Angle](#)
- [Camber Angle](#)
- [Pin height](#)

- [Setup/Running masses](#)

Ride Height

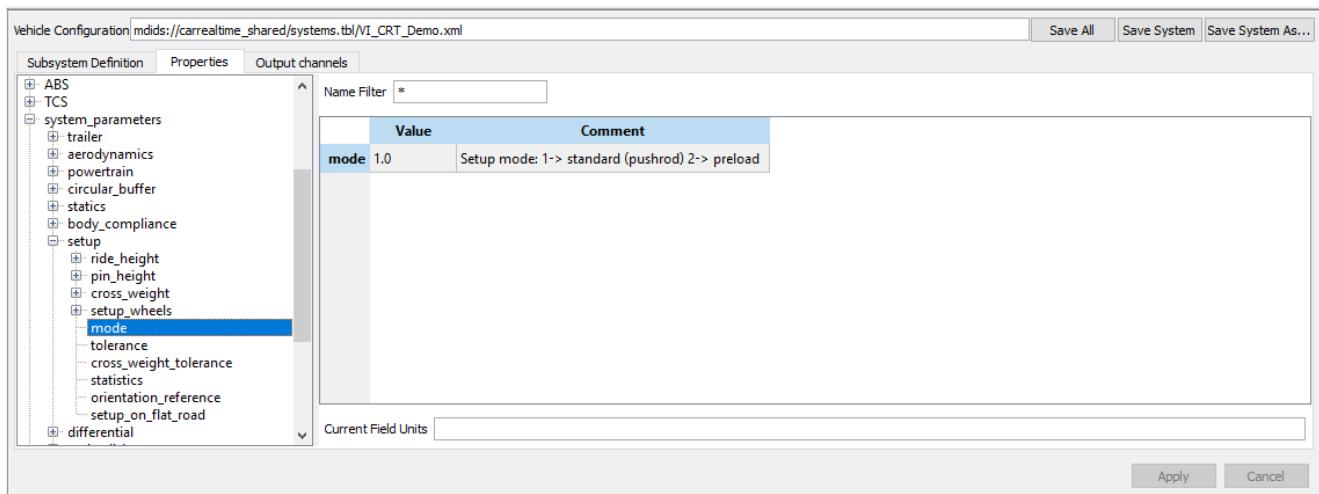
Ride height adjustment is achieved by modifying the vertical distance from the ground of specific points belonging to the body chassis. In sports-car disciplines the practice is important in order to tailor aero forces (which typically are depending on the distance of skid plate from ground), to avoid skid plate impact on the ground, and to tune the vehicle CG vertical position.

The adjustment in VI-CarRealTime is done by identifying two sensor points (belonging to the vehicle sprung part): one on the front suspension and the other one on the rear. For each sensor point it is possible to specify the position in the VI-CarRealTime global reference frame and the desired ride height.

In actual vehicles this kind of adjustments is done by means of:

- Push-rod length adjustments (mode 1);
- Spring pre-load (mode 2) for vehicle without push rods such as McPherson or double wishbone suspensions.

The main difference of the two methods is that in presence of a push rod the position of chassis part can be positioned w.r.t. the ground without affecting spring component loads. The suspension type is specified in system parameters tree as shown:



Cross Weight

Cross weight adjustment is achieved by modifying the wheel normal load in order to achieve the desired target. There are four methods available:

- Front Force ($CW = FrontRight_TireFz - FrontLeft_TireFz$);
- Rear Force ($CW = RearRight_TireFz - RearLeft_TireFz$);
- Ratio ($CW = (FrontLeft_TireFz - RearRight_TireFz) / Sum(TireFz)$);
- Cross Ratio ($CW = (FrontRight_TireFz - RearLeft_TireFz) / Sum(TireFz)$).

Toe Angle

Toe angle adjustment is achieved by adding a constant value to orientation kinematic splines in order to have the desired value of toe (wrt ground or chassis) after the setup.

Camber Angle

Camber angle adjustment is achieved by adding a constant value to orientation kinematic splines in order to have the desired value of camber (wrt ground or chassis) after the setup.

Pin Height

Pin height adjustment is achieved by modifying the spring motion ratio in order to achieve the spring target length after the setup.

Setup/Running Masses

In motorsport applications it is a common practice to have setup conditions different from running conditions. In order to do this, ballasts fuel and driver masses are added to the vehicle at the right time.

Setup Procedure

Setup phase in VI-CarRealTime is performed through a sequence of static equilibrium analyses during which all the targets specified ([ride height](#), [cross weight](#), [camber](#), [toe angles](#) and [pin height](#)) are all adjusted at the same time by mean of several PID controllers.

More in details the sequence of operation is the following:

- **Static analysis at design time.**
This analysis is performed using the model at design condition, without adding any additional setup mass.
- **Static analysis with setup masses.**
The setup masses are added to vehicle and a static is performed.
- **Setup analysis.**
This phase runs until the various target are achieved or until the maximum number of setup analysis iterations (3) is reached.

If the ride height mode is set to Preload, 2 PID controllers are added in series to spring forces, one for Ride Height (RH) target and the other for the cross weight target.

The RH's PID are active at the same time for each spring and have different targets due to spring's positions. The PID for the front left spring is responsible for the front left target (as the right spring for the front right side) and both the rear springs are responsible for the rear ride height target.

The activity of cross weight PIDs is related to the Cross Weight mode chosen.

More in detail: if the cross weight is set to **front_force_mode** only the front PIDs will be activated; when the Cross Weight mode is set to **rear_force_mode**, the rear springs PIDs will be activated and finally, for the **ratio_mode**, PIDs will be activated for front left spring and rear right spring.

At each integration step the solver computes the error between real value and target value (ε), its time derivative ($d\varepsilon$) and its time integral ($i\varepsilon$).

The resulting PID force is:

$$F_{PID} = -K_p \varepsilon - K_d d\varepsilon - K_i i\varepsilon$$

The sum of the contributes of the two PIDs (RH and CW) is the desired preload needed by spring to respect the target.

Moreover, in order to achieve the target for toe and camber angles, at each integration step the solver computes the needed offset for the kinematic orientation splines.

If the ride height mode is set to Pushrod, the PIDs are located as in the Preload mode, but their values are used

as correction for the suspension jounce used in the lookup table for spring motion ratio. If pin height adjustment is also active, at each integration step solver computes the spring preload needed to maintain the spring at the desired deflection.

- Static analysis with running masses.
The running masses are added to vehicle and a static analysis is performed.
 - Static with down forces.
This is the last static, it is performed adding to the vehicle the down forces it will have at the first step of dynamic analysis.

Straight Setup

In VI-CarRealTime dynamic analysis it is possible to detect transient dynamics in system states at the beginning of simulation. The effect is mainly due to interactions of tires, driver and aerodynamic forces with the vehicle model. In fact during the static equilibrium that preludes a dynamic solution, tires work in *mode 0*, implying that they are able to react only along road surface vertical direction.

When dynamic simulation starts, all force and torque components become active and the car requires a transient phase in order to find the dynamic equilibrium under the new external loads configuration. Moreover, at simulation start the longitudinal aerodynamic effect, not present during the static analysis, is typically applied to vehicle.

Finally, as the vehicle must follow an imposed trajectory with a target speed profile, in order to reduce the initial transient phase, an estimation for steering demand and throttle should be required.

In order to support straight line setup no changes are done to existent static and suspension setup phases. It is then introduced an *additional and optional procedure* for the straight setup before starting with the actual dynamic event.

The feature is controlled in the event vdf file through the key **INITIAL SETUP='STRAIGHT'** in STARTUP block.

```
$-----[STARTUP]-----$  
[STARTUP]  
STARTING_TIME = 0  
INITIAL_SPEED = 30  
INITIAL_STEERING = 0  
INITIAL_THROTTLE = 0  
INITIAL_BRAKING = 0  
INITIAL_BRAKING2 = 0  
INITIAL_GEAR = 1  
INITIAL_CLUTCH = 0  
INITIAL_SETUP = 'STRAIGHT'  
$-----[MANEUVERS LIST]-----$
```

The functionality works as follows: after statics and suspension setup phases have been executed, a pre-dynamic event of the full vehicle model, using the same tires, road, aerodynamic and driver configuration of following full vehicle analysis is run.

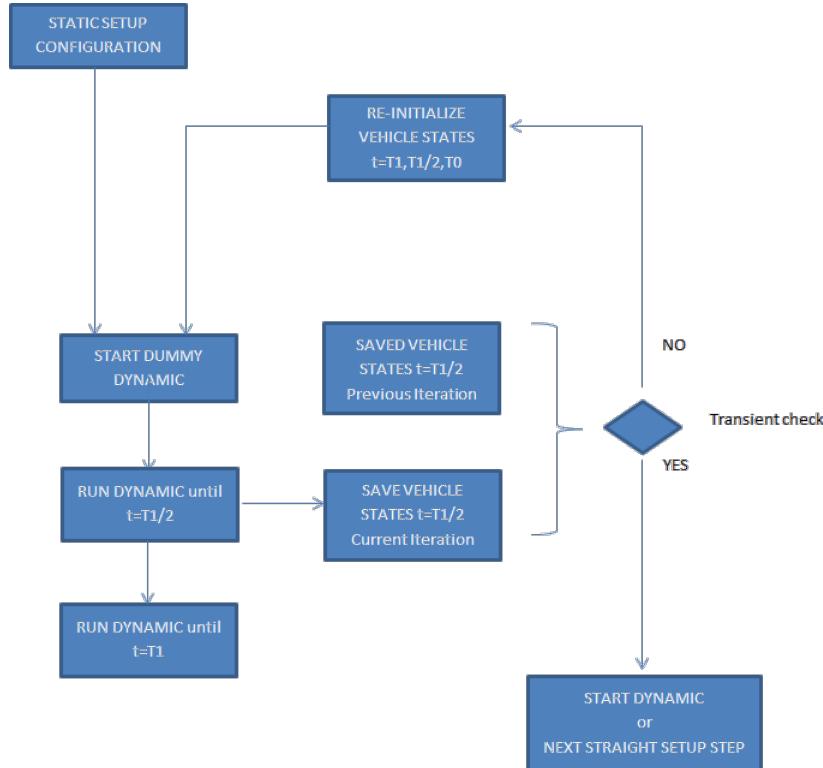
During the procedure any smoothing time for aerodynamic is deactivated in order to take into account of their full contribute to dynamic equilibrium. The procedure initializes the vehicle state, tire forces and driver startup values in order to setup the vehicle at the starting location under the correct initial conditions for dynamic event.

An iterative approach is adopted during the straight setup phase, consisting of the following steps:

1. Vehicle states are saved before the dynamic starts; simulation time is set at $T_0=0.0$.
 2. VI-CarRealTime vehicle model is simulated for the first T_1 seconds of analysis (T_1 is the time needed to perform a distance equal to target path value = 10 meters) with guess value for driver demands. The time elapsed from $T=0$ to $T=T_1$ constitutes an iteration.
 3. During the iteration driver demands are controlled in order to achieve the desired target according to the strategy described in the [startup on straight line](#) and [startup on generic path](#) topics.
 4. Three sets of vehicle states are collected for each iteration: at start time, at half target path and at the end of iteration path.

5. Vehicle states snapshots at half target path of two subsequent iterations are compared. If the difference of *longitudinal* and *lateral slips* is not within a tolerance, combination of states of time T_1 , T_0 is used to initialize the vehicle and the steps 2 and 3 are repeated after positioning the vehicle at initial location. The iterative procedure will last until the check will be successful or a maximum number of iterations will be reached. The following tolerance targets are set: Tolerance[$(\text{value_iteration_i-1} - \text{value_iteration_i}) / \text{value_iteration_i}$]: Longitudinal Slip: 0.1% Lateral Slip: 0.15% .
6. The converged states, if found, represent the initial condition for the next dynamic event.

The following picture represents a flowchart of the procedure implemented:



The procedure is executed in different ways depending on the curvature of target path to be used during the first dynamic mini-maneuver of vdf file.

Startup On Straight Line

In the straight line path case, targets for startup maneuver are longitudinal velocity and longitudinal acceleration. Using such information a specific VI-Driver event is created.

This event will include closed loop controllers for all driver demands:

1. Lateral controller will follow a straight line;
2. Longitudinal controller will maintain vehicle longitudinal velocity at target value.

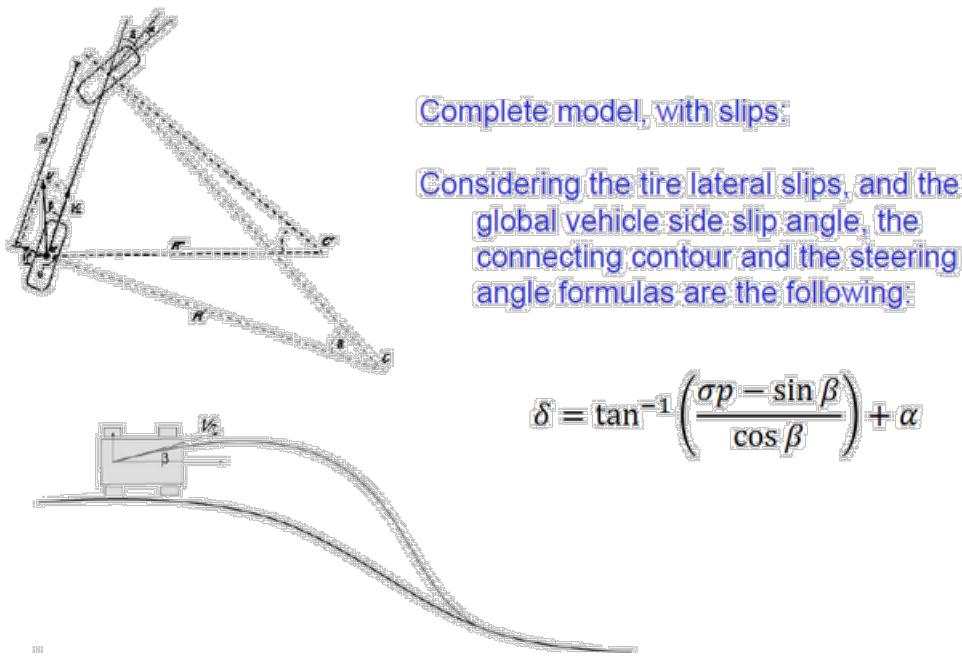
Longitudinal acceleration will be also taken into account by altering the gravitational field, adding a longitudinal contribute (along vehicle heading direction) corresponding to target acceleration.

States will be updated until convergence check will achieved or maximum number of iterations (200) is reached.

Startup On Generic Path

In such case targets for startup maneuver are longitudinal velocity and acceleration, lateral velocity and yaw rate.

A preliminary [startup on straight line](#) will be performed. Altered gravitational field also includes target lateral acceleration. After achieving convergence, a theoretical value for steering angle is computed, according to the expression below:



A set of open loop iterations is then performed in order to lead the current steering angle value to the target one.

For each iteration during this stage, the throttle is maintained constant and steer is increased using a step signal with max increment of 5 degrees per iteration.

Step steer iterations are alternated with constant steer ones in order to stabilize the system.
When target value for steer is achieved, the value is tuned in order to match target yaw rate.

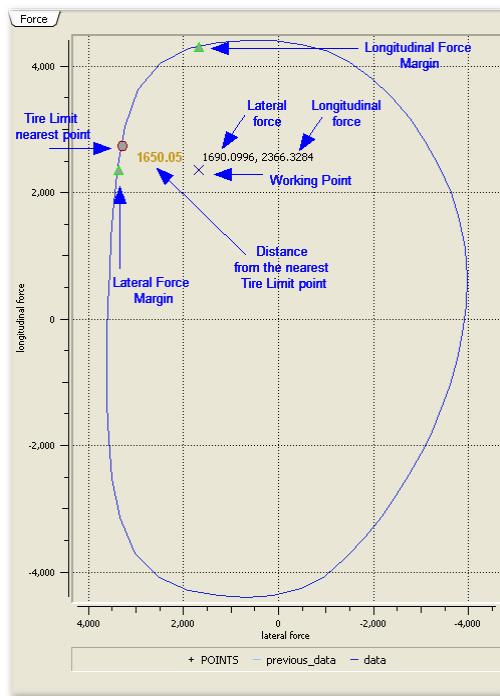
Finally, a set of open loop analysis is performed keeping constant all driver inputs, in order to stabilize all vehicle states. This phase is stopped when a check on slip tolerance (or maximum number of iterations) is achieved; multiple iterations are used also in this case to lead the vehicle to the target conditions by smooth application of the steering signal, without going far from simulation start point (to be used for subsequent dynamics).

Tire Limits

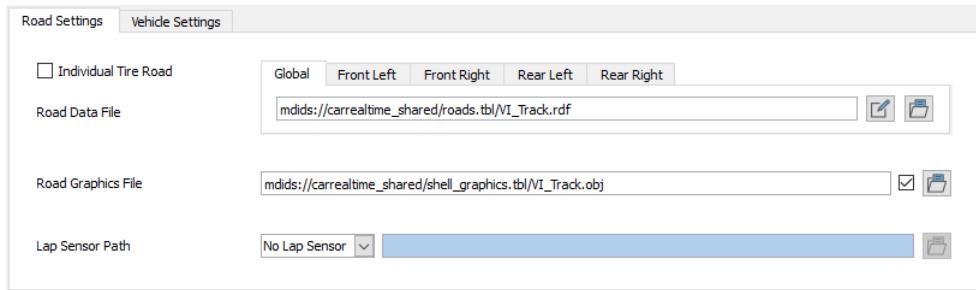
When dealing with tire forces it becomes important to know how far a specific tire condition is from its limit grip, because it potentially represents a stability issue for the vehicle.

The Tire Limits feature in VI-CarRealTime allows to calculate, during a dynamic simulation, the tire limit states compared to the actual ones met by the tire.

Looking to the figure below, a given working point, depending on tire kinematic states (slip, penetration, camber etc.) will define a combination of longitudinal and lateral force. Depending on tire model and properties it will be possible to determine a limit curve (related to vertical load and camber angle). From the working point it will be possible to approach to the limit by changing the longitudinal force (green triangle above working point), lateral force (green triangle on left of working point) or a combination of them; the minimum force increase is identified by the red circle (Tire Limit nearest point).



The Tire Limit functionality is activated through the Active toggle in Tire Limits container of the event property editor; at the beginning of the simulation the tool will compute a map of limit states in a range and with a resolution defined by specific parameters. During the dynamic simulation a set of channels will be computed for each vehicle tire and included in analysis results.



The following parameters can be set from the GUI for the Tire Limits tool:

- **Active**

The check box set the activity of the computation during dynamic analyses .

- **Vertical Force**

The tab contains the vertical force parameters needed for the creation of tire limit map:

- **Minimum**

Minimum value for tire limit map vertical force.

- **Maximum**

Maximum value for tire limit map vertical force.

- **Step**

Force increment step for tire limit map vertical force.

- **Camber Angle**

The tab contains the camber angle parameters needed for the creation of tire limit map:

- **Minimum**

Minimum value for tire limit map camber angle.

VI-CarRealTime**○ Maximum**

Maximum value for tire limit map camber angle.

○ Step

Angle increment step for tire limit map camber angle.

• Lateral Slip

The tab contains the lateral slip parameters needed for the creation of tire limit map:

○ Minimum

Minimum value for tire limit map lateral slip.

○ Maximum

Maximum value for tire limit map lateral slip.

○ Step

Angle increment step for tire limit map lateral slip.

• Longitudinal Slip

The tab contains the longitudinal slip parameters needed for the creation of tire limit map:

○ Minimum

Minimum value for tire limit map longitudinal slip.

○ Maximum

Maximum value for tire limit map longitudinal slip.

○ Step

Increment step for tire limit map longitudinal slip.

The result file generated by the tool will contain, for each tire, the following channels:

margin ΔF : distance between working point and the closest point

marginLon ΔF_x : distance between working point and the longitudinal (x) closest point

marginLat ΔF_y : distance between working point and the lateral (y) closest point

closest_LonSlip Closest point tire data characteristics

closest_LatSlip

closest_LonForce

closest_LatForce

closestLon_LonSlip Longitudinal (x) closest point tire data characteristics

closestLon_LatSlip

closestLon_LonForce

closestLon_LatForce

closestLat_LonSlip Lateral (y) closest point tire data characteristics

closestLat_LatSlip

closestLat_LonForce

closestLat_LatForce

1.2.5 Review Mode

Review mode in VI-CarRealTime is used to post-process results.

Two applications are supported for automatic post processing (results loading + animation). Other post-processors can be used to review the results that VI-CarRealTime produces in several formats.

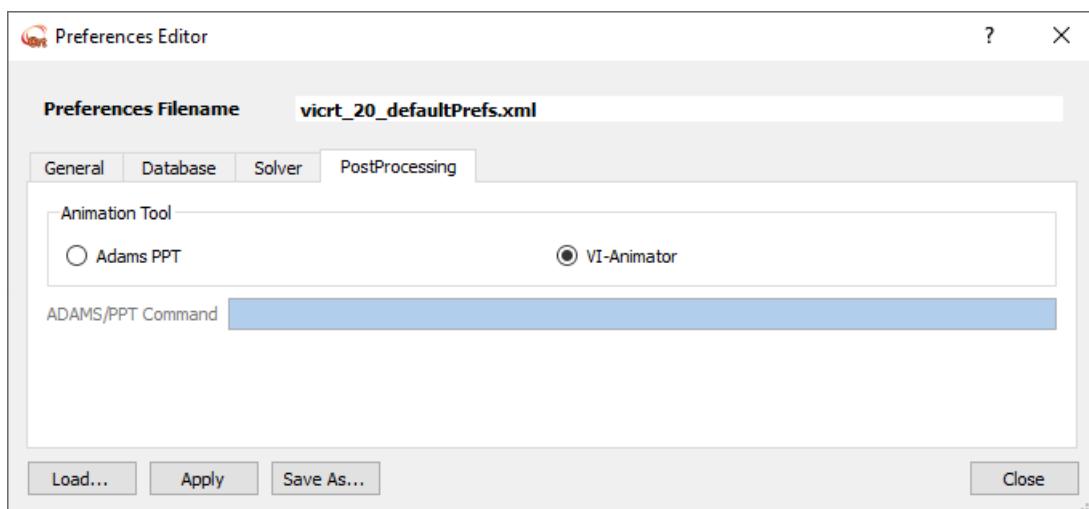
The default postprocessor is:

- **VI-Animator**

Optional postprocessing tool, not part of the VI-CarRealTime installer is:

- Adams/PostProcessor

The Animation tool can be selected via the [preferences editor](#).



Selecting VI-Animator the tool is automatically detected and is ready to be used.

Selecting Adams/PPT as desired Animation Tool the **Adams/PPT Command** field is enabled. You need to specify the full path to the launch script of Adams postprocessor, for example:

C:\Program Files\MSC.Software\Adams\2019_2\bin\appt2019_2.bat

NOTE: Adams/PostProcessor will properly work only if a valid Adams license is available. In case the license is temporary unavailable (e.g. maximum number of users reached), you may experience a delay starting the postprocessor, because the application will be queued for a maximum of 20 minutes waiting for a free license.

To switch to Review Mode, it's sufficient to press button in the Toolbar.

All the events that have been run in Test Mode will be displayed in the Fingerprints section.

To review the simulation results in VI-Animator just select the event and press button.

The following Post Processing Settings are available:

- **Plot Configuration File (.plt)**

Select the xpl file containing a set of custom plots that will be automatically generated in VI-Animator. For all the details about how to deal with xpl files please refer to VI-Animator documentation.

- **Plots**

Flag to load the plot results in the post-processing tool. The flag is always active by default and can't be deactivated.

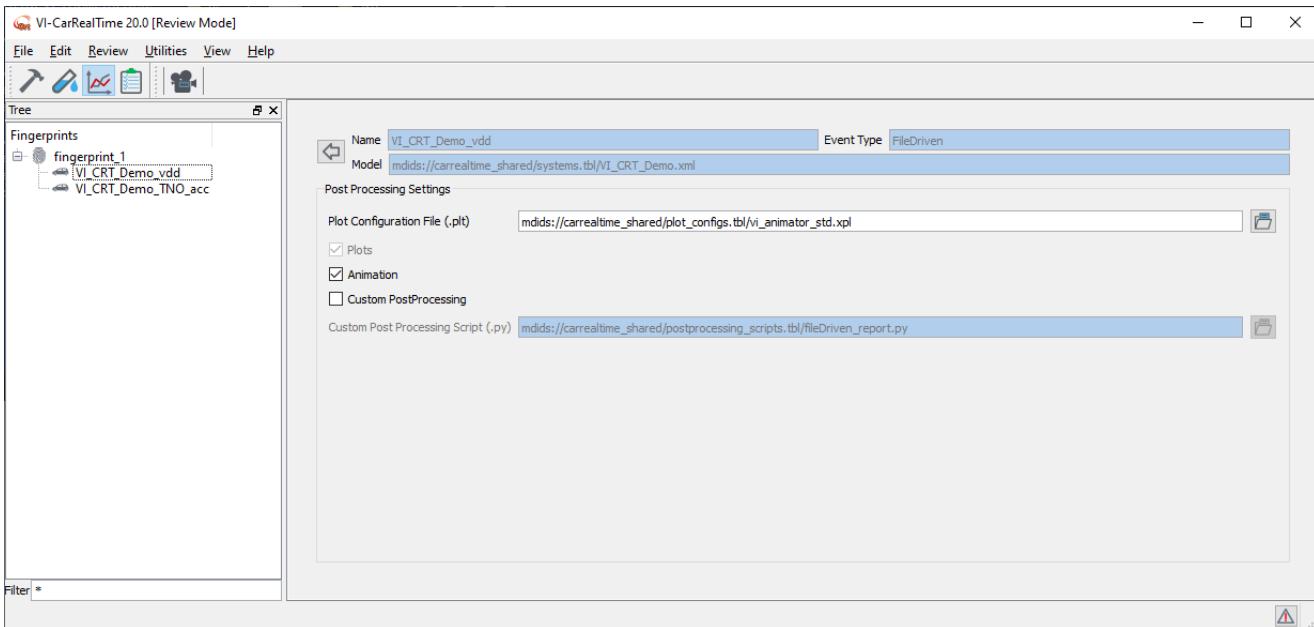
- **Animation**

Flag to load also the animation, together with the plots, in the post-processing tool

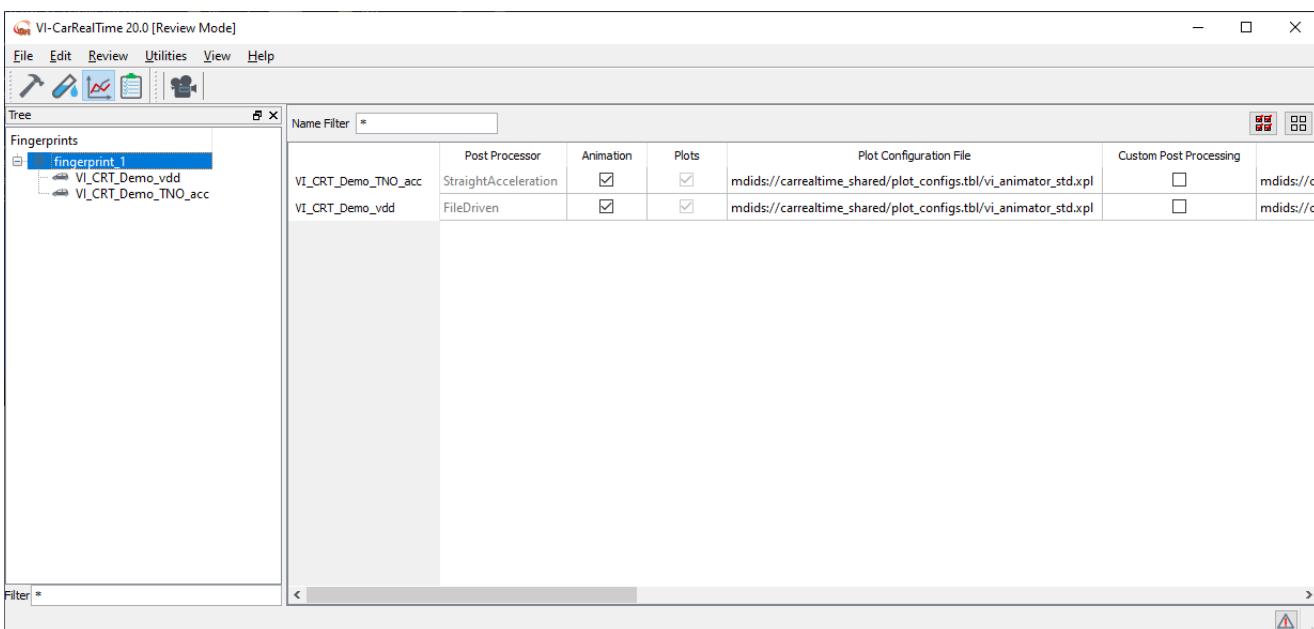
- **Custom PostProcessing**

When the flag is checked, the user has the possibility to select a [Custom Post Processing Script](#) (python format), which is executed when button is pressed. The script can be used to generate custom plots and execute custom actions at python level. Python script examples can be found in carrealtime_shared database (*postprocessing_scripts.tbl*).

VI-CarRealTime



The overlay of multiple simulations is supported in VI-CarRealTime. It's sufficient to select multiple events or directly click on a fingerprint:



A description of the output channels available in VI-CarRealTime can be found [here](#).

Solver Output Files

VI-CarRealTime solver uses the `<analysis_name>_send_svm.xml` file as input to run a submitted [event](#). Such file stores all the information about the system model and the references to the external property file (for example [vdf](#) file) that are needed to perform the submitted event.

After a simulation has been submitted, the following files are available in the working directory:

- **result file** (`.res`)

file embedding the simulation results (requests, part XFORM, etc). By default the res file is written, but the user is allowed to enable different [output file](#) types.

- **report file** (.rep)

here a set of parameters which show the vehicle information at different stages are written. Please look at [rep](#) file format for further details.

- **log file** (.log)

this file summarized the simulation process. It can be used to review the simulation steps to investigate if everything has worked as expected. Furthermore it shows the simulation vehicle statistics and the analysis performance.

- **graphic file** (.xgr)

this file defines the graphic objects that will be used to review the animation in VI-Animator. It defines a connection between the graphic object and the related rigid body XFORM so that the geometry movement during the animation can be defined. This file can be edited in order to apply an offset and/or to scale a graphic object with respect to the related part reference system. Please refer to [Convention and References](#) topic for further details about the reference system of each model part.

- **python file** (.py)

such file is used by VI-Animator to properly setup its environment when an analysis is loaded. It contains, among its commands, references to xgr graphic files (model, tireforces, widget) and to the plot configuration file (xpl). For further details about plot configuration file please refer to VI-Animator documentation.

Custom PostProcessing Script

By using a custom post processing script, the user has the possibility to fully customize his own post-processing relying on python.

If [Custom PostProcessing](#) flag is checked and a valid python script is provided in input, VI-CarRealTime executes it when Execute Postprocessing button  is pressed.

A banner is generated by VI-CarRealTime in Log windows and in command dos shell to show the script status. Here an example:

```
===== CUSTOM POSTPROCESSING STARTED =====
mdids://carrealtime_shared//output_maps.tbl/VI_CRT_Demo_output_map.xml
Executing custom post processing script: mdids://carrealtime_shared/postprocessing_scripts.tbl/rampSteer
===== CUSTOM POSTPROCESSING FINISHED =====
```

Note: the user can add his own *print* functions in python script to customize the banner according to his requirements.

The entry point function for the script is **postProcessing** function; the function is automatically called by VI-CarRealTime and defines the following inputs:

- **resultFile**
reference to simulation result file.
- **outputChannelMapFile [OPTIONAL]**
reference to the xml output map used by the system model for the simulation.
- **systemFile [OPTIONAL]**
reference to the system model used for the simulation.
- **eventType [OPTIONAL]**
type of the event that has been run (i.e. RampSteer, StepSteer, etc).

A set of example python scripts are shipped with carrealtime_shared database (*postprocessing_scripts.tbl*). In particular, by default almost each VI-CarRealTime event has an associated default python script with metrics and plots tailored for the event.

Here the python script example associated with [Ramp Steer](#) event:

█ rampSteer_report.py

```

import os, sys, math, string
from mdi.afc      import vfc
from mdi.visedit import vfc_files
from vigrade.viscripting import viframework
import numpy as np
from vigrade.vipostprocessing.reportUtils import *
from shutil import copyfile

def postProcessing(resultFile, outputChannelMapFile = None, systemFile=None, eventType=None):
    # get html template
    reportTemplate = os.path.join(os.environ['topdir'],'acarrt','carrealtime_shared.cdb','postp
    title          = 'VI-CarRealTime '+str(eventType)+' event report'
    summary         = [['Result File', resultFile],['System File',systemFile]]
    header          = [['Result Name', 'Value', 'Units']]
    pictures        = []
    captions        = []
    # define report file name
    outputPrefix    = string.replace(resultFile,'.res','')
    # loading analysis results
    analysis        = viframework.viobjects.simulation.Analysis('ana')
    anaReader       = viframework.viobjects.simulation.AnalysisReader()
    anaReader.read(resultFile,analysis)
    # retrieving simulation data
    t              = np.asarray(analysis.getChild('time.TIME').getValues())
    vx             = getResultData(analysis,'chassis_velocities_longitudinal', outputChannelMapFile)
    accx           = getResultData(analysis,'chassis_accelerations_longitudinal', outputChannelMapFil
    accy           = getResultData(analysis,'chassis_accelerations_lateral', outputChannelMapFile)
    pth_x          = getResultData(analysis,'driving_machine_monitor_path_x', outputChannelMapFile)
    pth_y          = getResultData(analysis,'driving_machine_monitor_path_y', outputChannelMapFile)
    ssa            = getResultData(analysis,'condition_sensors_side_slip_angle', outputChannelMapFil
    gear           = getResultData(analysis,'driver_demands_gear', outputChannelMapFile)
    brake          = getResultData(analysis,'driver_demands_brake', outputChannelMapFile)
    throttle       = getResultData(analysis,'driver_demands_throttle', outputChannelMapFile)
    swa            = getResultData(analysis,'driver_demands_steering', outputChannelMapFile)
    swt            = getResultData(analysis,'Steering_System_Steering_Wheel_Torque', outputChannelMa
    yawVel         = getResultData(analysis,'chassis_velocities_yaw', outputChannelMapFile)
    pitch          = getResultData(analysis,'chassis_displacements_pitch', outputChannelMapFile)
    roll           = getResultData(analysis,'chassis_displacements_roll', outputChannelMapFile)
    # computing metrics
    data = []
    data = addDataEntry(data,['Laptimes',str(max(t)) , 'sec'])
    data = addDataEntry(data,['Longitudinal velocity maximum',str(max(vx)), 'Km/h'])
    data = addDataEntry(data,['Longitudinal velocity minimum',str(min(vx)), 'Km/h'])
    data = addDataEntry(data,['Steering wheel angle maximum',str(max(swa)*180/math.pi),'deg'])
    data = addDataEntry(data,['Steering wheel angle minimum',str(min(swa)*180/math.pi),'deg'])
    data = addDataEntry(data,['Steering wheel torque maximum',str(max(swt)), 'N-m'])
    data = addDataEntry(data,['Steering wheel torque minimum',str(min(swt)), 'N-m'])
    data = addDataEntry(data,['Roll angle maximum',str(max(roll)*180/math.pi), 'deg'])
    data = addDataEntry(data,['Roll angle minimum',str(min(roll)*180/math.pi), 'deg'])
    data = addDataEntry(data,['Pitch angle maximum',str(max(pitch)*180/math.pi), 'deg'])
    data = addDataEntry(data,['Pitch angle minimum',str(min(pitch) *180/math.pi), 'deg'])
    data = addDataEntry(data,['Yaw velocity maximum',str(max(yawVel)), 'deg/sec'])
    data = addDataEntry(data,['Yaw velocity minimum',str(min(yawVel)), 'deg/sec' ])
    data = addDataEntry(data,['Longitudinal acceleration maximum',str(max(accx)), 'G'])
    data = addDataEntry(data,['Longitudinal acceleration minimum',str(min(accx)), 'G'])
    data = addDataEntry(data,['Lateral acceleration maximum',str(max(accy)), 'G'])
    data = addDataEntry(data,['Lateral acceleration minimum',str(min(accy)), 'G'])
    # creating plots
    if not os.path.exists(outputPrefix+'_files'):
        os.makedirs(outputPrefix+'_files')

    png = outputPrefix+'_files/figure_0.png'

```

```
addPlotEntry(fileName=png,
    x=swa*180/math.pi,
    y=-1.*accy,
    title='Understeer Gradient',
    xlabel='Steering Wheel Angle [deg]',
    ylabel='Lateral Acceleration',
    type = 'plot')

pictures.append(png)
captions.append('')

png = outputPrefix+_files/figure_1.png'
addPlotEntry(fileName=png,
    x=t,
    y=swa*180/math.pi,
    title='Steering Wheel Angle vs Time',
    xlabel='Time [sec]',
    ylabel='Steering Wheel Angle [deg]',
    type = 'plot')

pictures.append(png)
captions.append('')

png = outputPrefix+_files/figure_2.png'
addPlotEntry(fileName=png,
    x=t,
    y=vx,
    title='Longitudinal Velocity vs Time',
    xlabel='Time [sec]',
    ylabel='Longitudinal Velocity [Km/h]',
    type = 'plot')

pictures.append(png)
captions.append('')

png = outputPrefix+_files/figure_3.png'
addPlotEntry(fileName=png,
    x=t,
    y=accy,
    title='Lateral Acceleration vs Time',
    xlabel='Time [sec]',
    ylabel='Lateral Acceleration [G]',
    type = 'plot')

pictures.append(png)
captions.append('')

png = outputPrefix+_files/figure_4.png'
addPlotEntry(fileName=png,
    x=t,
    y=roll*180/math.pi,
    title='Roll Angle vs Time',
    xlabel='Time [sec]',
    ylabel='Roll [deg]',
    type = 'plot')

pictures.append(png)
captions.append('')

png = outputPrefix+_files/figure_5.png'
addPlotEntry(fileName=png,
    x=swa*180/math.pi,
    y=swt,
    title='Steering Wheel Torque vs Steering Wheel Angle',
    xlabel='Steering Wheel Angle [deg]',
    ylabel='Steering Wheel Torque [N-m]',
    type = 'plot')

pictures.append(png)
captions.append('')

# creating report
```

```
createHtmlReport(outputPrefix, reportTemplate, title= title, summary=summary, header=header, da
if os.path.isfile (outputPrefix+'.html'):
    return True
else:
    return False
```

A set of utility functions for custom python script are shipped with [reportUtils library](#).

reportUtils Library

reportUtils compiled library is shipped with VI-CarRealTime python package and defines the following functions:

- [getResultSetName](#)
- [getResultSetData](#)
- [addPlotEntry](#)
- [addDataEntry](#)
- [createHtmlReport](#)

Note: to include the library in the python script, just issue the command from `vigrade.vipostprocessing.reportUtils import *`

```
def getResultName(channelName, outputChannelMapFile):
```

The function returns a string with the output channel name in the form <result_set>. <component>.

The function inputs are:

- *channelName*
unique Name of the channel defined in the [Output Channel](#) table.
- *outputChannelMapFile*
xml file which defines the output map.

```
def getResultData(analyses, channelName, outputChannelMapFile):
```

The function returns the [data] values of an output channel.

The function inputs are:

- *analyses*
object variable defining the current analysis to postprocess.
- *channelName*
unique Name of the channel defined in the [Output Channel](#) table.
- *outputChannelMapFile*
xml file which defines the output map.

```
def addPlotEntry(fileName, x, y, color=None, title=None, xlabel=None, ylabel=None, type = 'plot')
```

The function creates a plot and save it in a figure.

The function inputs are:

- *fileName*
string value defining the name of the figure file to create.
- *x*
data values defining the independent axis of the plot.
- *y*
data values defining the dependent axis of the plot.
- *color*
data values defining the color scale of the (x,y) curve. It must be used only when *type='multicolor'* .

- *title*
string defining the title of the plot.
- *xlabel*
string defining the independent axis name of the plot.
- *ylabel*
string defining the dependent axis name of the plot.
- *type*
string defining the type of the plot. Available choices are:
 - 'plot': standard plot of x,y data
 - 'scatter' : scatter plot of x,y data
 - 'multicolor' : plot (x,y) curve is colored with the color scale defined by *color* input parameter.

```
def addDataEntry(data, values):
```

The function appends values to the input data object and returns it.

The function inputs are:

- *data*
data object list.
- *values*
value data to be added to data object list.

```
def createHtmlReport(outputPrefix, reportTemplate, title=None, summary=None, header=None, data=
```

The function creates an html report.

The function inputs are:

- *outputPrefix*
name of the report file that will be created.
- *reportTemplate*
string referencing the html file that will be used as template for report generation. An example is shipped with carrealtime_shared database:
`<carrealtime_shared>/postprocessing_scripts.tbl/custom_ppt_report_template.html`
- *title*
main title shown in the header of the html report.
- *summary*
string defining a summary that will be appended below the title in the html report.
- *header*
header string names of the table columns.
- *data*
object data list with the values that must be written in the report table.
- *pictures*
list of the pictures to add in the html report.
- *captions*
list of the captions to add in the html report.

1.2.6 Investigation Mode

The main window is organized in the following elements:

- **Fingerprints Bookshelf**

shows the list of the available fingerprints; it is used to add a specific fingerprint or an event instance to an [Investigation](#). The fingerprint bookshelf is filled up with the fingerprint defined in [Test Mode](#), so it's

VI-CarRealTime

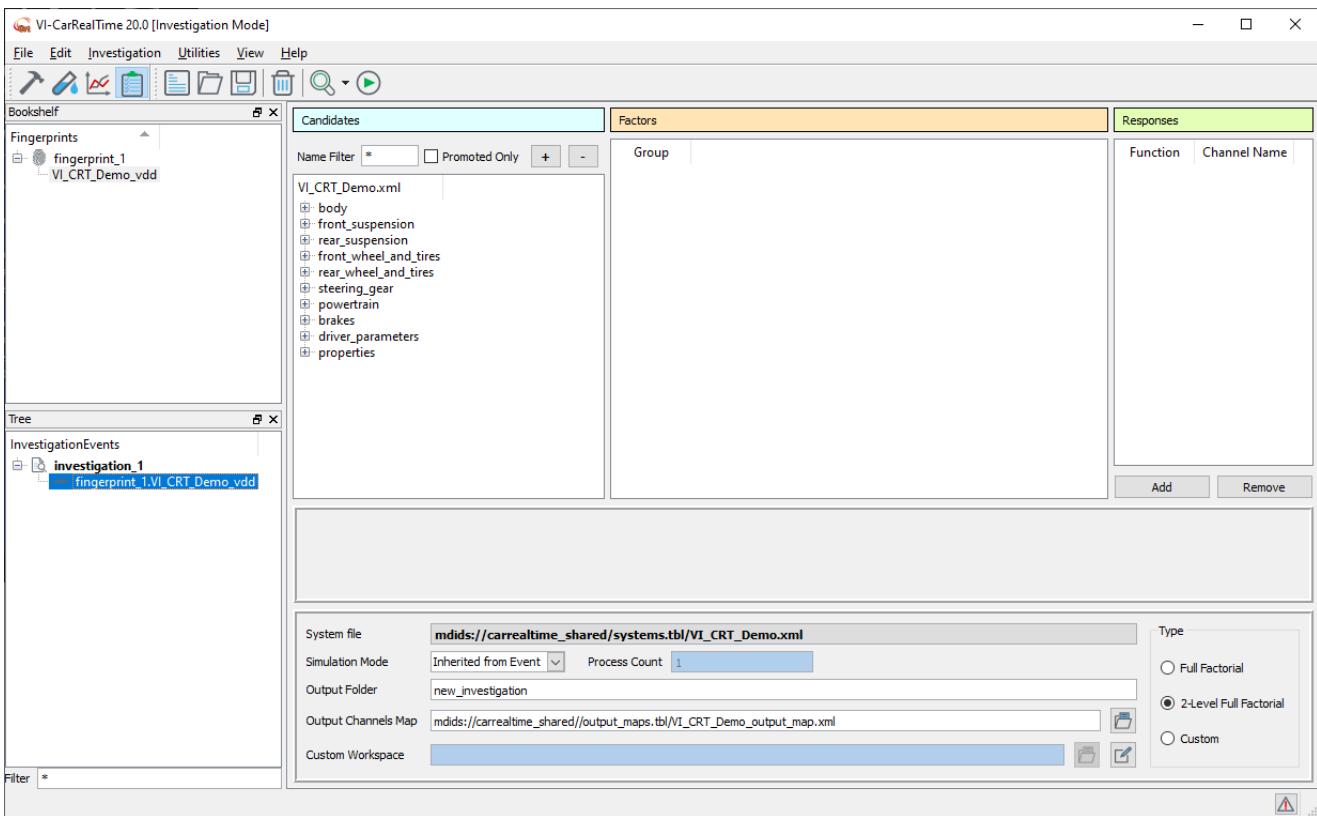
necessary that at least a fingerprint with at least an event has been created in [Test Mode](#) in order to create an [investigation](#).

- **Investigation Tree**

shows the tree of [investigations](#) present in session; click on a single [investigation](#) instance opens the specific Property Editor.

- **Property Editor**

sets the parameters for the [investigation](#) to be run.



The Property Editor of the investigation is composed by the following sections:

- [Candidates](#)
- [Factors](#)
- [Responses](#)
- [Global Settings](#)

While setting up the factors and responses for the investigation and once the investigation has been run, it's possible to visualize an [Investigation Summary Report](#) by pressing button.

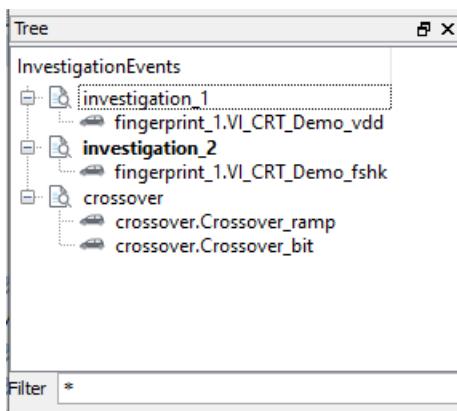
For all the details about the run process, refer to [Run Investigation](#) topic.

Investigations

Investigations are used in VI-CarRealTime to group a set of `fingerprint.event` instances.

Event instances are shown together in **InvestigationEvents** treeview and can be saved to external file.

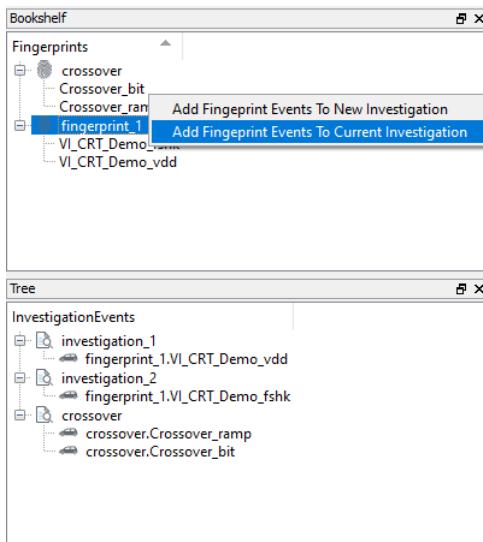
Creating, saving, copying and deleting investigations can be done from the "Investigation" menu or the "Investigation Mode" Toolbar. In addition many features are available in the treeview context menu.



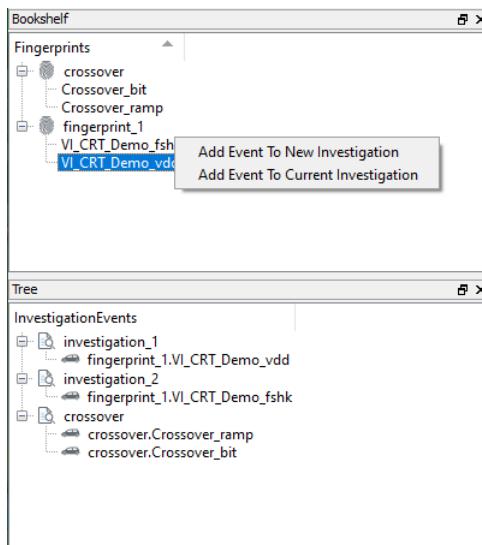
All events in a fingerprint or only a single event can be added to an investigation instance.

In particular:

1. right click on a fingerprint in the Fingerprints bookshelf to add all fingerprint events to a new (or to the current) investigation

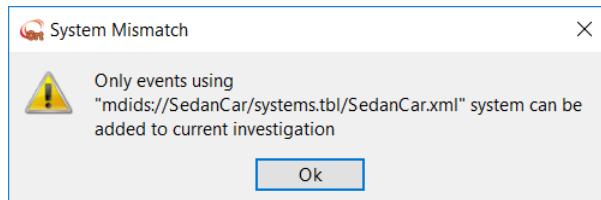


2. right click on the single event of a fingerprint in the Fingerprints bookshelf to add only the event to a new (or to the current) investigation



The investigation instance that has been generated contains the event (or the events) stored as `<fingerprint>.<event>`.

Note: Each Investigation Event refers to a single model (system). Trying to add events with a different system, the following message will be shown:



Right-clicking on the investigation instance, or selecting the Investigation menu from the main toolbar menu, the following options are available:

- **Load Investigation**
load an existing investigation xml file (the option is available even when there are no investigation instances in the session).
- **New Investigation**
create a new investigation instance in the session (the option is available even when there are no investigation instances in the session).
- **Run Selected Investigations**
run the selected investigations.
- **Update Candidates**
the current system in Build Mode is parsed and its parameters are compared with the actual investigation candidates. The procedure merges/adds/deletes the candidate parameters (useful in case the system has been modified, for example adding or removing auxiliary subsystems). At the end of the procedure a HTML report file is automatically generated in VI-CarRealTime working directory and a message windows showing the status pops up.
- **Import Settings From**
allows to import factors and response settings from another investigation instance defined in the session (this option is only available if a single Investigation Event is selected).
- **Export Workspace**

allows to export the workspace (no responses) of the selected investigation to a csv file. This file can be then modified and used as input for a "Custom" investigation (this option is only available if a single Investigation Event is selected).

- **Apply Variant**

allows to apply all the setup parameters of a selected variant to the current model opened in Build Mode. Useful when the user wants to simulate an event using the model set up of a specific variant of an investigation.

- **Generate Responses**

the option can be used with an investigation that has already been run. It computes the values of responses that have been added in the Responses panel after the investigation based on the results of the investigation itself. The [HTML report](#) and the CSV report in VI-CarRealTime working directory are automatically updated with the new responses.

Note: no warning message is thrown for ill-formed responses. An ill-formed response will be skipped.

- **Rename**

rename the investigation instance (it can be done also pressing F2 button after selecting the investigation instance to rename).

- **Delete Selected Events/Investigations**

delete the investigation from the session (if the investigation xml file has been saved, the delete action doesn't remove the file from the working directory).

- **Save Selected Investigations**

save the investigation instance in an xml file in the working directory. The xml file name will be the same as the investigation name instance in the session.

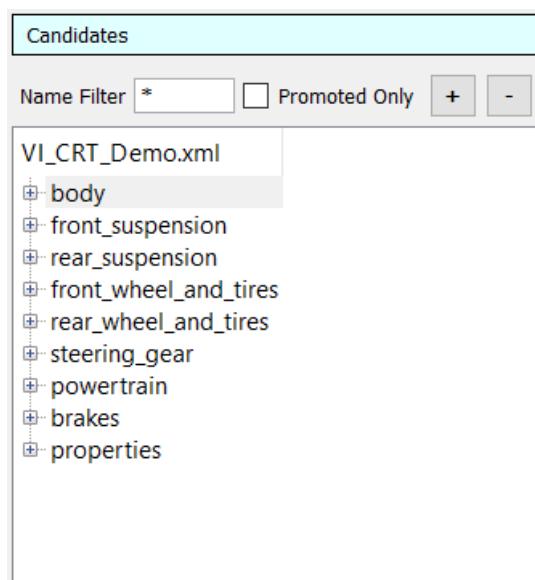
- **Save Investigation As**

save the investigation instance in an xml file in the working directory with the name specified by the user.

Candidates

The Candidates section in the property editor contains a tree view of all the model parameters that can be used (promoted) as factors for the investigation.

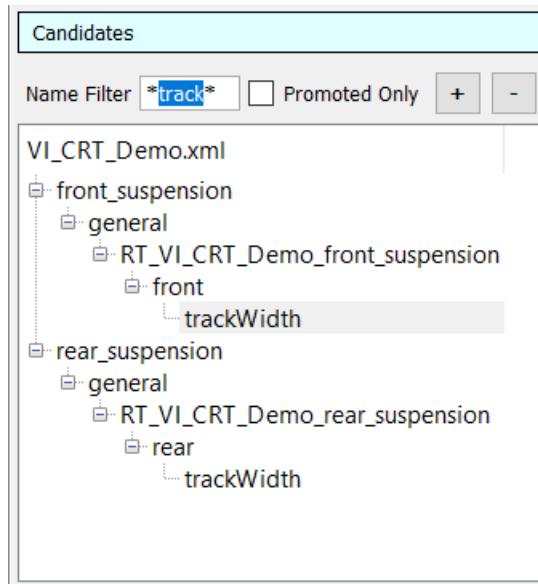
The view contains the system model that will be used for the investigation and all the candidates parameters are stored in the related subsystem block.



The user has the possibility to expand the tree for each subsystem in order to retrieve the parameter (or parameters) he want to promote as factors for the investigation.

To expand one subsystem it's sufficient to press the + button next to the subsystem. By using buttons it's possible to completely expand/collapse the whole tree.

Otherwise, it's possible to rely on Name Filter field to retrieve a candidate by name.

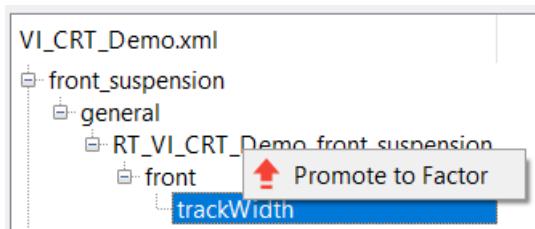


Promote a Candidate

Once the candidate has been retrieved, it must be promoted to a factor in order for such parameter to be used in the investigation.

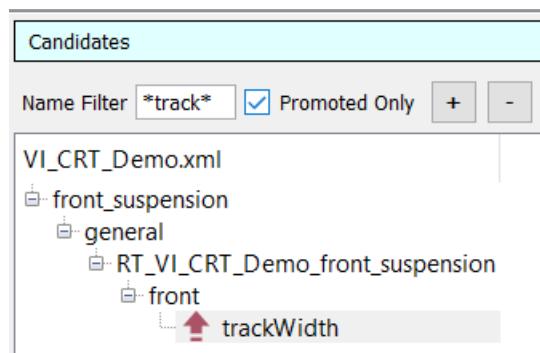
To promote a candidate:

- double click on it
- or
- right click on it and select **Promote to Factor**



In both cases, the selected parameter will become a factor and it's put in the Factors section of the editor.

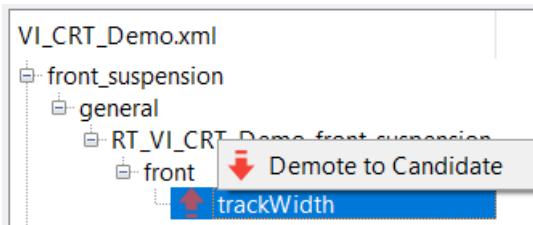
By checking the Promoted Only toggle button, it's possible to visualize in the tree only the candidates that have already been promoted as factors.



Demote a Candidate

To remove a candidate from the factors list (demote):

- double click on it
- or
- right click on it and select **Demote to Candidate**



In both cases, the parameter will be removed from the Factors section.

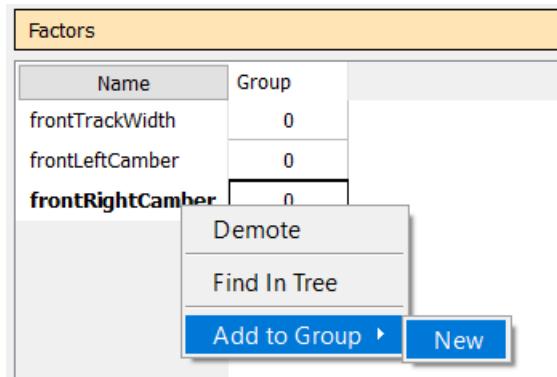
Factors

The Factors section in the property editor stores all the candidates that have been promoted to factors.

Name	Group
front_suspension.general.RT_VI_CRT_Demo_front_suspension.front.trackWidth	0
rear_suspension.general.RT_VI_CRT_Demo_rear_suspension.rear.trackWidth	0

Right click on a factor name in the *Factors* table to:

- Demote selected factor to candidate
- Search and highlight the selected factor in candidates tree
- Add factor to a new or existing group.



For all the details about how to group factors, please refer to [Group](#) section.

Click on a factor in the *Factors* table to display its properties in the bottom part of the editor.

Depending on factor type, either Float or String, a different editor will be shown.

- Real Factor Editor
- String Factor Editor

Each editor consist of the following sections:

- [Factor Information \(common for all types\)](#)
- Factor Settings (different for [Real Factor](#) and [String Factor](#))

Factor Information

The section displays the factor name and its Nominal Value. In addition, for real factors also units are shown if available. The Nominal Value is the value the factor has in [Build Mode](#) and the Units are the model units on the related subsystem.

By default the factor name is very long since it's composed by all the tree nodes separated by dots (full factor name).

The user has the possibility to rename a factor by editing the name in Factor Name field:

Name	Group
trackWidth_front	0

Factor Name: trackWidth_front Nominal Value: 1760.0 Units: mm

The name of the factor will be updated also on *Factors* table. The "Factor Name" will be used in investigation reports.

Real Factor Settings

The Type section allows the user to choose how to vary the factor value during the investigation. Available choices are:

- **Continuous**

the factor is varied between a range that the user can specify by using the following fields:

- **Lower Limit**

lower boundary limit value of the factor. Its value will be interpreted as Absolute, Relative or Relative Percentage depending of the [Delta Type](#) choice.

- **Upper Limit**

upper boundary limit value of the factor. Its value will be interpreted as Absolute, Relative or Relative Percentage depending of the [Delta Type](#) choice.

- **Levels**

it defines the number of values of the factor that will be generated between lower limit and upper limit. The values are equally spaced.

Here an example: the *cgHeight* factor is continuously varied between the absolute lower limit (300 mm) and the absolute upper limit (500 mm) in 5 levels. The value the factor will assume during the investigation are 300, 350, 400, 450, 500 .

Factor Name	body.springmass.RT_VI_CRT_Demo_body.body.cgHeight	Nominal Value	413.404	Units	mm
Type	<input checked="" type="radio"/> Continuous	Delta Type	<input checked="" type="radio"/> Absolute	Lower Limit	300
	<input type="radio"/> Discrete	<input type="radio"/> Relative	<input type="radio"/> Relative Percentage	Upper Limit	500
				Levels	5

- **Discrete**

it's up to the user to enter the values of the factor. The following field is available:

- **Values**

the user must enter the values of the factor (comma separated). The values entered in the field will be interpreted as Absolute, Relative or Relative Percentage depending of the [Delta Type](#) choice.

Here an example: the *cgHeight* factor is discretely varied with respect to its Nominal Value (413.404 mm) of -100,-70,0,40,60,100 . The value the factor will assume during the investigation are respectively: 313.404, 343.404, 413.404, 453.404, 473.404, 413.404 .

Factor Name: body.springmass.RT_VI_CRT_Demo_body.body.cgHeight

Nominal Value: 413.404 Units: mm

Type: Continuous Discrete

Delta Type: Absolute Relative Relative Percentage

Values: -100,-70,0,40,60,100

The Delta Type of the factor can be:

- **Absolute**

the values specified in Lower Limit and Upper Limit fields are absolute values of the factor.

- **Relative**

the values specified in Lower Limit and Upper Limit fields are considered as variations of the factor with respect to its Nominal Value.

- **Relative Percentage**

the values specified in Lower Limit and Upper Limit fields are considered as percentage variations from the Nominal Value of the factor (1.0 means 1%, and the actual value will 1% bigger than the Nominal Value).

Factor Name: front_suspension.general.RT_VI_CRT_Demo_front_suspension.front.trackWidth

Nominal Value: 1760.0 Units: mm

Type: Continuous Discrete

Delta Type: Absolute Relative Relative Percentage

Lower Limit: -1.0

Upper Limit: 1.0

Levels: 2

String Factor Settings

The Type section is deactivated since the only possibility for string factors is "Discrete".

Factor Name: body.aerodynamics.SedanCar_body.aero_forces.propertyFile

Nominal Value: midids://SedanCar/aero_forces.tbl/SedanCar_aero.aer

Type: Continuous Discrete

Values: midids://SedanCar/aero_forces.tbl/SedanCar_aero.aer

- **Values**

it is the only available field for string type factors. It's up to the user to enter a list of comma separated values. In case the factor represents a property file name, the user can select the files to be added to the list (on by one), using the Open File button placed next to the field. The file type is automatically detected.

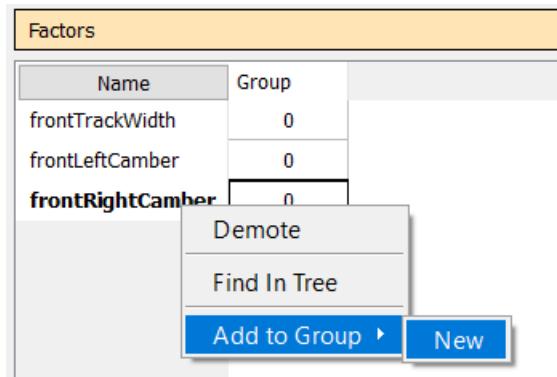
Groups

It is possible to connect two or more factors together by using groups. Grouped factors will vary together (in terms of combinations they behave like a single factor).

To associate a factor to a group:

1. right click on the factor name field;

2. select **Add to Group**;
3. click **New**.



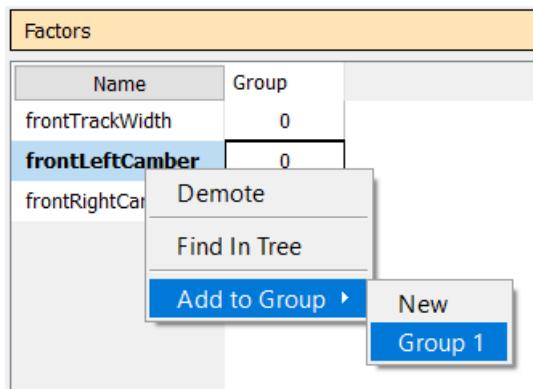
A new group (*Group 1*) will be created and the factor will be associated to the group.

Factors	
Name	Group
frontTrackWidth	0
frontLeftCamber	0
frontRightCamber	1

It's now possible to connect, for example, *frontLeftCamber* factor with *frontRightCamber* factor by putting them in the same group.

To do it:

1. right click on *frontRightCamber* field;
2. select **Add to Group**;
3. click **Group 1**.

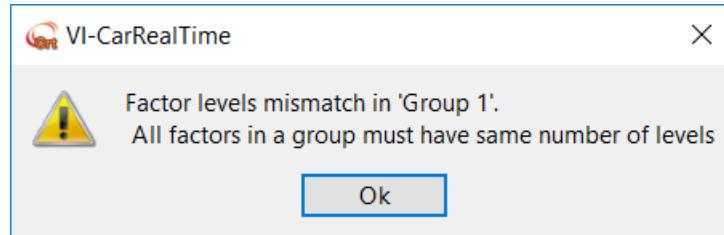


The two factors are now connected since they belong to the same group.

Factors		
Name	Group	
frontTrackWidth	0	
frontLeftCamber	1	
frontRightCamber	1	

It means that the values of the two factors will vary together.

Note: when two or more factors are in the same group it is mandatory that they define the same number of [Levels](#) or the same number of [Values](#) when a [Full-Factorial](#) investigation is performed. If such condition is not verified, the following message is thrown when trying to [Run the Investigation](#) or to create the [Summary Report](#):



Note: no limitation exists when the investigation type is set to [2-Level Full Factorial](#), since in such case only the boundary values (2) of each factor are used for the investigation.

To remove a factor from a group:

1. right click on the factor field;
2. click **Remove from Group**.

Factors		
Name	Group	
frontTrackWidth	0	
frontRightCamber	1	
frontLeftCamber	1	

- Demote
- Find In Tree
- Remove from Group**

Note: use **Find In Tree** option to retrieve the selected factor in the candidates tree view. The tree will be expanded and the candidate will be highlighted.

Responses

The Responses section in the property editor allows to define the metrics to be monitored on each run. The view stores all the responses that the user adds to the investigation. To add a response, click **Add** button in the bottom part of the section.

Responses		
Name	Function	Channel Name
response_01	Maximum Value	
Add		Remove

Once at least a response has been created, click on the response in the Responses table to display its properties in the bottom part of the editor.

The screenshot shows the 'Response Properties' dialog box. At the top, there is a text input field labeled 'Response Name' containing 'response_01'. Below it, there are several configuration options:

- Function:** A dropdown menu set to 'Maximum Value' with a checked checkbox 'Use Absolute Value' next to it.
- Output Channel Name:** An input field with a 'Select Output Channel' button to its right.
- Optional Function Parameters:** A large blue input field.
- Custom Python Script:** A large blue input field with a 'Select File' button to its right.

The editor of a response is composed by the following parameters:

- **Response Name**

field to select the name of the response object that has been created. By default each response is created with a progressive number, but its name can be changed in order to be compliant with the metric to monitor.

- **Function**

option menu to select the function to apply to the selected output channel. There is a number of predefined function, as well as the possibility to specify a Custom Computed Value.

- **Use Absolute Value**

when the flag is active, the computed response is the absolute value of the Function.

Note: the flag is inactive when Function option menu is set to Custom Computed Value.

- **Output Channel Name**

select the output channel to monitor during the investigation. **Select Output Channel** button allows to easily select the channel from the all outputs available. When function is set to one of the predefined response types, only one channel will be used, so in case of multiple channels selected in the Selector Dialog, only the first one will be used. When Function is set to Custom Computed Value, more than one output channel is allowed; please refer to [Using Custom Python Script](#) topic for all the details.

- **Optional Function Parameters**

the field is active only when Function is set to Custom Computed Value; it allows to define additional parameters to be passed to the Custom Python Script.

- **Custom Python Script**

the field is active only when Function is set to Custom Computed Value; it sets the python script that will be used to compute the response. Please refer to [Using Custom Python Script](#) topic for all the details.

Using Custom Python Script

For each response that is created for the investigation, the user must define a metric to be applied to the monitored channel.

In particular, the following built-in functions are available:

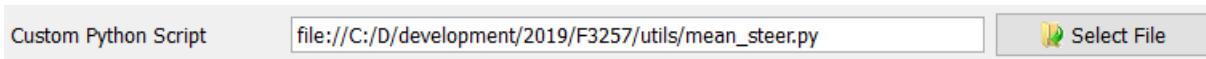
- Average Value
- Last Value
- Maximum Value
- Minimum Value
- Root Mean Square
- Standard Deviation

Whenever the user needs to create a metric response which is not included in the list above, it's possible to rely on **Custom Computed Value**.

To generate the custom computed value a python script can be used.

Note: The investigation procedure will read the python script and will try to call a function whose name is equal to the script file, so it's mandatory that the script defines a function named as the file itself, otherwise a *Traceback* will be thrown and the response won't be computed.

The python script file must be referenced with its full path:



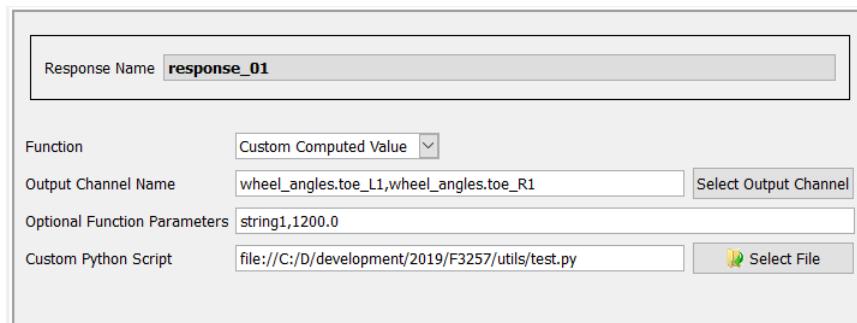
Note: when the investigation is submitted, the procedure will automatically add the python script path to the python system path, so that the python script function can be called. It means that the python script can be located in any folder of the pc.

The procedure to follow is:

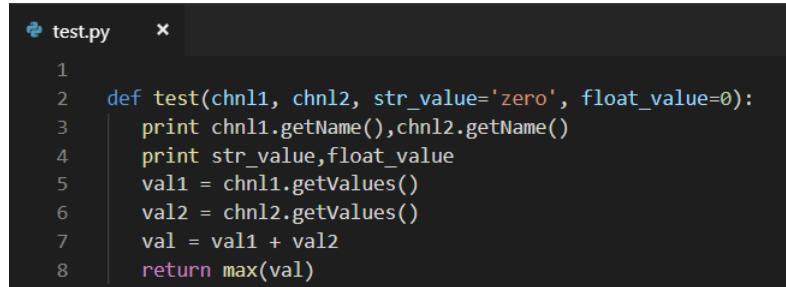
1. select, in the *Output Channel Name* field, all the results that are needed in the python script;
2. add in the *Optional Function Parameters* field, if required, additional parameters (string, integer, real) to be passed in input to the python script;
3. reference the python script file the investigation must use to compute the response.

Here an example which explains how to pass values to a python script:

- response_01 is created;
- Function is set to Custom Computed Value;
- two output channels have been selected (*wheel_angles.toe_L1, wheel_angles.toe_R*)
- two optional function parameters have been added: *string1* (a string) and *1200* (real number)



- a total of four inputs (2 output channels, 1 string and 1 real number) have been inserted. The order of the parameters follows the order with which the parameters have been entered in the related fields (*Output Channel Name* values precede *Optional Function Parameters* values).
- the custom python script (*test.py*) will define a function named *test* (as the file itself) and such function must have 4 parameters:
 - **chnl1** = *wheel_angles.toe_L1*
 - **chnl2** = *wheel_angles.toe_R1*
 - **str_value** = *string1*
 - **float_value** = *1200.0*



```

 1
 2 def test(chnl1, chnl2, str_value='zero', float_value=0):
 3     print chnl1.getName(),chnl2.getName()
 4     print str_value,float_value
 5     val1 = chnl1.getValues()
 6     val2 = chnl2.getValues()
 7     val = val1 + val2
 8     return max(val)

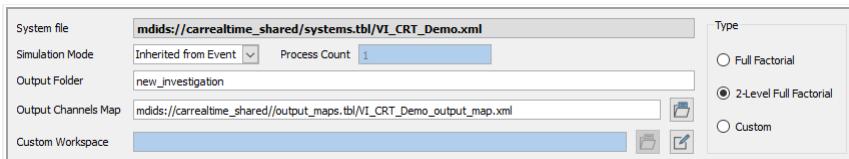
```

Note: when the investigation is submitted, the *test* function will print in the Python shell the names of the channels (*chnl1* corresponding to *wheel_angles.toe_L1* and *chnl2* corresponding to *wheel_angles.toe_R1*), the string *str_value* (whose value is *string1*) and the real number *float_value* (whose value is *1200.0*). Moreover, the script will return the response value, computed as the maximum of the sum between the values of *chnl1* and *chnl2*.

Note: an example of Custom Python Script is shipped with carrealtime_shared database (*mdids://carrealtime_shared.cdb/postprocessing_scripts.tbl/mean_understeer_gradient_response.py*)

Global Settings

The section contains information and parameters that are proper of the investigation; the section is unique for all the factors that have been defined for the current investigation.



The following parameters are defined:

- **System file**
the field displays the system model that has been chosen for the investigation
- **Simulation Mode**
It specifies the simulation mode with which the events of the investigation must be run. Available choices are:
 - *Inherited from Event* : it uses the [simulation mode](#) defined for the event in [Test Mode](#).
 - *interactive* : it forces the event to be run in interactive mode, overwriting the [simulation mode](#) defined in [Test Mode](#) for the event.
 - *files_only* : it forces the event to be run in files_only mode, overwriting the [simulation mode](#) defined in [Test Mode](#) for the event.
- **Output Folder**

the field defines the name of the output folder used for the investigation. To avoid to pollute VI-CarRealTime working directory with the files generated by the investigation procedure, all the files related to the investigation are stored in such specific folder.

- **Output Channel Map**

specifies the output map xml file to be used by VI-CarRealTime solver when running the investigation analyses. The value overrides the output map xml defined in Build Mode.

- **Custom Workspace**

when Type radio button is set to *Custom*, the user has the possibility to set a [CSV file](#) for the investigation.

The Type radio button controls how the factors are combined in the investigation. Available choices are:

- **Full-Factorial**

all the factor [Levels](#) (or [Values](#)) are used for the investigation. The number of analysis variants run are equal to the product of the [Levels](#) (or [Values](#)) of all the factors defined for the current investigation (each set of grouped factor counts as one factor only in the generation of the combinations).

- **2-Level Full-Factorial**

only the limits ([Lower Limit](#) and [Upper Limit](#) or the first and last value of the [Values](#) field) are actually used for the investigation. The number of analysis variants run are equal to the product of the number of factors defined for the current investigation multiplied by 2. (each set of grouped factor counts as one factor only in the generation of the combinations)

- **Custom**

the variant analyses to be run in the investigation are defined in the [CSV file](#) referenced in *Custom Workspace* field. When this option is selected factors defined in the investigation are neglected, and the model variants to be executed are fully determined by the content of the CSV file as explained in the dedicated section. Responses instead must be explicitly defined in the investigation.

Press  button to browse for a CSV file.

Press  button to open the CSV file in the default program used to manage such files.

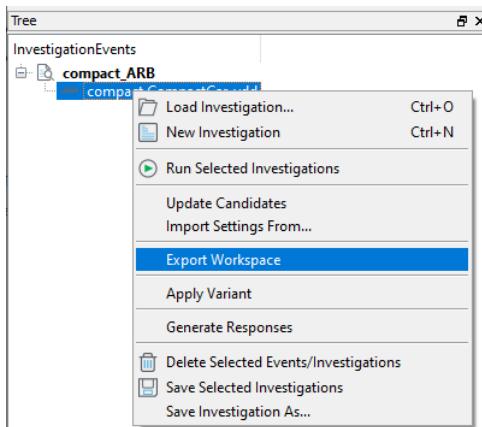
Custom Workspace (CSV file)

In order to cover all the factors combination matrix, beside 2 Level Full-Factorial and Full-Factorial modes, the investigation mode gives the user the possibility to customize his own workspace.

To enable the custom workspace factor combination:

1. Type ratio button must be set to **Custom**;
2. a valid CSV input file must be select in **Custom Workspace** field.

The CSV file can be exported from an investigation; in particular, once the factors have been defined as explained in Factors section and the Type has been set either to [2 Level Full-Factorial](#) or to [Full-Factorial](#), to export the current factors combination (workspace) it's sufficient to right-click on the investigation instance in the Tree view and select **Export Workspace**.



A CSV file will be created in VI-CarRealTime working directory. The name of the file is composed by [Output Folder](#) name plus the suffix `_workspace`.

Note: to export the CSV, *Type ratio* button can't be set to *Custom*. A warning message is thrown when trying to export a workspace using a custom investigation strategy.

The CSV file contains all the factor combinations (rows) as they have been defined in the investigation GUI and it is formatted as follows:

- first column defines the `activeFlag` for each combination: by default all combinations are active (1); to switch off a combination just set the related `activeFlag` to 0;
- one column for each factor that has been defined (the first row reports the factor name and the second row reports the [full factor name](#));

	A	B	C	D
1	activeFlag	frontARB_scaleFactor	rearARB_scaleFactor	
2	activeFlag	front_suspension.antiroll.CompactCar_front_suspension.front.scaleFactor	rear_suspension.antiroll.CompactCar_rear_suspension.rear.scaleFactor	
3	1	0	0	0
4	1	0	0.5	0.5
5	1	0	1	1
6	1	0	1.5	1.5
7	1	0	2	2
8	1	0.5	0	0
9	1	0.5	0.5	0.5
10	1	0.5	1	1
11	1	0.5	1.5	1.5
12	1	0.5	2	2
13	1	1	0	0
14	1	1	0.5	0.5
15	1	1	1	1
16	1	1	1.5	1.5
17	1	1	2	2
18	1	1.5	0	0
19	1	1.5	0.5	0.5

Note: it's possible to directly add one or more factors in the CSV file. To do it, just add one or more columns formatted as the other ones.

Note: it's mandatory that the factor name defined in the second row of the CSV file exactly coincides with the [full factor name](#), otherwise a warning message will be thrown and the related factor will be skipped.

Run Investigation

Once the investigation parameters have been set-up, to run the investigation click on button, or right click on the investigation instance and select [Run Selected Investigation](#).

When the investigation is launched:

1. the investigation folder ([Output Folder](#)) is created in VI-CarRealTime working directory and all the investigation files are generated in such folder.
2. the system model used for the investigation is published in a new database in the investigation directory. The database is named <investigation_name>.cdb . The investigation will use such model database when running the variant analyses in order to do not pollute the original system model.
3. for each investigation variant to run, the procedure generates all the solver files needed for a standard VI-CarRealTime analysis plus a VRP file in which the current value of each factor is written. VI-CarRealTime solver is called to run each simulation.
4. A folder named custom_ppt is created in the investigation folder. The folder contains a python script which is generated on the fly by the investigation procedure. It contains a function (*postProcessing*) which evaluates all the [responses](#) defined for the current investigation (this folder is not created in case no response is defined).

Finally, when all the investigation variant analyses have been run, two report files are generated in VI-CarRealTime working directory:

- [<investigation_name>.summary.html](#)
- [<investigation_name>_summary_<investigation_event_name>.csv](#)
it's the equivalent version, in comma-separated values form, of the HTML report. For investigation with multiple events, multiple csv files are generated.

Run Investigation in batch mode

It is possible to run an investigation in batch mode by issuing this command from a dos shell:

vig20 acreal ru-st b <investigation_file_name> <vicrtsession_file_name>

where:

- <investigation_file_name> is a valid name for an existing investigation xml file.
- <vicrtsession_file_name> (OPTIONAL) is the xml session file of VI-CarRealTime (default is *vicrt_20_defaultSession.xml*).

<vicrtsession_file_name> is an optional parameter; if such parameter is passed as second input, it is used as session file.

If no session file is passed in input, the script looks for the session xml file in VI-CarRealTime working directory and if it's found, it is used as session file.

In case no session file is retrieved (neither as second input parameter, nor from working directory), the script looks for the fingerprint files referenced inside the investigation xml file (<investigation_file_name>) and expects to found such fingerprint xml files in VI-CarRealTime working directory.

Note: Executing the simulations of an investigation using this approach is equivalent to run them from the VI-CarRealTime GUI. The referenced models are loaded from the database, simulation files are created and executed in batch mode.

Summary Report

A summary report is available for the user in order to check the investigation results, but also to check at design time how the investigation has been configured.

The report is in HTML format and it's automatically created based on the current investigation status.

To open the report, click on the stack button  in the Toolbar, or click on the *Investigation* menu and select **View Design Space**.

The stack button defines two different buttons:

- **View Design Space**

the button generates on the fly an HTML based on the current investigation configuration and loads the report in the default browser.

- **View Design Space and Results**

the button generates on the fly an HTML based on the current investigation configuration and attempts to retrieve the results from the investigation folder. The report maybe incomplete if the investigation has not been run, or if the settings have been modified after execution (the html file generated at the end of the investigations is not affected).

Here an example of investigation summary report generated *before* submitting the investigation:

General Info

Investigation Name	investigation_1
System File	mdids://SedanCar/systems.tbl/SedanCar.xml
Strategy	2-Level Full Factorial
Output Folder	lateral_investigation

Event

Selected Event: fingerprint_1.SedanCar_step

Variants

Identifier	front_suspension.general. SedanCar_front_suspension.front.trackWidth	front_suspension.springs. SedanCar_front_suspension.front.left.scaleFactor	Success	MAX_yaw_rate	MAX_lat_acc
v0001	1498.00	0.500000	?	N/A	N/A
v0002	1498.00	1.500000	?	N/A	N/A
v0003	1698.00	0.500000	?	N/A	N/A
v0004	1698.00	1.500000	?	N/A	N/A

Factor 1

Factor 2

Response 1

Response 2

The report is composed by the following sections:

- **General Info**

the table contains general information regarding the investigation to submit. In particular:

1. the name of the investigation instance to run
2. the VI-CarRealTime system model that will be used for the investigation
3. the strategy used to combine the factors
4. the name of the output folder where all the investigation files will be generated

- **Event**

the box reports the event that will be run. If the investigation defines more than one event, it's possible to switch the events by clicking on *Selected Event* button

- **Investigation Table**

the investigation table reports all the variants that will be run for the investigation. In particular the following columns are displayed:

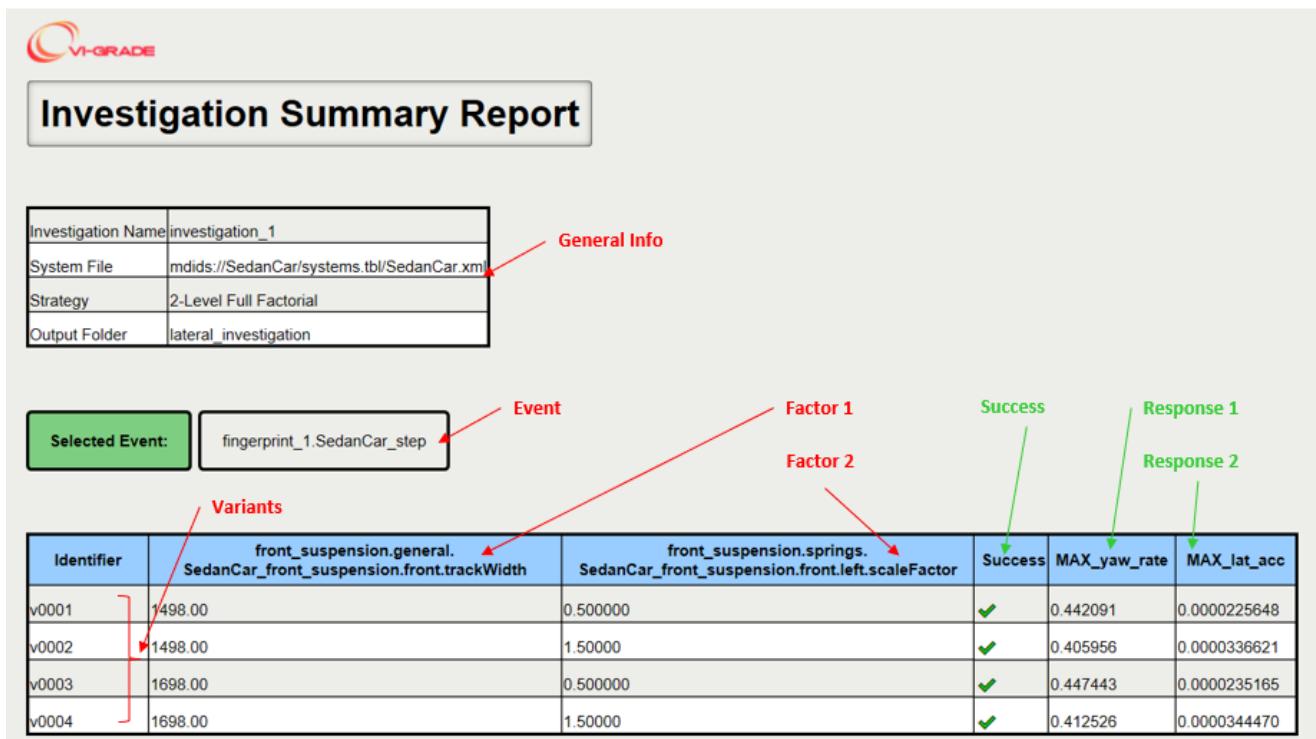
1. *Identifier* which defines an identification number for each variant that will be run
2. N columns for the N factors chosen for the investigation
3. *Success* defining the status of the simulation for each variant (the question mark in the figure means that the related analysis hasn't been run yet)

4. N columns for the N responses defined in the investigation (N/A means that the result is not available - the investigation event has not been run yet)

- **Plot**

the plot section is available only when the investigation has been run.

Here an example of investigation summary report generated *after* submitting the investigation:



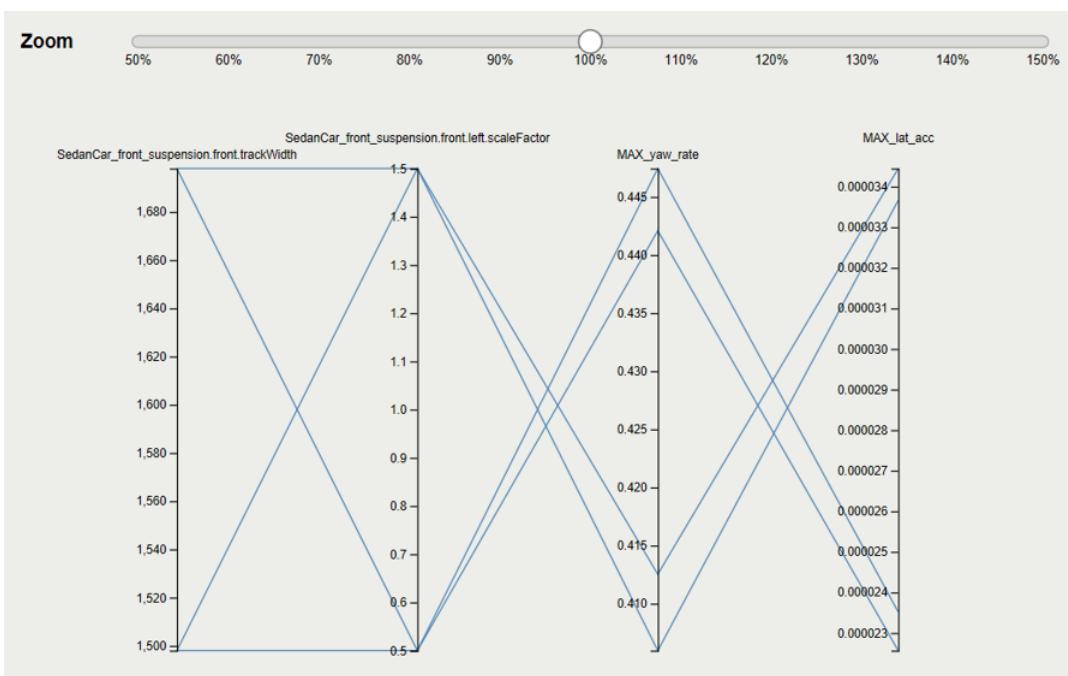
The screenshot shows the 'Investigation Summary Report' interface. At the top left is the VI-GRADE logo. Below it is a title bar with 'Investigation Summary Report'. The main area contains two tables. The first table, labeled 'General Info', lists parameters: Investigation Name (investigation_1), System File (mdids://SedanCar/systems.tbl/SedanCar.xml), Strategy (2-Level Full Factorial), and Output Folder (lateral_investigation). The second table, labeled 'Variants', lists four variants (v0001, v0002, v0003, v0004) with their identifiers and success status. Red arrows point from labels to specific fields: 'General Info' points to the System File field; 'Event' points to the 'Selected Event' field; 'Factor 1' and 'Factor 2' point to the 'Identifier' column of the Variants table; 'Success' points to the 'Success' column of the Variants table; and 'Response 1' and 'Response 2' point to the 'MAX_yaw_rate' and 'MAX_lat_acc' columns of the Variants table respectively.

Investigation Name	investigation_1
System File	mdids://SedanCar/systems.tbl/SedanCar.xml
Strategy	2-Level Full Factorial
Output Folder	lateral_investigation

Identifier	front_suspension.general. SedanCar_front_suspension.front.trackWidth	front_suspension.springs. SedanCar_front_suspension.front.left.scaleFactor	Success	MAX_yaw_rate	MAX_lat_acc
v0001	1498.00	0.500000	✓	0.442091	0.0000225648
v0002	1498.00	1.50000	✓	0.405956	0.0000336621
v0003	1698.00	0.500000	✓	0.447443	0.0000235165
v0004	1698.00	1.50000	✓	0.412526	0.0000344470

Note: the report now shows the investigation results. In particular, the Success column reports the simulation status (✓ means ok, ✗ means error during the variant event simulation, the warning sign means that warnings have been issued during the simulation) and the response columns are filled up with the results.

Furthermore, a *Plot* section showing the combination between [factor](#) values and [response](#) values is available in the bottom part of the report:



1.2.7 Utilities

VI-CarRealTime features some useful support tools that can be accessed using the Utilities menu in application Menu Bar:

- [Database Utilities](#)
- [Tire Testrig](#)
- [Vehicle Wizard](#)
- [K&C Wizard](#)
- [Trailer Wizard](#)
- [Property File Obfuscation](#)
- [VDF Converter](#)
- [CarSim Converter](#)
- [CarMaker Converter](#)

Database Utilities

Database utilities editor offers the following features:

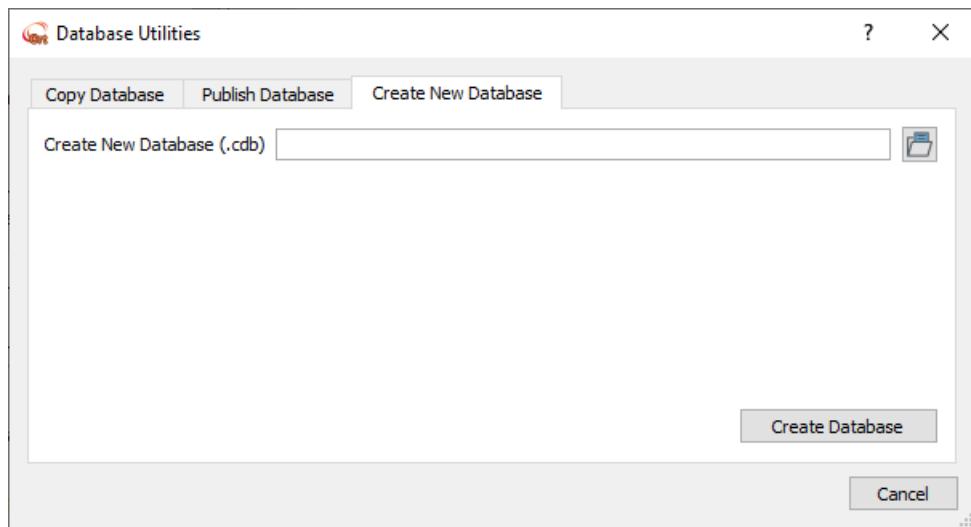
[Create New Database](#)

[Copy Database](#)

[Publish Database](#)

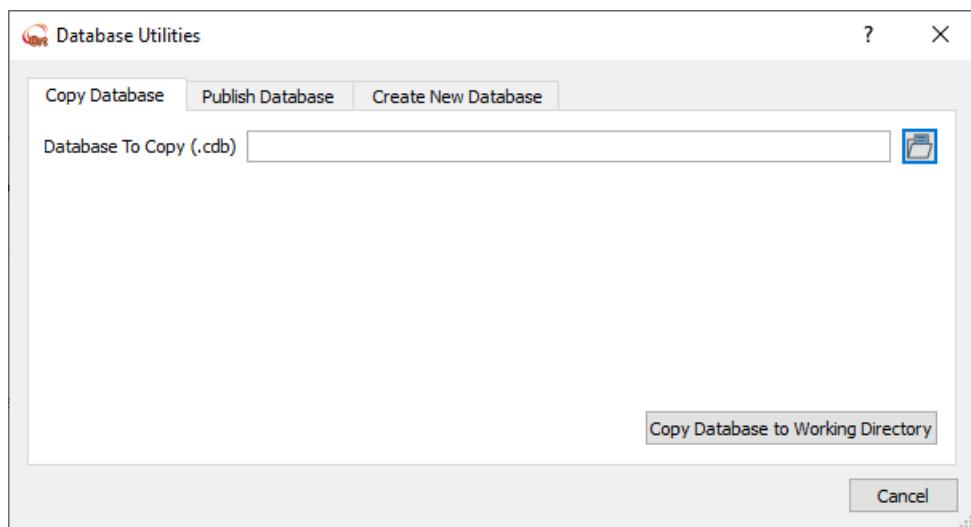
Create New Database

The user specifies the path of a new database (.cdb); the utility creates all the tables needed by VI-CarRealTime.



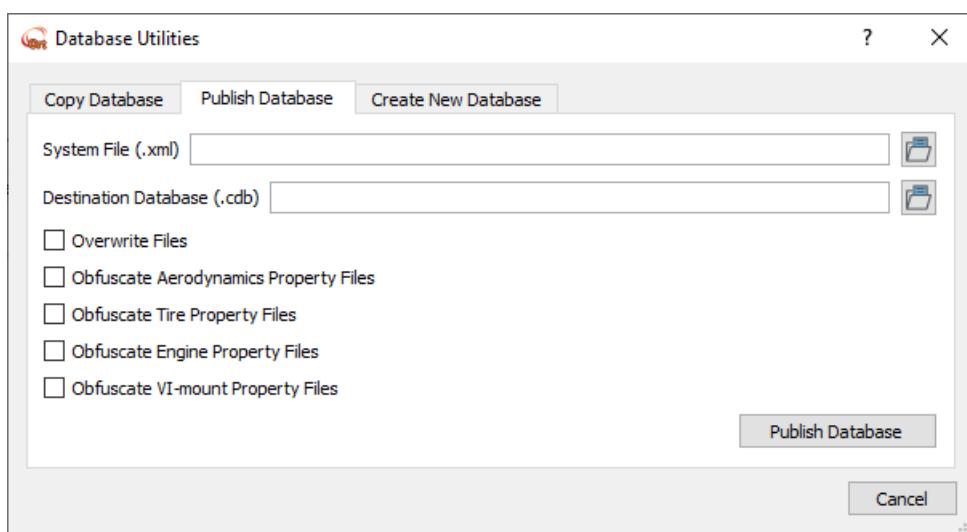
Copy Database

The user can enter the path of an existing database; the utility copies it to working directory:



Publish database

The user enter the path of a system file (.xml); the utility publish the file to a target database and updates all path references.



- **Overwrite Files**

when the toggle button is checked, two or more property files having the same names are overwritten, otherwise a backup copy is made.

- **Obfuscate Aerodynamics Property Files**

when the toggle button is checked the exported model will have the aerodynamic property file obfuscated.

- **Overwrite Tire Property Files**

when the toggle button is checked the exported model will have the tire property files obfuscated.

- **Overwrite Engine Property Files**

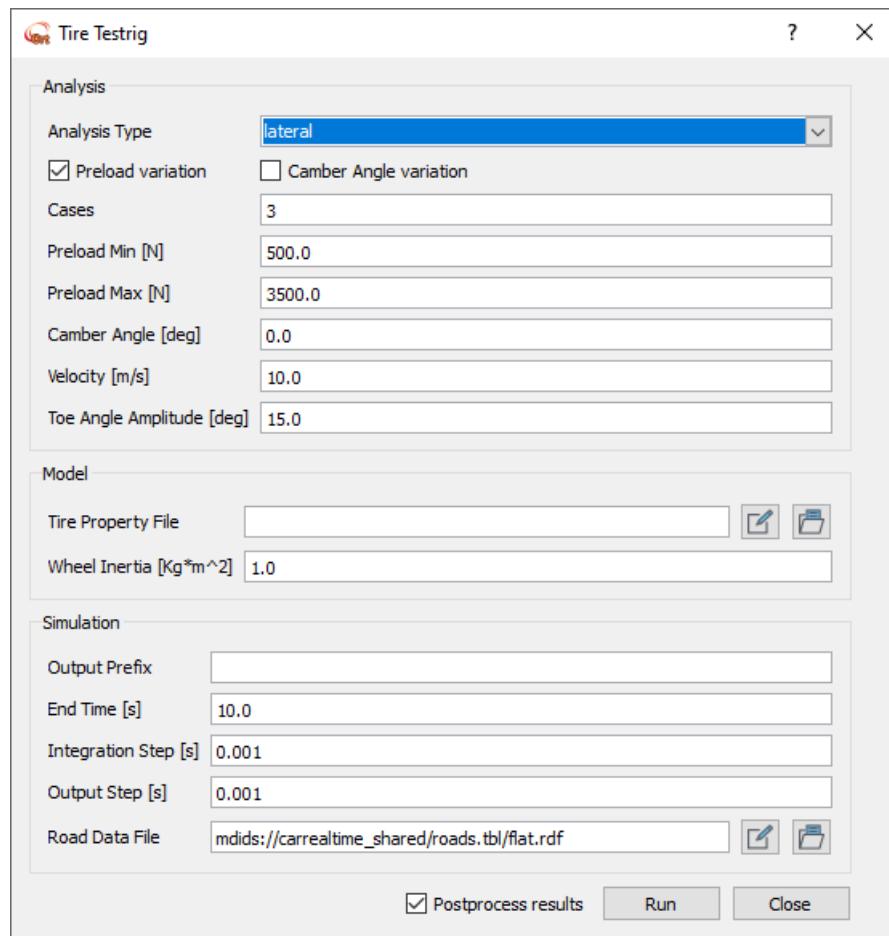
when the toggle button is checked the exported model will have the engine property file obfuscated.

For further info about property file obfuscation please refer to [Property File Obfuscation](#) topic.

Tire Testrig

VI-TireTestrigRealtime Editor permits to compare the tire dynamic behavior (according to data written in the associated property file) with tire manufacturer plot curves.

VI-TireTestrigRealtime is useful to run several types of dynamic analysis in order to get tire characteristic plots.



In order to run VI-TireTestrigRealtime analysis user have to supply the following data:

Analysis

- **Analysis Type**

available options are lateral slip variation (lateral), longitudinal slip variation (longitudinal), combined slip (combined) external file (file).

With respect to analysis choice, a different multibody system is handled:

- **Combined and longitudinal**

Analysis will be performed using a prescribed spindle rotation (1 dof - heb vertical direction)

- **Lateral**

Analysis will be performed using a 2 dof system (wheel rotation and spindle vertical direction)

- **File**

Analysis will be performed using one of 4 dynamic model available:

- 1 dof z free (hub vertical displacement)
- 2 dof (spindle rotation and hub vertical displacement)
- 6 dof (free wheel)
- 1 dof omega free (spindle rotation)

Vtt file description will show how user can switch between different model.

- **Camber Angle**

camber angle value used during analysis.

- **Velocity**

wheel longitudinal velocity.

- **Toe Angle Amplitude**

amplitude of toe angle variation (used in lateral and combined analyses).

- **Longitudinal Slip**
fixed value of longitudinal slip (used in combined analysis).
- **Preload**
vertical wheel preload.
- **Input File**
.vtt input file (used in file analysis, all other analysis parameters are disabled).

Model

- **Tire Property File**
tire property file to be analyzed.
- **Wheel Inertia**
wheel rotational inertia (Kg*m^2).

Simulation

- **Output Prefix**
name prefix for result file.
- **End Time**
duration of simulation.
- **Integration Step**
solver integration step.
- **Output Step**
output file time step.
- **Road Data File**

Postprocess results

Optionally calls the currently selected postprocessing utility (VI-Animator or Adams PostProcessor) and load result file.

Running Tire Testrig Realtime

VI-TireTestrigRealtime is available as standalone version too. Using standalone solver it is possible to access additional feature and different simulation type. Solver will require as input a configuration file (.vtt).

To Run VI-TireTestrigRealtime you must open a shell dos and edit the command line:

```
$VI-CarRealTime\Installationdir\acarrt\win64\ttrt.exe -f input_file.vtt
```

```
Administrator: C:\Windows\system32\cmd.exe
E:\D\working_directory\vicrt\test>"C:\Program Files\VI-grade\VI-CarRealTime 19\carrt\win64\ttrt.exe" -f PAC_lateral_fz_3500.0_camber_0.0.vtt
=====
=          VI-TireTestrig RT          =
=====
VERSION : 19.0
REVISION: 37371_dev//trunk/software
BUILT ON: phoenix/15-Jul-18
=====

Initializing tires...
Loading road data file... <file=E:/svn/trunk/software/INTERM^1/win64/11.0/MoAdam
s/release/VICRT_~1/acarrt/CARREA^1.CDB/roads.tbl/flat.rdf>
road file total reading time = 0.002s
FLAT road model has been initialized...
initializing VI-TIRE Pacejka 96/02 model... OK
Done.

Solving for static equilibrium...
Done.

Performing dynamic simulation...
Done.

Writing output to file. Please Wait ...Done.

Simulation is complete.
```

Vehicle Wizard

Vehicle Wizard is a tool which allows the creation of VI-CarRealTime vehicle models.

The editor features several sections, each of them related to a particular vehicle area; the user can provide as input either a set of parameters to define specific vehicle subsystems or supply existing subsystem files.

Depending on geometric inputs vehicle graphics will also be created.

The vehicle model is stored into one of registered databases and the vehicle configuration (containing the set of parameters used to generate the model) can be saved/reloaded to be the starting point for a new vehicle model.

[General Parameters & Subsystem Files](#)

[Mass and Aerodynamics](#)

[Suspensions and Tires](#)

[Powertrain](#)

[Brakes](#)

[Auxiliary Sensors/Loads](#)

The following actions are available on the left bottom of the editor:

- **Load Configuration**

button loads xml data needed to populate Vehicle Wizard editor. Files are stored in database under vehicle_configs.tbl table.

- **Save Configuration**

button saves to xml data populating Vehicle Wizard editor.

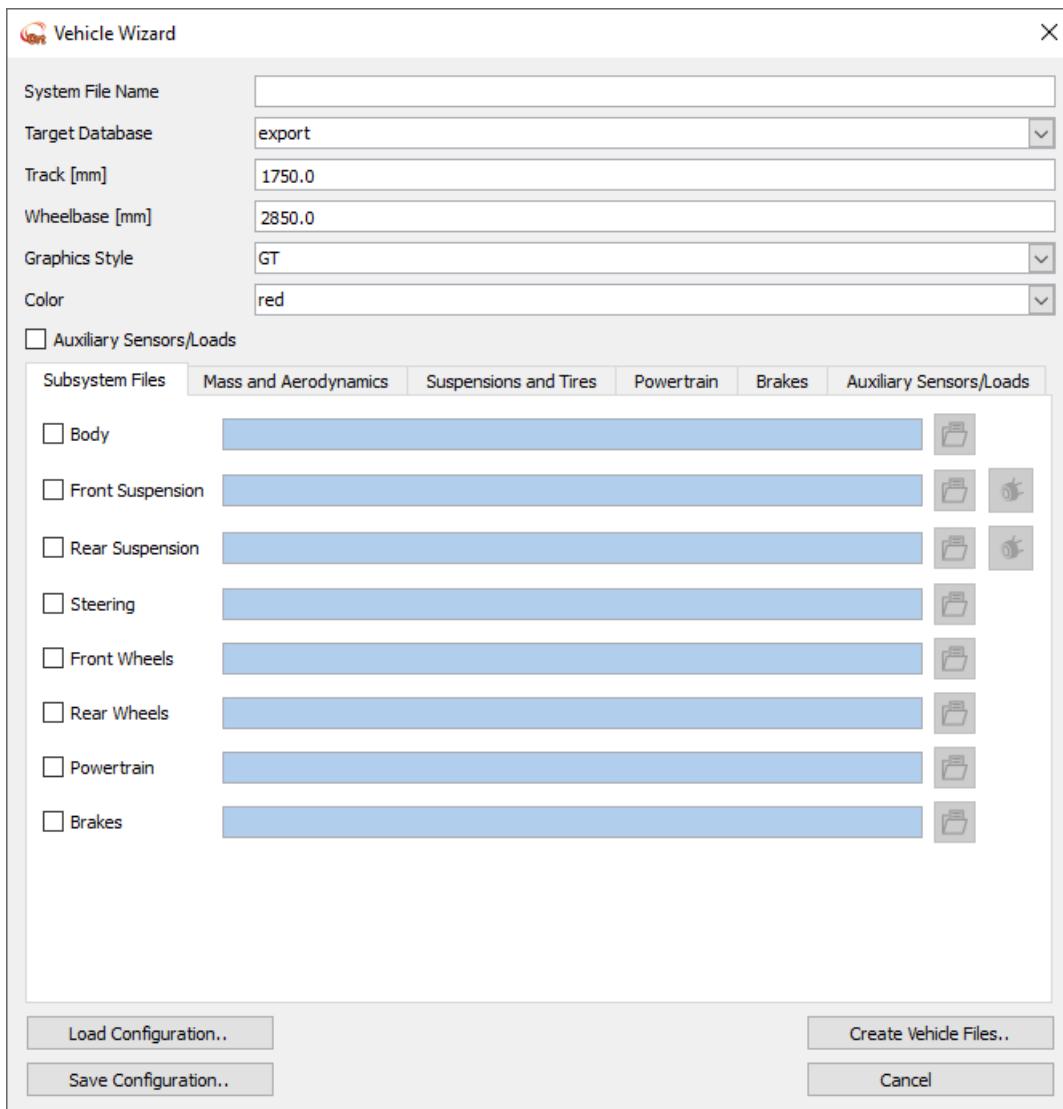
- **Create Vehicle Files**

button starts vehicle data creation process. Popup dialog will show the successful/unsuccessful status of process. Created vehicle models will **not** be automatically loaded into VI-CarRealTime session.

- **Cancel**

button closes the editor.

General Parameters & Subsystem Files



General parameters in Vehicle Wizard are:

- **System File Name**
file name for created vehicle model;
- **Target Database**
registered database used to store the model;
- **Track**
vehicle track;
- **Wheelbase**
vehicle wheelbase;
- **Graphics Style**
type of vehicle (used to generate chassis graphics); available options are: Compact, GT, Sedan, Open Wheels, SUV, Spider;
- **Color**
chassis graphics color;
- **Subsystem File Names**

name of existing subsystem files to be used in vehicle model (activating a specific subsystem will deactivate related parameter options in related editor tabs).

Mass and Aerodynamics

Subsystem Files		Mass and Aerodynamics	Suspensions and Tires	Powertrain	Brakes	Auxiliary Sensors/Loads
<input checked="" type="checkbox"/> Use global mass			Weight Front Bias	0.6		
Vehicle Mass [Kg]	1600.0		Sprung Mass CGZ [mm]	460.0		
Front Wheel Mass [Kg]	30.0		Rear Wheel Mass [Kg]	30.0		
Chassis Mass [Kg]	1000.0					
Fuel Tank Mass [Kg]	50.0					
Powertrain Mass [Kg]	150.0					
	X [mm]	Y [mm]	Z [mm]			
Chassis CG Loc	-1500.0	0.0	420.0			
Fuel Tank CG Loc	-1800.0	0.0	300.0			
Powertrain CG Loc	-2100.0	0.0	450.0			
Vehicle Frontal Area [mm ²]	2000000.0					
Aero Drag Coefficient	0.3					
Aero Front DownForce Coefficient	0.3		Aero Rear DownForce Coefficient	0.3		

Mass and aerodynamics properties are defined through the following data:

- Use global mass**

The toggle defines the way the vehicle mass is computed. Active status enables mass computing through full vehicle, wheel masses and weight bias values; inactive status requires chassis, fuel tank and powertrain parts mass definition;

- Weight Front Bias**

portion of vehicle weight acting on front wheels;

- Vehicle Mass**

total vehicle mass;

- Sprung Mass CGZ**

vertical position of sprung mass CG;

- Front/Rear Wheel Mass**

unsprung masses values;

- Chassis Mass**

chassis part mass;

- Fuel Tank Mass**

fuel tank part mass;

- Powertrain Mass**

powertrain part mass;

- Chassis CG Loc**

chassis part center of gravity location;

- Fuel Tank CG Loc**

fuel tank part center of gravity location;

- Powertrain CG Loc**

powertrain part center of gravity location;

- **Vehicle Frontal Area**
vehicle area needed for aerodynamic force computation;
- **Aero Drag Coefficient**
coefficient for aero drag force;
- **Aero Front/Rear DownForce Coefficient**
coefficient for aero down forces.

Suspensions and Tires

Subsystem Files	Mass and Aerodynamics	Suspensions and Tires	Powertrain	Brakes	Auxiliary Sensors/Loads
Front Spring Stiffness [N/mm]	150.0				
Front Spring Preload [N]	4000.0				
Front ARB Stiffness [N/mm]	25.0				
Front Damper Damping [N sec/mm]	8.0				
Front Tire Property File	mdids://carrealtime_shared/tires.tbl/p96f.tir				
Steering Ratio	15.0				
Rear Spring Stiffness [N/mm]	150.0				
Rear Spring Preload [N]	4000.0				
Rear ARB Stiffness [N/mm]	25.0				
Rear Damper Damping [N sec/mm]	8.0				
Rear Tire Property File	mdids://carrealtime_shared/tires.tbl/p96r.tir				
<input type="checkbox"/> Rear Twin Wheels Lateral Offset	300.0				
Rear Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/p96f.tir				

The parameters needed to define suspension characteristics are:

- **Front/Rear Spring Stiffness**
spring component linear stiffness (symmetric);
- **Front/Rear Spring Preload**
spring component preload;
- **Front/Rear ARB Stiffness**
anti roll bar device ground stiffness;
- **Front/Rear Damper Damping**
damper component damping coefficient;
- **Front/Rear Tire Property File**
property file needed for tire force.

By a flag, the user can activate the rear twin wheel; the parameters needed are:

- **Rear Twin Wheels Lateral Offset**
lateral distance between the CGs of the two wheels; the virtual wheel center location is centered between these two CGs;
- **Rear Twin Tire Property File**
property file for tire forces computation; supported format are: MF_05, PAC2002, PAC_CT, [VI_TIRE](#).

Suspension curves and motion ratios are considered flat; user can eventually create or modify them using VI-SuspensionGen.

Powertrain

Subsystem Files	Mass and Aerodynamics	Suspensions and Tires	Powertrain	Brakes	Auxiliary Sensors/Loads
Maximum Torque [Nm]	500.0				
Maximum Torque Ang. Vel. [rpm]	5500.0				
Idle Ang. Vel. [rpm]	1000.0				
Rev Limit Ang. Vel. [rpm]	8000.0				
Transmission Ratio	3.5				
Front Drive Bias	0.0				
Differential Type	open				
Gear Ratios	Plot/Edit Data				

Powertrain definition is based on some parameters needed to compute engine torque map.

- **Maximum Torque**
maximum torque value produced by the engine;
- **Max Torque Angular Velocity**
engine angular velocity for maximum torque;
- **Idle Ang. Vel.**
engine idle RPM;
- **Rev Limit Ang. Vel.**
engine maximum RPM.

Using the parameters above a predefined shaped engine torque map is created. Further modification can be done in vehicle subsystem editor after loading the model in VI-CarRealTime.

A further set of parameters is needed to fully define the powertrain subsystem:

- **Transmission Ratio**
gearbox transmission ratio;
- **Front Drive Bias**
parameter defining the way drive torque is applied to wheels; FWD=1.0, RWD=0.0, 0.0<AWD<1.0;
- **Differential Type**
type of differential;
- **Gear Ratios**
spline containing ratios for powertrain gears.

Brakes

Subsystem Files	Mass and Aerodynamics	Suspensions and Tires	Powertrain	Brakes	Auxiliary Sensors/Loads
Front Brake Bias	0.6				
Friction Coefficient	0.4				
Front Piston Effective Radius [mm]	180.0				
Front Piston Area [mm ²]	2300.0				
Rear Piston Effective Radius [mm]	180.0				
Rear Piston Area [mm ²]	2300.0				

Brake system parameters are:

- **Front Brake Bias;**
- **Friction Coefficient;**
- **Front/Rear Piston Effective Radius;**
- **Front/Rear Piston Area.**

Auxiliary Sensors/Loads

Subsystem Files	Mass and Aerodynamics	Suspensions and Tires	Powertrain	Brakes	Auxiliary Sensors/Loads
Vehicle Sensors	X [mm]	Y [mm]	Z [mm]	Components	Symmetric
<input checked="" type="checkbox"/> ECU	0.0	0.0	0.0	ACC x,y,z; V x,y,z	No
<input type="checkbox"/> Upfront	0.0	0.0	0.0	ACC x,y,z	Symmetric
<input type="checkbox"/> A-pillar	0.0	0.0	0.0	ACC x,y,z	Symmetric
<input type="checkbox"/> B-pillar	0.0	0.0	0.0	ACC x,y,z	Symmetric
<input type="checkbox"/> C-pillar	0.0	0.0	0.0	ACC x,y,z	Symmetric
<input type="checkbox"/> Auxiliary 1	0.0	0.0	0.0	ACC x,y,z	No
<input type="checkbox"/> Auxiliary 2	0.0	0.0	0.0	ACC x,y,z	No
Vehicle Loads	X [mm]	Y [mm]	Z [mm]	Mass	
<input type="checkbox"/> Roof	0.0	0.0	0.0	0.0	

Auxiliary Sensors/Loads parameters are:

Vehicle Sensors:

- **ECU**
x,y,z coordinates;
- **Upfront**
x,y,z coordinates;
- **A-pillar**
x,y,z coordinates;
- **B-pillar**
x,y,z coordinates;
- **C-pillar**
x,y,z coordinates;
- **Auxiliary 1**
x,y,z coordinates;
- **Auxiliary 2**
x,y,z coordinates;

Vehicle Roof:

- **Roof**
x,y,z coordinates and mass;

VI-Safety toolkit is capable of combining the standard features of VI-CarRealTime model with specific entities and functionalities.

In order to allow a modular approach to simulation and to give to possibility of extending existing models to the use with VI-Safety toolkit, all the specific modeling entities have been confined into an auxiliary subsystem including:

- Sensors
- Roof load properties

The creation of a new VI-CarRealTime system featuring the specific entities can be performed using the VI-CarRealTime Vehicle Wizard.

The creation of auxiliary subsystem can be created through the Auxiliary Sensors/Loads check box in the upper section of the dialog box; as a result entities in the Auxiliary Sensors/Loads tab will become active.

Auxiliary sensors can be configured as explained in the following table:

Parameter	Description
ECU	Enter coordinates of the ECU-Sensor location. This sensor is always active.
Upfront	Select the check box to activate the Upfront-Sensor. Then enter coordinates of the left Upfront-Sensor location, because it is a symmetric sensor.
A-Pillar	Select the check box to activate the A-Pillar-Sensor. Then enter coordinates of the left Upfront-Sensor location, because it is a symmetric sensor.

Parameter	Description
B-Pillar	Select the check box to activate the B-Pillar-Sensor. Then enter coordinates of the left Upfront-Sensor location, because it is a symmetric sensor.
C-Pillar	Select to activate the C-Pillar-Sensor. Then enter coordinates of the left Upfront-Sensor location, because it is a symmetric sensor.
Auxiliary 1	Select to activate the additional Sensor 1. Then enter coordinates of the additional Sensor 1 location.
Auxiliary 2	Select to activate the additional Sensor 2. Then enter coordinates of the additional Sensor 2 location.

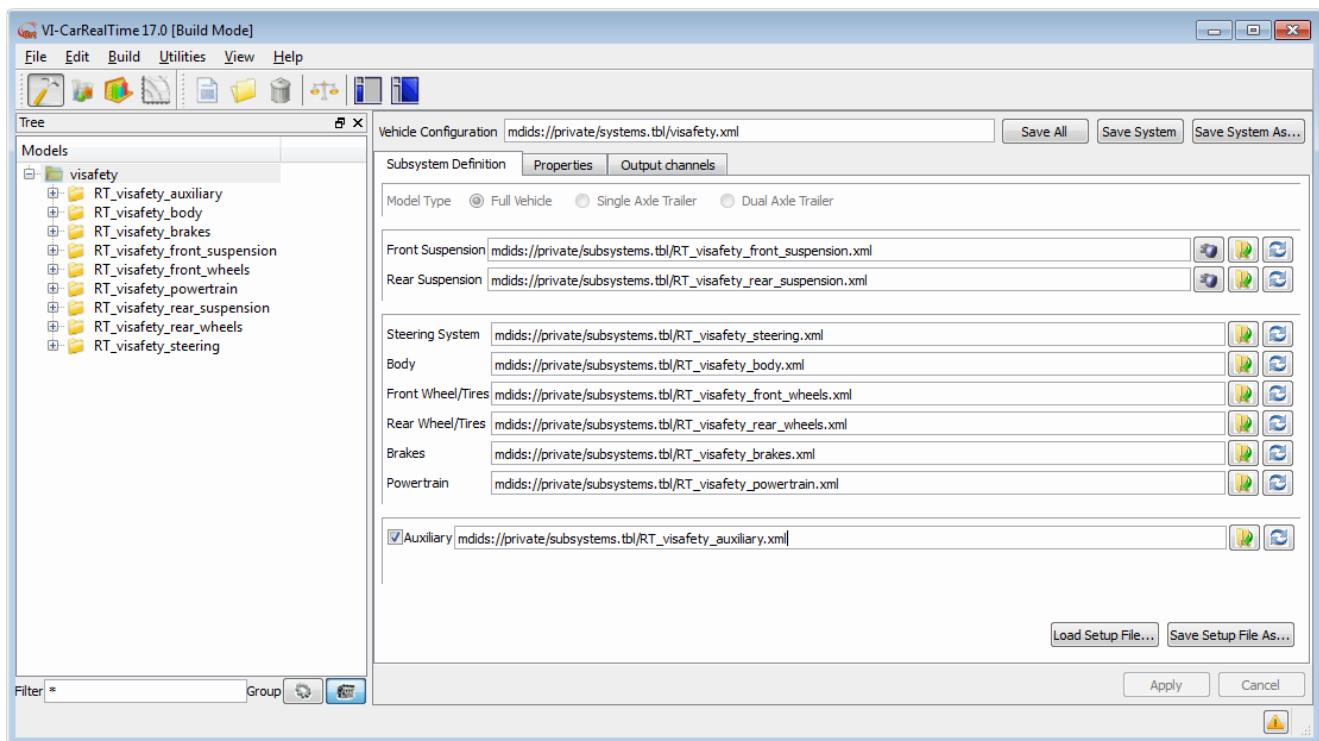
Vehicle Loads can be configured as explained in the following table:

Parameter	Description
Roof Load	Select the check box if the test vehicle should contain a roof load. If yes, enter mass and location of the roof load.

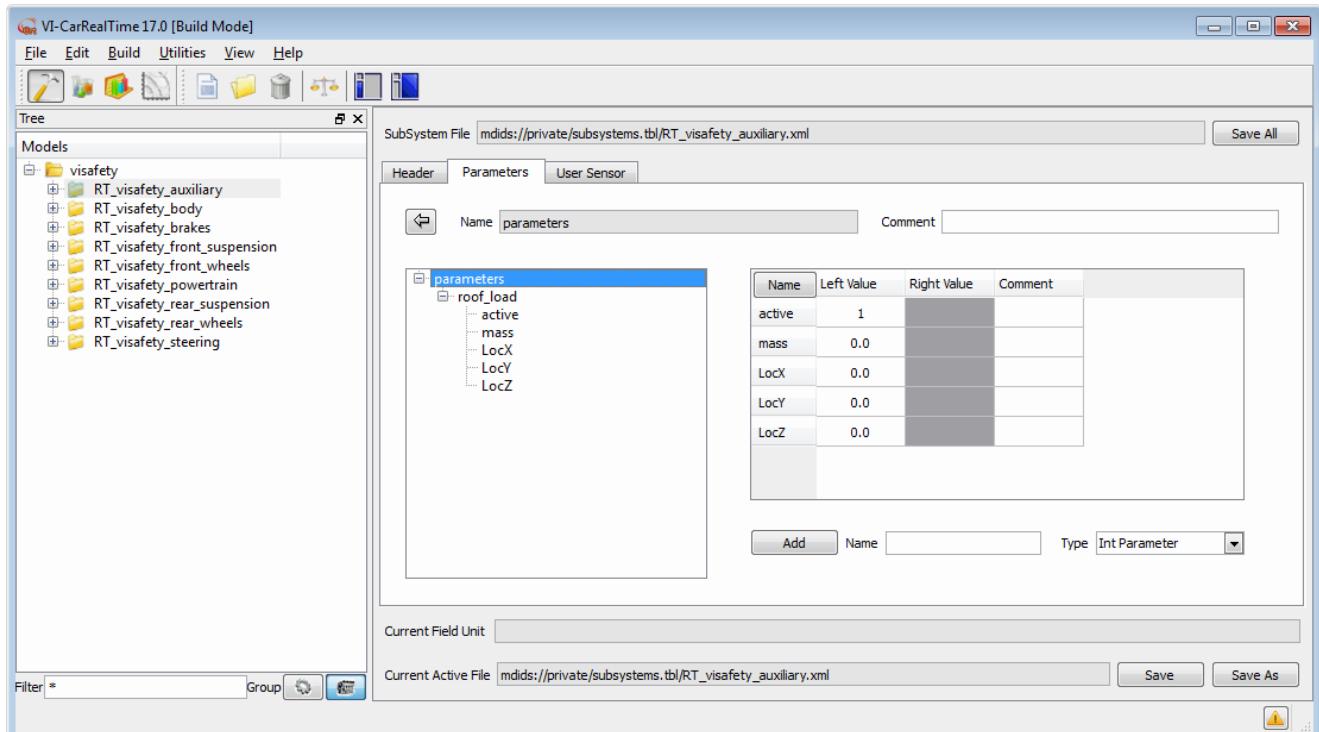
Save Modifications, Block to save Data

Parameter	Description
Name of new Safety Subsystem	Enter name of the new Safety Subsystem.
Save modified Subsystem to disc	Select if you want to save the new created Safety Subsystem to disc.
Save Assembly as	Select if you want to save the extended Assembly (including the Safety Subsystem now) to disc. If yes, enter the name of the new Assembly including the Safety Subsystem.
Target Database	Select the database to store Safety Subsystem and/or Assembly in.

After the creation of the model through the Vehicle Wizard it can be loaded into the session; the system editor will show the auxiliary subsystem in the subsystem definition tab.



Auxiliary subsystem properties can then be edited in the subsystem property editor:



K&C Wizard

K&C Wizard is a tool which allows the creation of VI-CarRealTime vehicle models starting from Kinematics and Compliance Suspension Test-Rig data and a few user entered data.

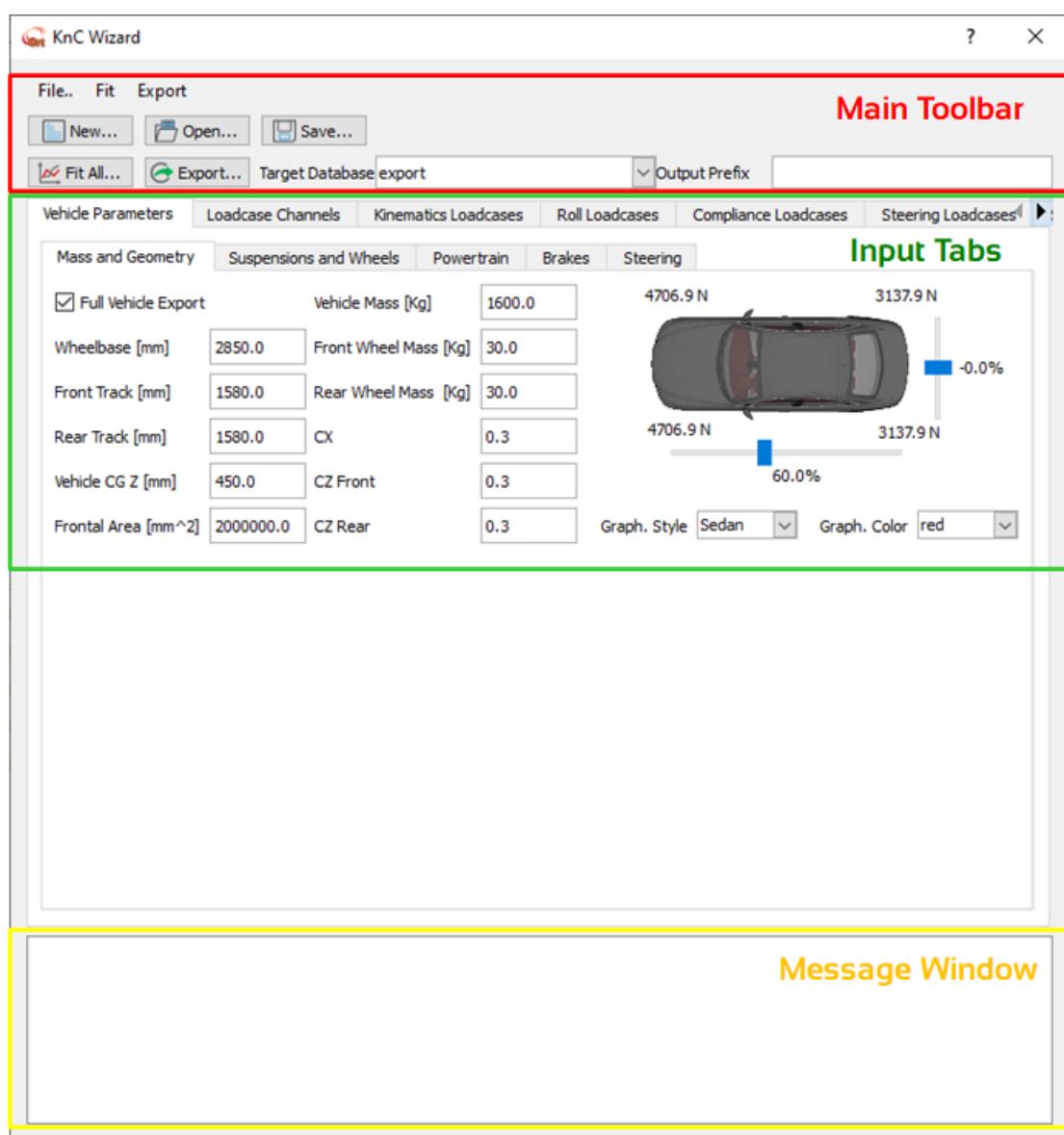
A Kinematics and Compliance test rig is commonly used in the automotive industry to perform *quasi-static* tests on vehicles in order to retrieve the kinematics and compliance characteristics of the suspension and steering systems by imposing a combination of vertical wheel travel, steering wheel angles and applying loads on the wheels in a suite of tests. The resulting curves, that describe the characteristics of the suspensions, are quite valuable for the creation of a VI-CarRealTime model because they can be processed and mapped onto the curves that are used in a VI-CarRealTime suspension subsystem.

The procedure to convert the raw K&C Suspension Test-Rig data into a suspension subsystem file is involving file management and interpolation.

The vehicle model is stored into one of registered databases and the vehicle configuration (containing the set of parameters used to generate the model) can be saved/reloaded to be the starting point for a new vehicle model.

The editor layout features four main sections:

- [Main Toolbar](#)
- [Input Tabs](#)
- [Plotting Area](#)
- [Message Window](#)



Main Toolbar



The following actions are available in the Main Toolbar:

- **New Configuration**
button creates brand new configuration data and cleans up K&C Wizard editor.
- **Load Configuration**
button loads xml data needed to populate K&C Wizard editor. Files are stored in database under knc_configurations.tbl table.
- **Save Configuration**
button saves to xml data populating K&C Wizard editor.
- **Fit All**
button starts loadcases data fitting and populate Plotting area.

- **Export**

button starts vehicle data creation process and populate Plotting area. Created vehicle models will **not** be automatically loaded into VI-CarRealTime session.

- **Target Database**

registered database used to store the model;

- **Output Prefix**

Name for VI-CarRealTime system file.

Input Tabs

The editor features an interface based on tabs that allows the input of all information that the procedure needs in order to create a full vehicle VI-CarRealTime model.

Every tab has a specific structure that allows to enter a different type of data and they can be grouped in four main categories:

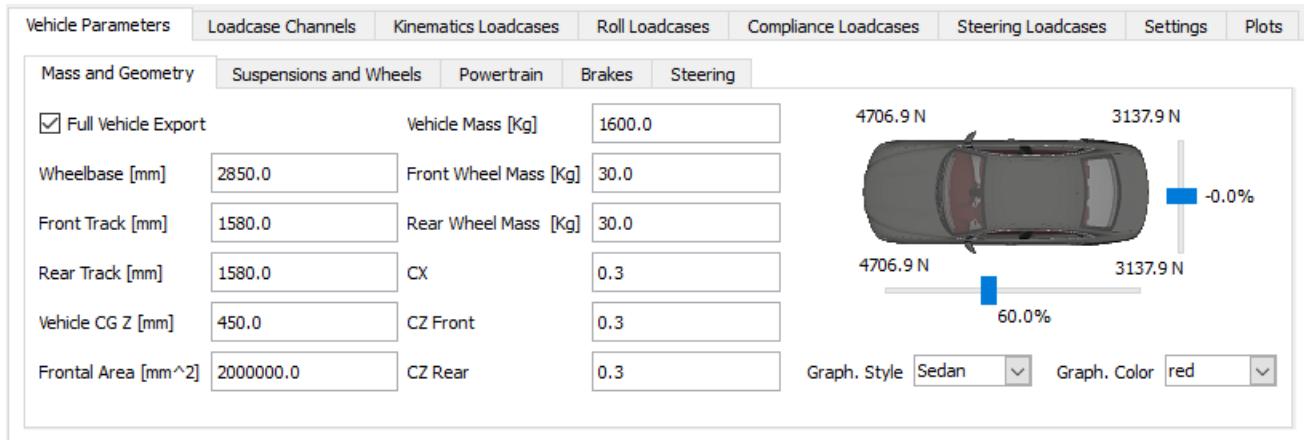
- [Vehicle Parameters](#)
- [Loadcase Channels](#)
- [Loadcases](#)
- [Settings](#)

Read [K&C Data Breakdown Into VI-CarRealTime Model](#) to know more about the use of these data to create a full vehicle model.

Vehicle Parameters

This section of the tool allows to enter every data that can not be retrieve from Kinematics and Compliance Suspension Test-Rig data. They include information from all over the vehicle:

- [Mass and Geometry](#)
- [Suspensions and Wheels](#)
- [Powertrain](#)
- [Brakes](#)



Mass and geometric properties are defined through the following data:

- **Wheelbase**

vehicle wheelbase;

- **Front Track**
vehicle front track;
- **Rear Track**
vehicle rear track;
- **Vehicle CG Z**
vertical position of full vehicle CG;
- **Frontal Area**
vehicle area needed for aerodynamic force computation;
- **Vehicle Mass**
total vehicle mass;
- **Front/Rear Wheel Mass**
unsprung masses values;
- **Weight Front Bias**
portion of vehicle weight acting on front wheels;
- **Weight Lateral Bias**
portion of vehicle weight acting on left wheels.

Aerodynamics properties are defined through the following data:

- **CX (Aero Drag Coefficient)**
coefficient for aero drag force;
- **CZ Front/Rear (Aero Front/Rear DownForce Coefficient)**
coefficient for aero down forces.

Graphics properties are defined through the following data:

- **Graphics Style**
type of vehicle (used to generate chassis graphics); available options are: Compact, GT, Sedan, Open Wheels, SUV, Spider;
- **Graphics Color**
chassis graphics color.

VI-CarRealTime model creation options are the following:

- **Full Vehicle Export:**
The checkbox defines if a full vehicle model will be created at the end of the export procedure. Active status enables the creation of full vehicle; inactive status enables the creation of suspension subsystems only.

Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
<input checked="" type="checkbox"/> Mass and Geometry	<input type="checkbox"/> Suspensions and Wheels	<input type="checkbox"/> Powertrain	<input type="checkbox"/> Brakes	<input type="checkbox"/> Steering			
Front Dampers Motion Ratio	1.0		Rear Dampers motion Ratio	1.0			
Front Dampers Rate [N·sec/mm]	8.0		Rear Dampers Rate [N·sec/mm]	8.0			
Front Tires Property File	mdids://carrealtime_shared/tires.tbl/p	<input type="button" value="..."/>	Rear Tires Property File	mdids://carrealtime_shared/tires.tbl/p	<input type="button" value="..."/>		

The parameters needed to define suspension characteristics are:

- **Front/Rear Dampers Motion Ratio**
damper constant compression ratio value;
- **Front/Rear Dampers Rate**
damper component damping coefficient;
- **Front/Rear Tire Property File**
property file needed for tire force.

Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
Mass and Geometry	Suspicions and Wheels	Powertrain	Brakes	Steering			
Maximum Torque [Nm]	350.0		Transmission Ratio	3.5			
Maximum Torque RPM	5500.0		Front Drive Bias	0.0			
Idle RPM	1000.0		Differential Type	open			
Rev. Limit RPM	8000.0		Gear Ratios		Open...		

Powertrain definition is based on some parameters needed to compute engine torque map.

- **Maximum Torque**
maximum torque value produced by the engine;
- **Max Torque RPM**
engine RPM for maximum torque;
- **Idle RPM**
engine idle RPM;
- **Rev Limit RPM**
engine maximum RPM.

Using the parameters above a predefined shaped engine torque map is created. Further modification can be done in vehicle subsystem editor after loading the model in VI-CarRealTime.

A further set of parameters is needed to fully define the powertrain subsystem:

- **Transmission Ratio**
gearbox transmission ratio;
- **Front Drive Bias**
parameter defining the way drive torque is applied to wheels; FWD=1.0, RWD=0.0, 0.0<AWD<1.0;
- **Differential Type**
type of differential;
- **Gear Ratios**
spline containing ratios for powertrain gears.

Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
Mass and Geometry	Suspensions and Wheels	Powertrain	Brakes	Steering			
Front Brake Bias	0.6						
Friction Coefficient	0.4						
Front Effective Piston Radius [mm]	180.0						
Front Piston Area [mm ²]	2300.0						
Rear Effective Piston Radius [mm]	180.0						
Rear Piston Area [mm ²]	2300.0						

Brake system parameters are:

- **Front Brake Bias**
gives the portion of braking system pressure going to front wheels;
- **Friction Coefficient;**
- **Front/Rear Piston Effective Radius**
radius for applying friction force;
- **Front/Rear Piston Area.**

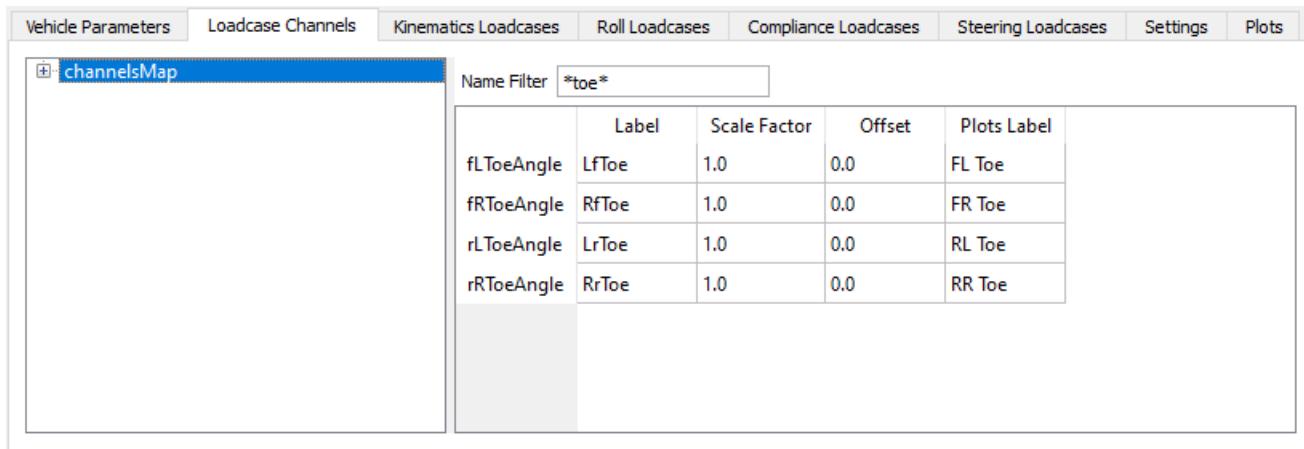
Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
Mass and Geometry	Suspensions and Wheels	Powertrain	Brakes	Steering			
<input checked="" type="checkbox"/> Use Steering Ratio							
Steering Ratio	15.0						

Steering system parameters are:

- **Use Steering Ratio:**
The checkbox defines if Rack Travel channel will be read from Steering Loadcase or defined as Steering Wheel Angle channel divided by Steering Ratio value.
- **Steering Ratio**
Ratio between Steering Wheel Angle and Rack Travel.

Loadcase Channels

Loadcase Channels tab contains a map that link all the channels used by the procedure with the channels inside the K&C loadcase files.



For each of the available channels it is possible to customize:

- **Label**
name of the channel in K&C loadcase file;
- **Scale Factor**
multiplication factor of the raw data of the channel;
- **Offset**
offset factor of the raw data of the channel;
- **Plots Label**
label of the channel used during the creation of the plots in the [Plotting Area](#) and in the [Report File](#).

A description of the channels needed for a full vehicle model export is reported in the [next chapter](#).

In the following table is present a list of the channels needed in order to obtain a complete and working model. All the channels have to be expressed in Body Reference Frame, with orientation compliant to [Vehicle Reference System](#) of VI-CarRealTime, and expressed in SI units with length in millimeters (mm).

Channel Description	MTS Channels Name	ABD Channels Name
Vertical Wheel Travel (variation w.r.t design condition at wheel travel = 0.0)	LfZ0, RfZ0, LrZ0, RrZ0	FLH_wheel_to_body_Z_displacement, FRH_wheel_to_body_Z_displacement, RLH_wheel_to_body_Z_displacement, RRH_wheel_to_body_Z_displacement
Toe Angle	LfToe, RfToe, LrToe, RrToe	Front_LH_wheel_toe-in, Front_RH_wheel_toe-in, Rear_LH_wheel_toe-in, Rear_RH_wheel_toe-in
Camber Angle	LfCamber, RfCamber, LrCamber, RrCamber	Front_LH_wheel_camber, Front_RH_wheel_camber, Rear_LH_wheel_camber, Rear_RH_wheel_camber
Spin Angle (Side-View Angle)	LfSpin, RfSpin, LrSpin, RrSpin	Front_LH_wheel_spin, Front_RH_wheel_spin, Rear_LH_wheel_spin, Rear_RH_wheel_spin
Track (variation w.r.t design condition at wheel travel = 0.0)	LfY0, RfY0, LrY0, RrY0	Front_LH_wheel_centre_Y_disp., Front_RH_wheel_centre_Y_disp., Rear_LH_wheel_centre_Y_disp., Rear_RH_wheel_centre_Y_disp.
Base (variation w.r.t design condition at wheel travel = 0.0)	LfX0, RfX0, LrX0, RrX0	Front_LH_wheel_centre_X_disp., Front_RH_wheel_centre_X_disp., Rear_LH_wheel_centre_X_disp., Rear_RH_wheel_centre_X_disp.
Vertical Force	LfVertForceFbk, RfVertForceFbk, LrVertForceFbk, RrVertForceFbk	Front_LH_wheel_Force_Z, Front_RH_wheel_Force_Z,

		Rear_LH_wheel_Force_Z, Rear_RH_wheel_Force_Z
Lateral Force	LfLatForceFbk, RfLatForceFbk, LrLatForceFbk, RrLatForceFbk	Front_LH_wheel_Force_Y, Front_RH_wheel_Force_Y, Rear_LH_wheel_Force_Y, Rear_RH_wheel_Force_Y
Longitudinal Force applied at Contact Patch or at Wheel center depending on the Test Case	LfLongForceFbk, RfLongForceFbk, LrLongForceFbk, RrLongForceFbk	Front_LH_wheel_Force_X, Front_RH_wheel_Force_X, Rear_LH_wheel_Force_X, Rear_RH_wheel_Force_X
Aligning Torque	LfAligningTorqueFbk, RfAligningTorqueFbk, LrAligningTorqueFbk, RrAligningTorqueFbk	Front_LH_wheel_Moment_Z, Front_RH_wheel_Moment_Z, Rear_LH_wheel_Moment_Z, Rear_RH_wheel_Moment_Z
Steer Angle	LfSteer, RfSteer, LrSteer, RrSteer	-Front_LH_wheel_toe-in, Front_RH_wheel_toe-in, -Rear_LH_wheel_toe-in, Rear_RH_wheel_toe-in
Steering Wheel Angle	SteeringWheelAngle	steering_wheel_position

Loadcases

This section of the tool allows to enter the information about the loadcase files to be used for the creation of the model.

Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
Name Filter <input type="text"/> Fit.							
PWT_Kin	parallel travel	front and rear	any	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vvert	
OWT_Kin	opposite travel	front and rear	any	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vroll	
SWT_FL_Kin	single travel	front	left	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vvert	
SWT_RR_Kin	single travel	rear	right	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vvert	
SWT_FR_Kin	single travel	front	right	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vvert	
SWT_RL_Kin	single travel	rear	left	no symmetry	SampleTime	mdids://KnC/knc_results.tbl/SEDAN_4WD/vvert	
< > Add Name							

The layout of the tabs provides for the use of a table where every loadcase is entered on a single line specifying the following parameters:

- **Name**
name of the K&C loadcase;
- **Type**
type of the K&C loadcase (see [Loadcase Types](#) for available options);
- **Role**
axle where K&C measurement machine operates (available options are front, rear, front and rear);
- **Side**
side of the axle where K&C applies forces (available options are left, right, any);
- **Symmetry**
symmetry of the fitted data (available options are no symmetry, left to right, right to left, average); for every [Test Case Analysis](#) there is a certain type of loadcase whose symmetry is prevalent on the others;

- **Header Start String**

name of the first channel in K&C loadcase file; it is used to locate the start of the table of channels in the K&C loadcase file (see [Data Management](#) for further information);

- **File**

path of the K&C loadcase file.

Loadcases are used exclusively for the analysis concerning their own tab. For example, Opposite Wheel Travel loadcases in Kinematics tab are used to define kinematics of suspensions but they are not used in order to define Anti-Roll Bar element.

[Loadcase Types](#)

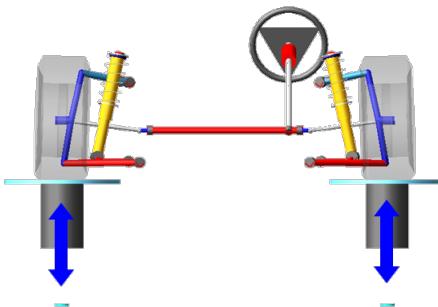
[Test Case Analysis](#)

[Data Management](#)

Four different main [Loadcase Types](#) exist:

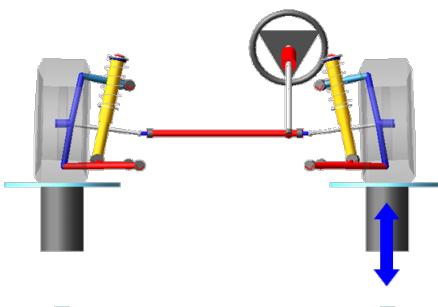
- [Kinematics Loadcases](#)
- [Roll Loadcases](#)
- [Compliance Loadcases](#)
- [Steering Loadcases](#)

Parallel Wheel Travel



Vertical movement is applied in phase to both the wheels of a single axle.
Wheel travel sweep can be eventually repeated.

Single Wheel Travel

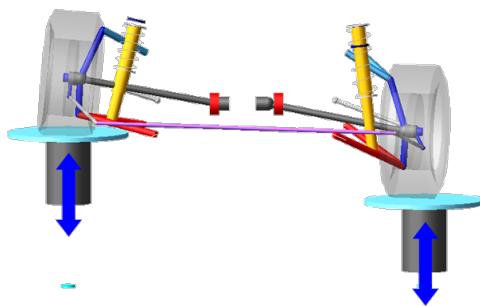


Vertical movement is applied to one wheel of the axle holding the opposite wheel fixed in a specified position. Suspension kinematic properties are recorded for both wheels allowing to detect cross kinematic effect on the wheel not subjected to vertical movement.

Wheel travel sweep can be eventually repeated.

Multiple Single Wheel Travel loadcases are possible changing the jounce of the fixed wheel from a loadcase to another.

Opposite Wheel Travel (with Anti Roll Bar)



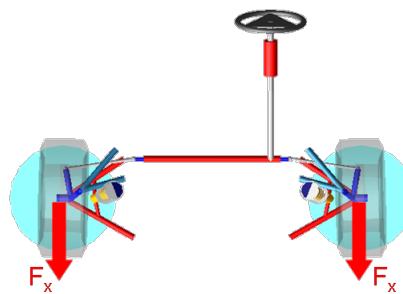
Vertical movement is applied out of phase to the wheels of a single axle to simulate body roll.

Wheel travel sweep can be eventually repeated.

Anti Roll Bar is connected during the test.

Multiple Opposite Wheel Travel loadcases are possible changing the average wheel travel from a loadcase to another.

Contact Patch Longitudinal Force Aiding

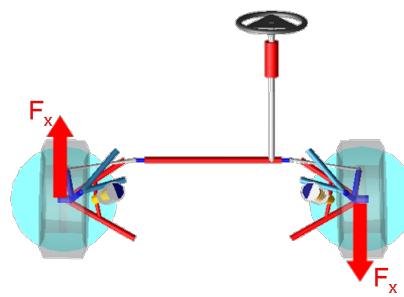


A pair of longitudinal forces is applied at contact patches; force intensity sweeps a range, possibly from negative to positive values. Force value is the same (in phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Contact Patch Longitudinal Force Opposing

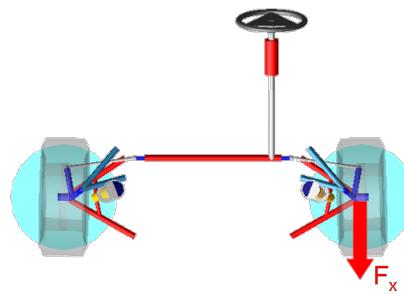


A pair of longitudinal forces is applied at contact patches; force intensity sweeps a range, possibly from negative to positive values. Force value is opposite (out of phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Contact Patch Longitudinal Force Single

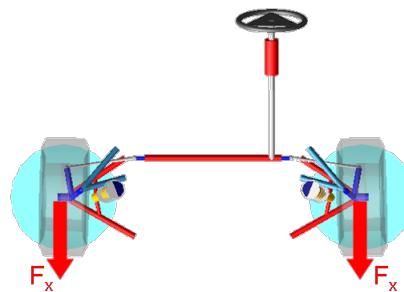


A longitudinal force is applied to a single contact patch; force intensity sweeps a range, possibly from negative to positive values.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. K&C Wizard procedure allows only for loadcases with the same value of jounce on both wheels.

Wheel Center Longitudinal Force Aiding

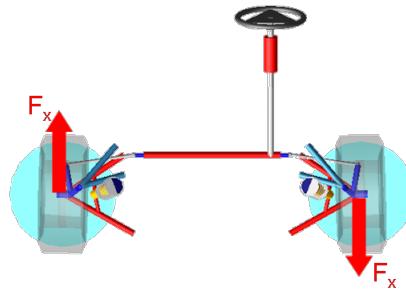


A pair of longitudinal forces is applied at wheel centers; force intensity sweeps a range, possibly from negative to positive values. Force value is the same (in phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Wheel Center Longitudinal Force Opposing

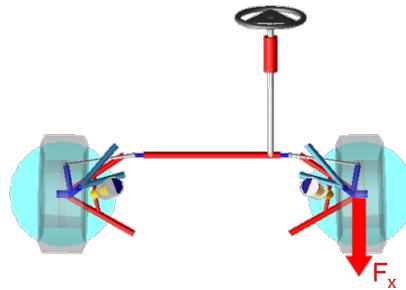


A pair of longitudinal forces is applied at wheel centers; force intensity sweeps a range, possibly from negative to positive values. Force value is opposite (out of phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Wheel Center Longitudinal Force Single

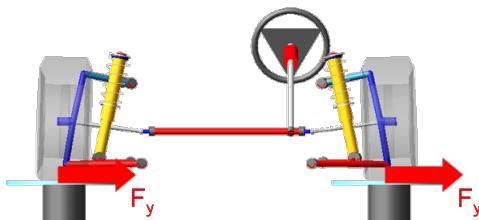


A longitudinal force is applied to a single wheel center; force intensity sweeps a range, possibly from negative to positive values.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. K&C Wizard procedure allows only for loadcases with the same value of jounce on both wheels.

Contact Patch Lateral Force Aiding

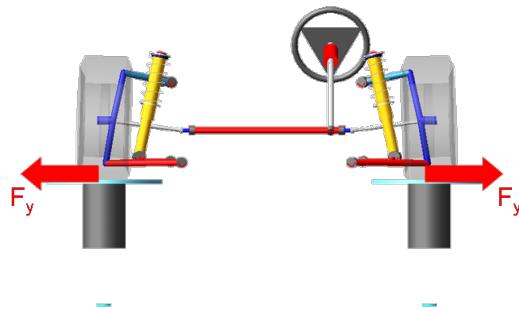


A pair of lateral forces is applied at contact patches; force intensity sweeps a range, possibly from negative to positive values. Force value is the same (in phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Contact Patch Lateral Force Opposing

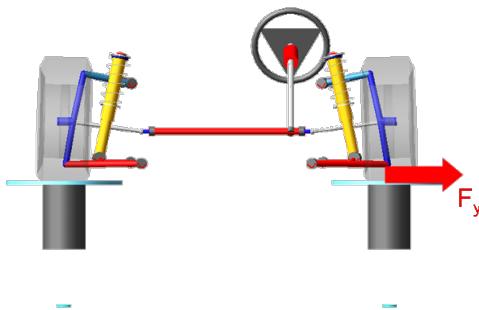


A pair of lateral forces is applied at contact patches; force intensity sweeps a range, possibly from negative to positive values. Force value is opposite (out phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Contact Patch Lateral Force Single

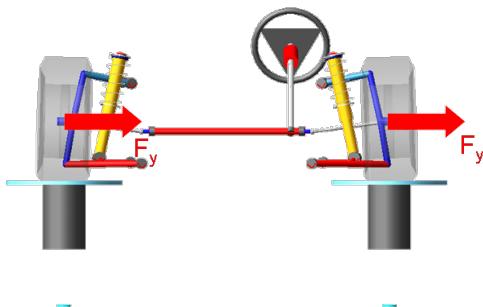


A single lateral force is applied at contact patch; force intensity sweeps a range, possibly from negative to positive values.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. K&C Wizard procedure allows only for loadcases with the same value of jounce on both wheels.

Wheel Center Lateral Force Aiding

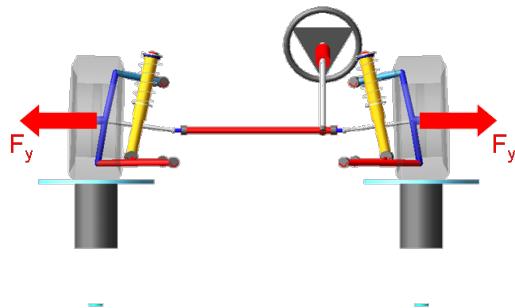


A pair of lateral forces is applied at wheel centers; force intensity sweeps a range, possibly from negative to positive values. Force value is the same (in phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Wheel Center Lateral Force Opposing

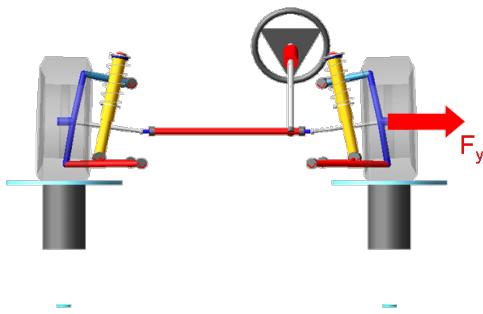


A pair of lateral forces is applied at wheel centers; force intensity sweeps a range, possibly from negative to positive values. Force value is opposite (out phase) on both sides.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Wheel Center Lateral Force Single

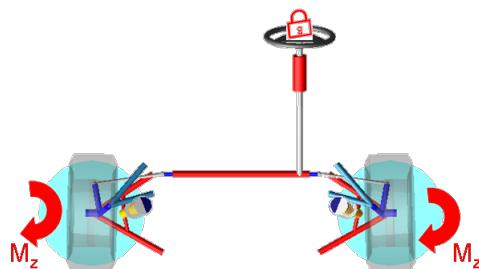


A single lateral force is applied at wheel centers; force intensity sweeps a range, possibly from negative to positive values.

Force sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. K&C Wizard procedure allows only for loadcases with the same value of jounce on both wheels.

Aligning Torque Aiding

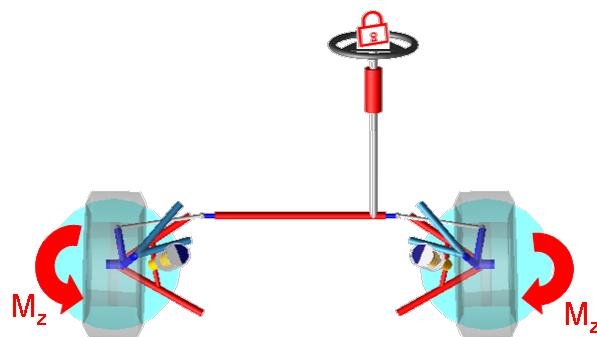


A pair of aligning torques is applied to wheels; torque intensity sweeps a range, possibly from negative to positive values. Torque value is the same (in phase) on both sides.

Torque sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Aligning Torque Opposing

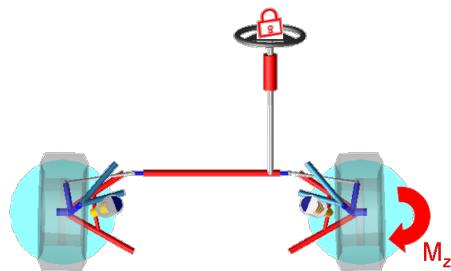


A pair of aligning torques is applied to wheels; torque intensity sweeps a range, possibly from negative to positive values. Torque value is opposite (out of phase) on both sides.

Torque sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. The jounce of both wheels have to be the same.

Aligning Torque Single

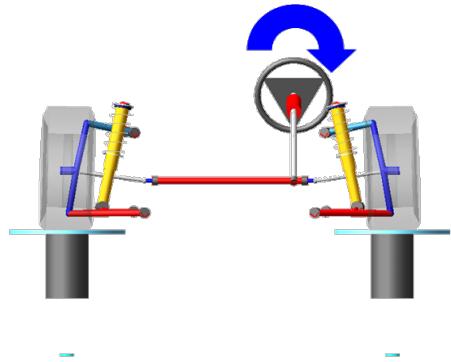


A single aligning torques is applied to a wheel; torque intensity sweeps a range, possibly from negative to positive values.

Torque sweep can be eventually repeated along the test.

Multiple loadcases are possible changing the jounce of both wheels from a loadcase to another. K&C Wizard procedure allows only for loadcases with the same value of jounce on both wheels.

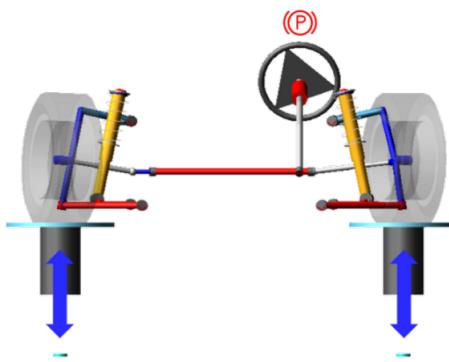
Steering



Motion input is applied to steering wheel. Wheel travel is kept constant during the load case. Suspension kinematic properties are recorded for both wheels.

Multiple Steering loadcases are possible changing the jounce the wheels from a loadcase to another.

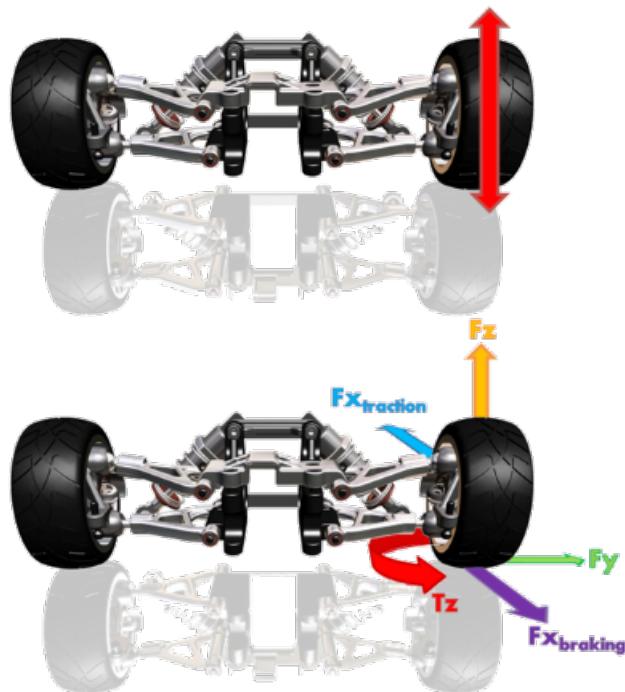
Vertical



Vertical movement is applied to both wheels of the axle holding the steering wheel fixed in a specified position. Suspension kinematic properties are recorded for both wheels.

Multiple Vertical Travel loadcases are possible changing the steering wheel angle from a loadcase to another.

Kinematics & Compliance Test-Rig can be used to retrieve the needed data for populating a VI-CarRealTime subsystem file.



The following test cases must be included:

- [Suspension Kinematics Load Cases](#)
- [Suspension Roll Load Cases](#)
- [Suspension Compliance Load Cases](#)
- [Steering Kinematics Load Cases](#)

In order to extract suspension kinematics properties from K&C tests, one of the following load cases combination must be included into experimental data set:

- **Parallel Wheel Travel**

The test case is used to obtain independent kinematic curves for suspension.

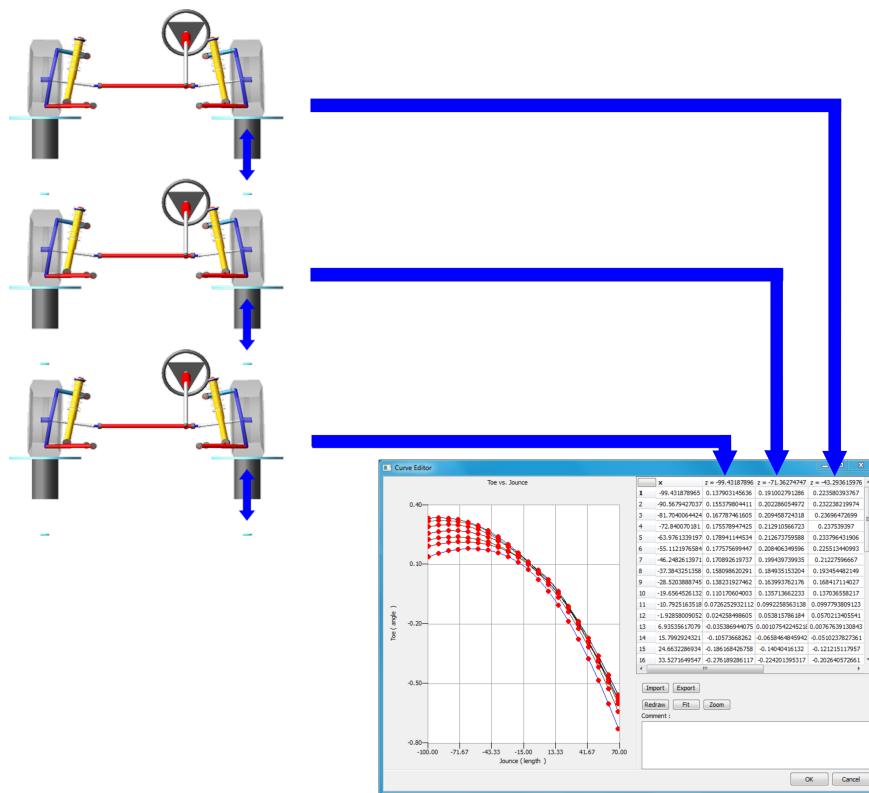
Parallel Wheel Travel loadcase leads Symmetry options.

- **Single Wheel Travels**

Three or more Single Wheel Travel test cases (for each wheel) are used to obtain dependent kinematic curves for suspension.

In case the number of loadcases is higher than five, every single loadcase corresponds to one column of the kinematics tables in VI-CarRealTime model; otherwise, the procedure interpolate between the curves in order to make data consistent to VI-CarRealTime curve data requirements.

First entered Single Wheel Travel loadcase of Side left leads Symmetry options.

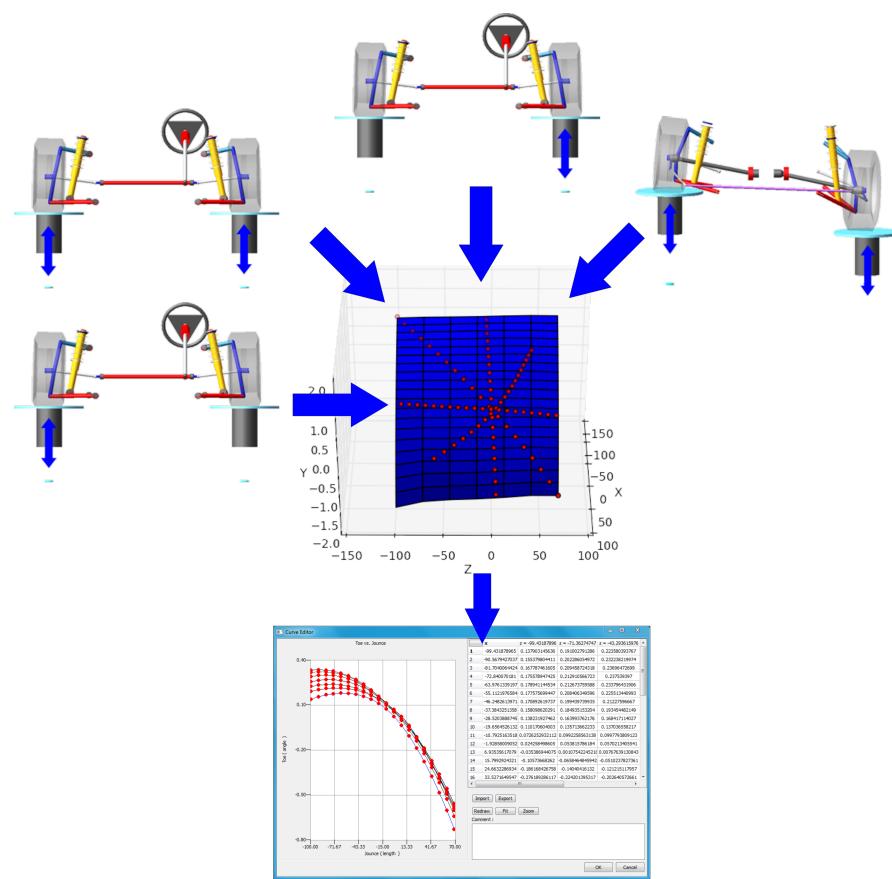


- **Parallel Wheel Travel - Single Wheel Travel - Opposite Wheel Travel**

This combination of test cases is used to obtain dependent kinematic curves for suspension.

Suspension kinematic properties, function of left and right wheel travel, from every single load case, are interpolated in order to create a surface consistent to VI-CarRealTime curve data requirements. Before to perform the interpolation of the surface, the procedure removes the offset between the kinematic properties from Parallel Wheel Travel loadcase and the same properties from the other loadcases; the computation of the offset is performed where the data pass through the same wheel travels.

Parallel Wheel Travel loadcase leads Symmetry options.



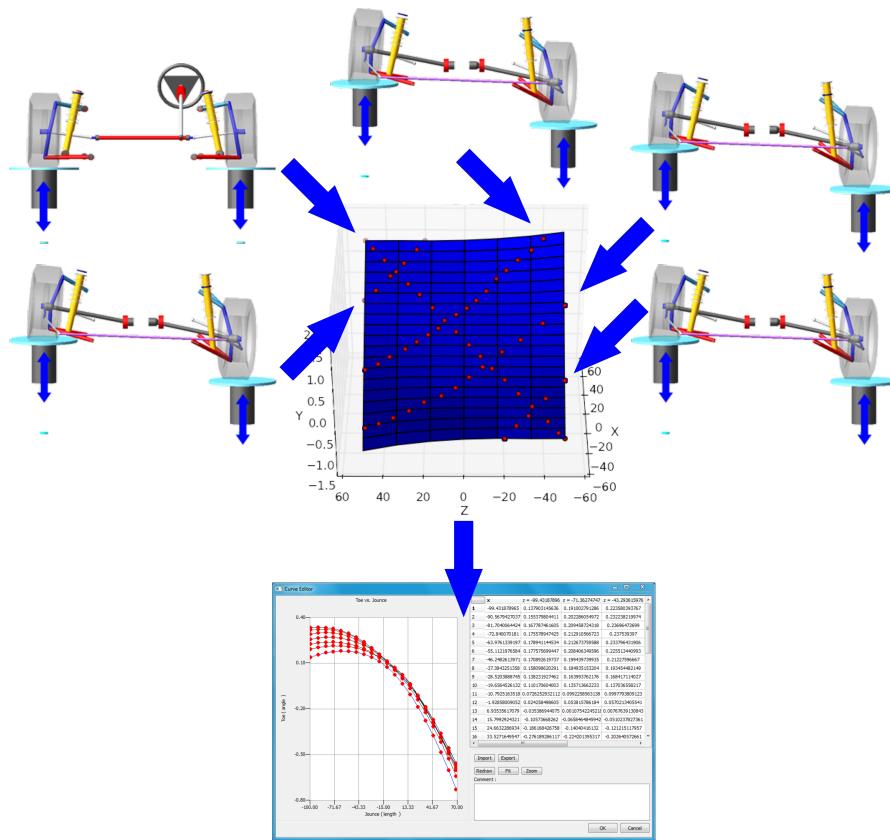
• Parallel Wheel Travel - Opposite Wheel Travels

This combination of test cases is used to obtain dependent kinematic curves for suspension. Suspension kinematic properties, function of left and right wheel travel, from every single load case, are interpolated in order to create a surface consistent to VI-CarRealTime curve data requirements. Before to perform the interpolation of the surface, the procedure removes the offset between the kinematic properties from Parallel Wheel Travel loadcase and the same properties from the other loadcases; the computation of the offset is performed where the data pass through the same wheel travels.

Parallel Wheel Travel loadcase leads Symmetry options.

Opposite Wheel Travels requirements:

- delta wheel travel have to be big enough to cover almost the whole space identified by the wheel travel during Parallel Wheel Travel test;
- spacing between average wheel travel for every loadcase have to be small.

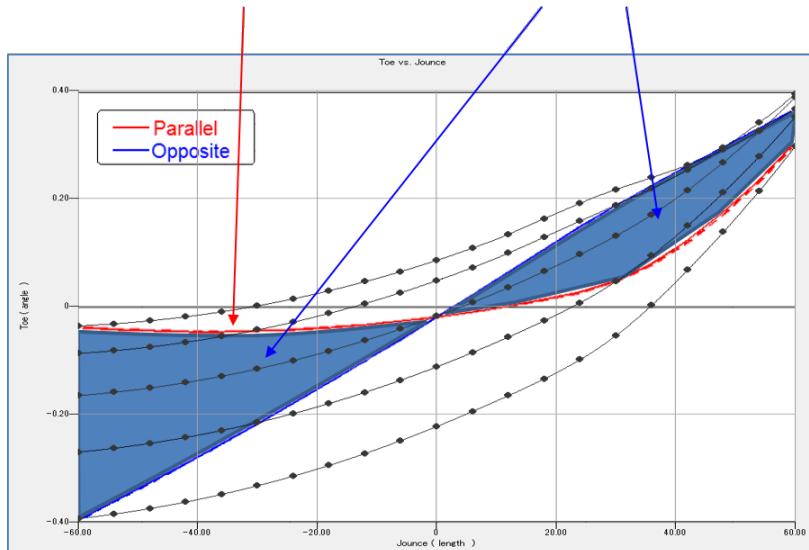


- **Parallel Wheel Travel - Opposite Wheel Travel**

This test case combination is used to obtain dependent kinematic curves for suspension.

The algorithm creates dependent kinematic curves summing up, in every point of the map, the value of the variable (Toe, for example) in Parallel Wheel Travel loadcase and the delta of the variable between Parallel Wheel Travel and Opposite Wheel Travel loadcases.

$$\text{Toe} = f(\text{toe}, \text{jounce_parallel}) + f(\text{delta_toe}, \text{delta_jounce})$$



If more than one Test case is available, they are applied in the order presented above.

The effect of anti roll devices (ARB) can be modeled in VI-CarRealTime using two methods:

- Auxiliary roll forces: effect of device is projected to ground and expressed as vertical forces at wheel centers as function of axle wheel travel difference;
- Component: effect of device is described similarly to other elastic components (springs) using motion ratio and component properties.

K&C Wizard is supporting the possibility of converting anti roll devices data using the first method.

In order to isolate the effect of anti roll device (ARB) from other elastic components one of the following combinations is required:

- **Parallel Wheel Travel - Opposite Wheel Travel**

Subtracting the vertical forces at wheels measured during the two load cases described above it is possible to retrieve auxiliary anti roll force as a Single Table (see [Auxiliary Anti Roll Force](#)).

Parallel Wheel Travel loadcase leads Symmetry options.

Wheel Travel range of Parallel Wheel Travel loadcase have to include the range of Opposite Wheel Travel loadcase.

- **Parallel Wheel Travel - Single Wheel Travels**

Subtracting the vertical forces at wheels measured during the two load cases described above it is possible to retrieve auxiliary anti roll force as Dual Table (see [Auxiliary Anti Roll Force](#)).

Parallel Wheel Travel loadcase leads Symmetry options.

In order to extract suspension compliance properties from K&C tests, one of the following load cases combination must be included into experimental data set:

- **Direct (aiding loads)**

The test case consists of single load case where in phase forces/moments are applied to both axle wheels at the same time.

In this case, only direct compliance is extracted.

- **Combined (aiding and opposing loads)**

The test case consists of two load cases where in phase and out of phase forces/moments are applied to both axle wheels at the same time, respectively.

In this case, direct and cross compliance are extracted solving the following systems of equations:

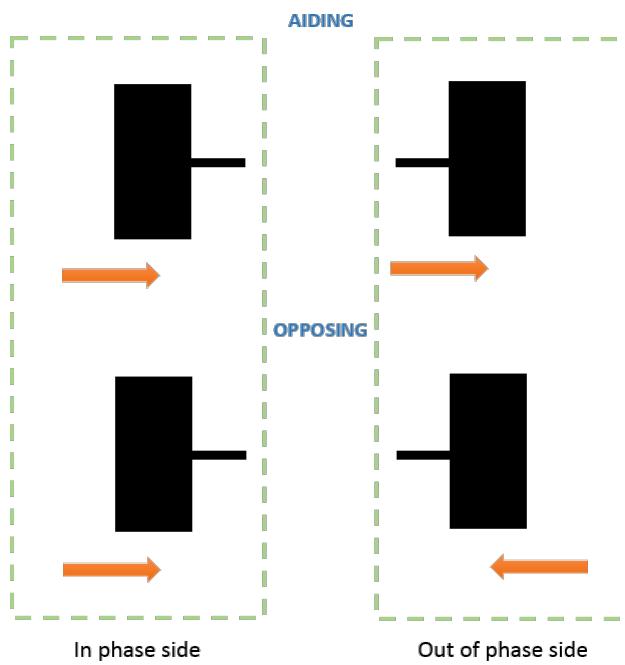
$$\begin{cases} AID_{in}(F) = D_{in}(F) + C_{in}(F) \\ OPP_{in}(F) = D_{in}(F) + C_{in}(-F) \end{cases}$$

$$\begin{cases} AID_{out}(F) = D_{out}(F) + C_{out}(F) \\ OPP_{out}(-F) = D_{out}(-F) + C_{out}(F) \end{cases}$$

where:

- AID(F) is the suspension property in the Aiding loadcase function of the applied force;
- OPP(F) is the suspension property in the Opposing loadcase function of the applied force;
- D(F) is the Direct compliance function of the applied force;
- C(F) is the Cross compliance function of the applied force;
- the suffix *in* and *out* indicate the in-phase and out of phase side, respectively.

The in phase side is the side where forces/moments are applied in phase for the two loadcases of aiding and opposing loads; out of phase side is the side where the forces/moments are applied in antiphase for the two loadcases (see the following figure).



In order to solve the two systems the following assumptions are used:

- Direct compliance is an odd function ($-D(F) = D(-F)$);
- Cross compliance is an odd function ($-C(F) = C(-F)$).

If the last assumptions are not satisfied, single load cases should be used.

In order to obtain better results, Aiding and Opposing loads should have the same range variation.

- **Single (single wheel loads)**

The test case consists of single load case where force/moment is applied to a single wheel.

In this case, direct (from wheel on which is applied the load) and cross (from wheel on which no load is applied) compliance data are retrieved.

First entered Aiding or Single loadcase leads Symmetry options for every Test Case.

Below is a list of allowed test cases:

- Contact Patch Longitudinal Force Aiding
- Contact Patch Longitudinal Force Aiding - Opposing
- Contact Patch Longitudinal Force Single
- Wheel Center Longitudinal Force Aiding
- Wheel Center Longitudinal Force Aiding - Opposing
- Wheel Center Longitudinal Force Single
- Contact Patch Lateral Force Aiding
- Contact Patch Lateral Force Aiding - Opposing
- Contact Patch Lateral Force Single
- Wheel Center Lateral Force Aiding
- Wheel Center Lateral Force Aiding - Opposing
- Wheel Center Lateral Force Single
- Aligning Torque Aiding
- Aligning Torque Aiding - Opposing
- Aligning Torque Single

In order to extract steering kinematics properties from K&C test the following load cases must be included into experimental data set:

- **Steering**

Test case(s) is used to compute suspension kinematics dependency on steering input. Motion input is applied to steering wheel. Wheel travel is kept constant during the load case.

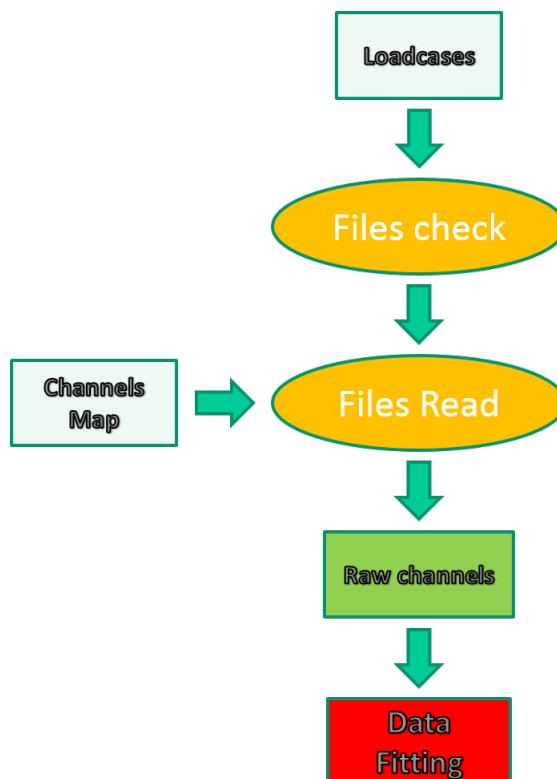
First entered Steering loadcase leads Symmetry options.

- **Vertical**

The test case consists of at least 5 load cases where vertical motion is applied in phase to both wheels of the same axle while holding the steering wheel fixed in a specified position. The steering position variates from a load case to another from positive lock to negative lock position.

The addition of a steering load case at zero jounce is mandatory.

First entered Steering loadcase leads Symmetry options.



K&C data reading is configured as an automatic procedure using the following inputs:

- [Loadcase Channels](#) map
- [Loadcases](#) : raw data files

The following operations are performed during raw data reading:

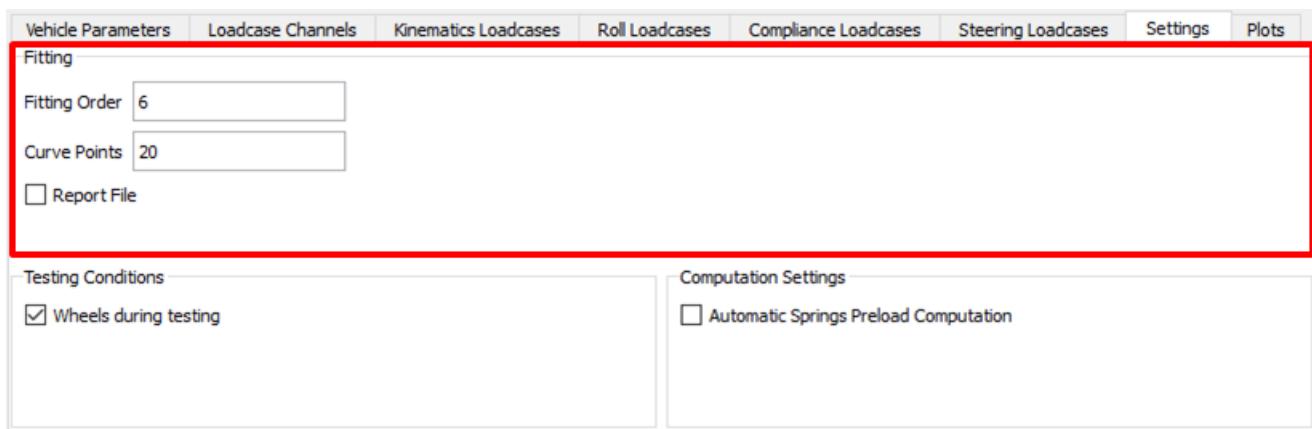
1. retrieve of K&C data file names from [Loadcases](#);
2. check existence of all load case files;
3. retrieve channel names, scaling factors and offsets from [Loadcase Channels](#) map;
4. retrieve all data channel values from load case files;
5. send raw data to data [Fitting](#) module.

Settings

This section of the tool allows to enter general settings of the tool regarding:

- [Fitting](#)
- [Testing Conditions](#)
- [Computation Settings](#)

Fitting group contains fitting parameters used by the procedure to convert raw data into curves.

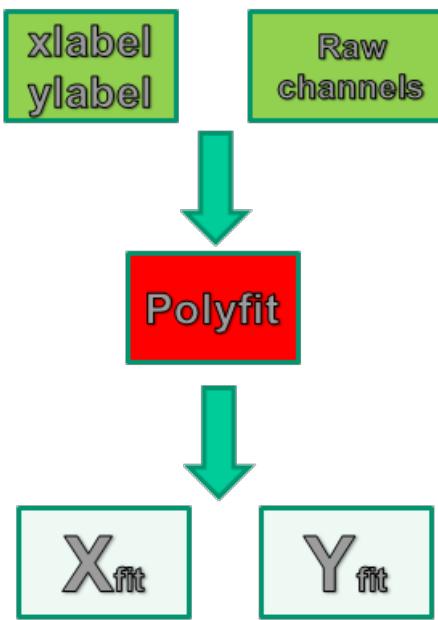


Customizable fitting parameters are:

- **Fitting Order**
order of the polynomial fitting;
- **Curve Points**
number of point of the output curves;
- **Report File**
The checkbox defines if the tool will create an html report of the data fitting procedure.

[Data Fitting Methodology](#)

[Report File](#)



The data fitting module is the portion of the process that is utilized to convert the raw data into curves (lookup tables) suitable for the VI-CarRealTime model. In particular it is needed to get a set of data with a monotonic ascending x-vector and to filter hysteresis in the raw data. The module gets as input the label of the x and y channels and the raw data set (all channels contained in the single file). The two channels identified through the labels are then subjected to polynomial fitting (n-th order) and the resulting vectors are returned within the same interval of inputs with the number of points selected by the user. The module is invoked each time a curve vector pair is required to compute a curve for the VI-CarRealTime model.

[K&C Wizard](#) is featuring fitting report utilities.

Selecting the Fitting Report check box in [K&C Fitting Tab](#) it is possible to create an html report of the data fitting procedure comparing raw data curves with fitting results.

K&C Testrig Data Input Fitting Report

Loadcase Type	Role	Side	Path
parallel travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_PWT_Front.tab
parallel travel	rear	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_PWT_Rear.tab
opposite travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Front_0.tab
opposite travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Front_up35.tab
opposite travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Front_up17.tab
opposite travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Front_dn17.tab
opposite travel	front	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Front_dn35.tab
opposite travel	rear	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Rear_0.tab
opposite travel	rear	any	mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_OWT_Rear_up35.tab

align torque single	front	right	mdids://Sedan_KnC/knc_results.tbl/Single/Sedan_Single_Align_Right_F.tab
align torque single	front	left	mdids://Sedan_KnC/knc_results.tbl/Single/Sedan_Single_Align_Left_F.tab
align torque single	rear	right	mdids://Sedan_KnC/knc_results.tbl/Single/Sedan_Single_Align_Right_R.tab
align torque single	rear	left	mdids://Sedan_KnC/knc_results.tbl/Single/Sedan_Single_Align_Left_R.tab

Validation Plots

FLTOEANGLE vs FLWHEELTRAVEL

Analysis: front parallel travel
Input File: mdids://Sedan_KnC/knc_results.tbl/PWT_OWT/Sedan_PWT_Front.tab

Testing Conditions group contains parameters used by KnC Wizard to understand the data in load case files.

Vehicle Parameters	Loadcase Channels	Kinematics Loadcases	Roll Loadcases	Compliance Loadcases	Steering Loadcases	Settings	Plots
Fitting							
Fitting Order	6						
Curve Points	20						
<input type="checkbox"/> Report File							
Testing Conditions				Computation Settings			
<input checked="" type="checkbox"/> Wheels during testing				<input type="checkbox"/> Automatic Springs Preload Computation			

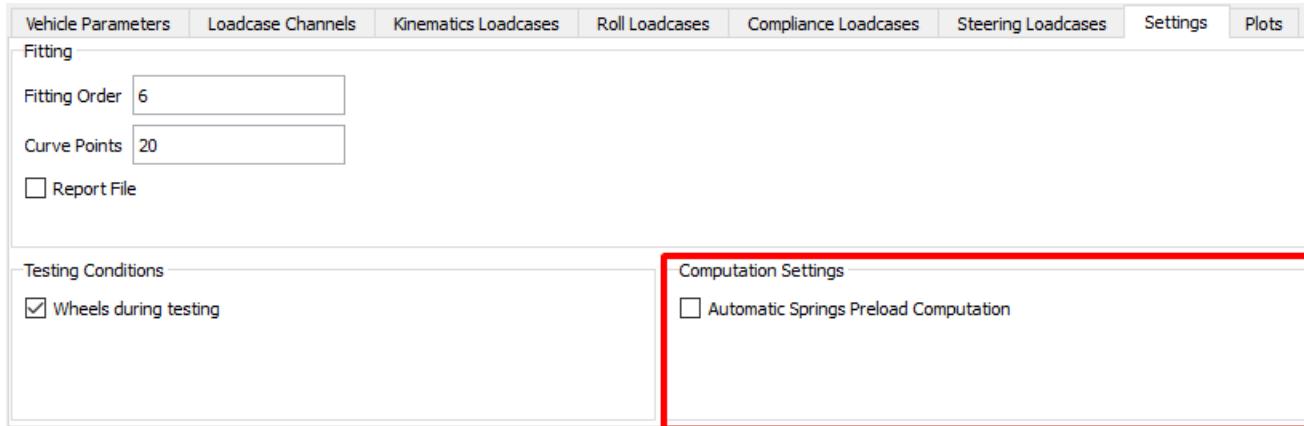
Customizable testing conditions are:

- **Wheels during testing**

The checkbox defines if, during the Kinematics and Compliance analyses, the wheels were present on the assembly.

If checked, the procedure will remove the force due to the wheels during the generation of the elastic elements of the vehicle.

Computation Settings group contains parameters used by the procedure to switch between different modes of calculation of various elements of the vehicle.



Customizable computation settings are:

- **Automatic Springs Preload Computation**

The checkbox defines if the procedure will compute the spring preload using the load on one corner of the vehicle calculated in one of these two ways:

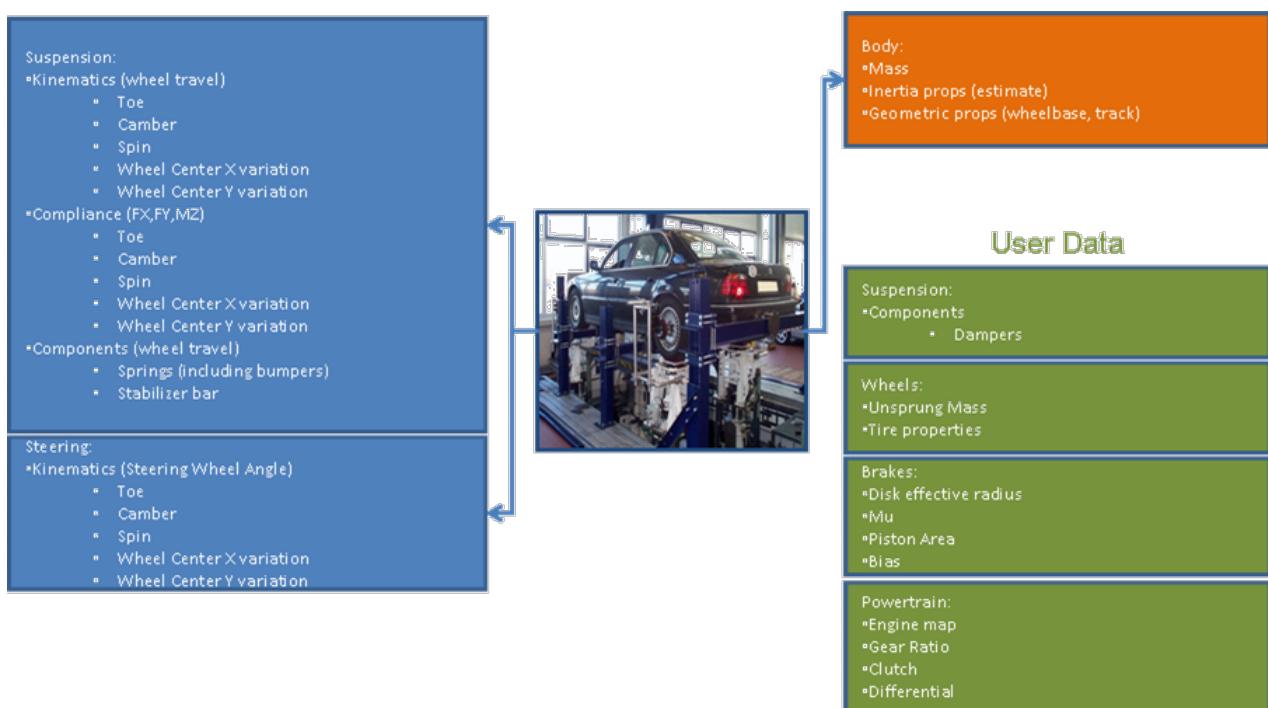
- using the [Mass and Geometry](#) data in the panel of KnC Wizard;
- using the value of the normal force on the wheel during Parallel Wheel Travel analysis at zero jounce.

K&C Data Breakdown Into VI-CarRealTime Model

The figure below shows the breakdown of model data required to populate a VI-CarRealTime model from K&C data. The [K&C Wizard](#) allows you to easily select files and to enter suspension data as well as mandatory user parameters; in fact only a few fields have to be filled to complete the process.

The chart contains the following information:

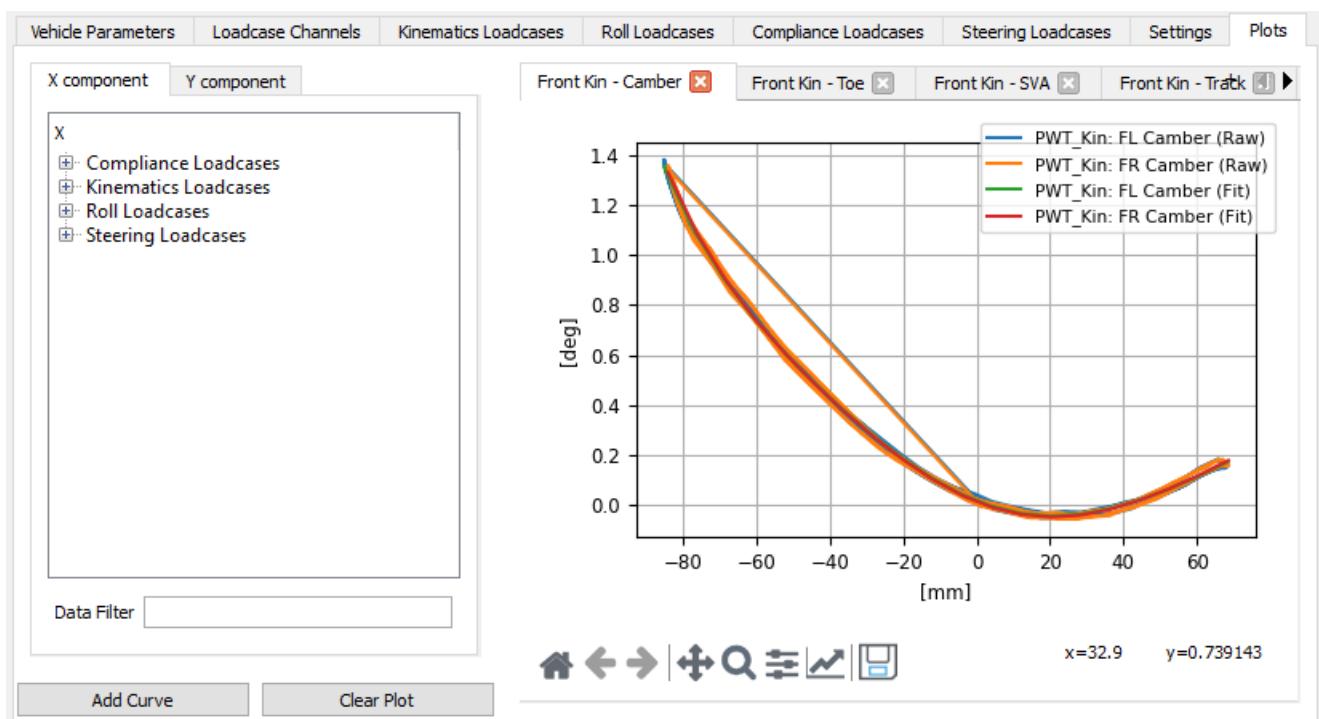
- The information listed in the blue boxes are the curves resulting from loadcase data fitting, they refer directly to K&C time domain measurements: suspension kinematics, suspension compliance, steering kinematics, spring and ARB components.
- The information in the orange box are vehicle parameters manually entered and mapped through the [Vehicle Parameters](#) tabs.
- The information in the green boxes are user input, because typically this type of date are not included in K&C data sets.



With regard to suspension components such as bumpstops and springs it must be noted that since in VI-CarRealTime they are described through motion ratios and component properties they are treated as follows in the case of import of from K&C data sets:

- Springs: uniform motion ratio is assumed; spring property is projected onto wheel level, including bumpstop and reboundstop forces.
- Damper: it is not possible to retrieve damping force from K&C quasi-static measurements; property (damping ratio) must be input by the user.
- Bumpstop and Reboundstops: are not included, the respective instances are created with dummy null values in the VI-CarRealTime data set.

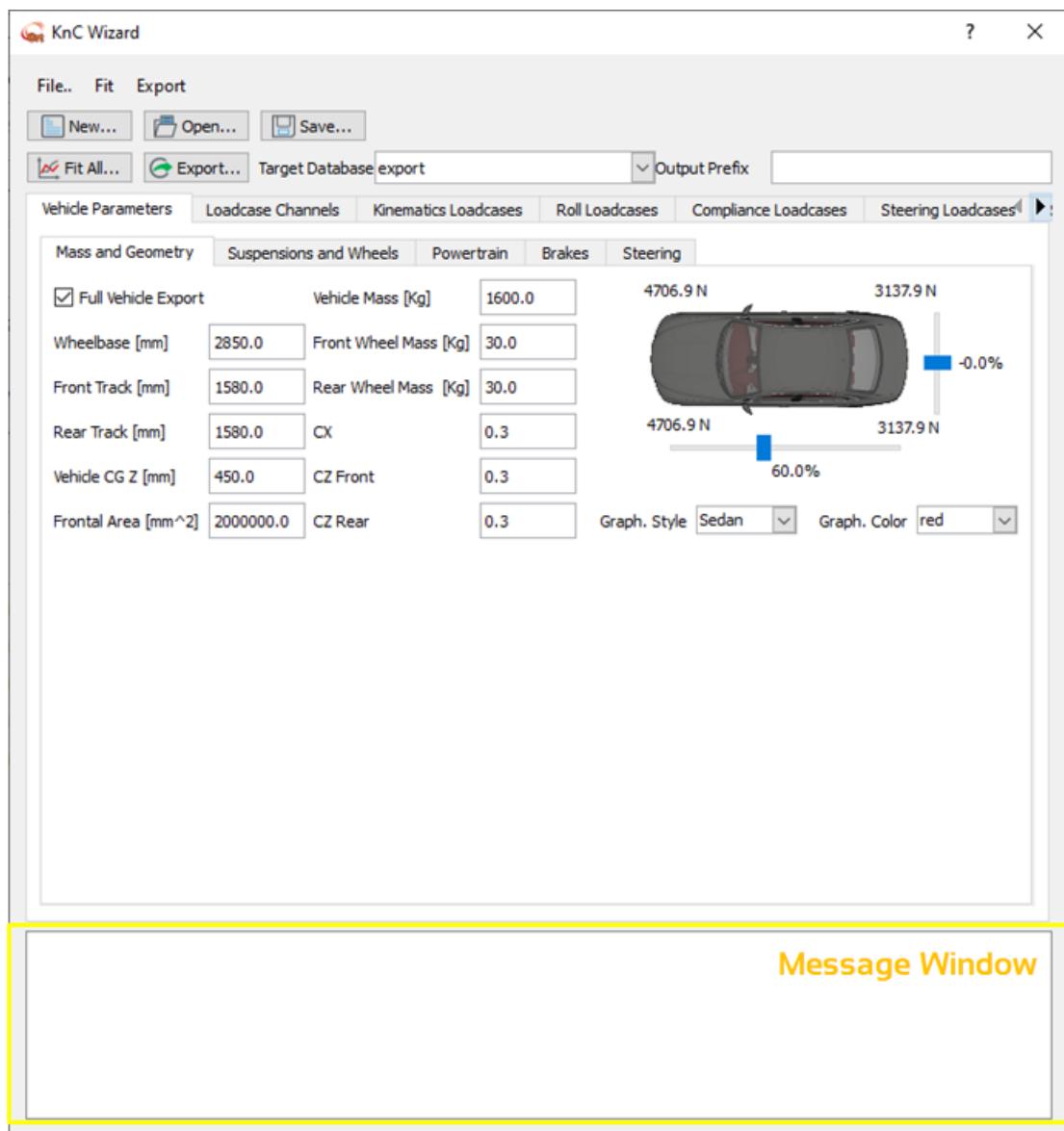
Plotting Area



Plotting area features:

- **X Axis Tree**
hierarchical structure that allow the selection of the channel to be used as independent variable for the plot;
- **Y Axis Tree**
hierarchical structure that allow the selection of the channel to be used as dependent variable for the plot;
- **Plotting Canvas**
two dimensional drawing area into which 2D plots are painted;
- **Plotting Navigation Toolbar**
set of buttons used to navigate through the data set. A description of each of the buttons can be retrieved in [Matplotlib Documentation](#).
- **Add Curve Button**
button populates Plotting Canvas with the selected channels;
- **Clear Plot Button**
button clear the Plotting Canvas.

Message Window



Message window shows messages from the procedure in order to permit a simple check and debugging of the procedure.

The window shows both messages about the operations execute by the procedure and warnings about the erroneous settings of the interface.

Trailer Wizard

Trailer Wizard is a tool allowing the creation of VI-CarRealTime trailer models.

The editor features several sections, each of them related to a particular trailer area; the user can provide as input either a set of parameters to define specific trailer subsystems or supply existing subsystem files.

Depending on geometric inputs trailer graphics will also be created.

The trailer model is stored into one of registered databases and the trailer configuration (containing the set of parameters used to generate the model) can be saved/reloaded to be the starting point for a new trailer model.

[General Parameters & Subsystem Files](#)

[Mass and Aerodynamics](#)

[Suspensions and Tires](#)[Brakes](#)

The following actions are available on the left bottom of the editor:

- **Load Configuration**

button loads xml data needed to populate Trailer Wizard editor. Files are stored in database under vehicle_configs.tbl table;

- **Save Configuration**

button saves to xml data populating Trailer Wizard editor;

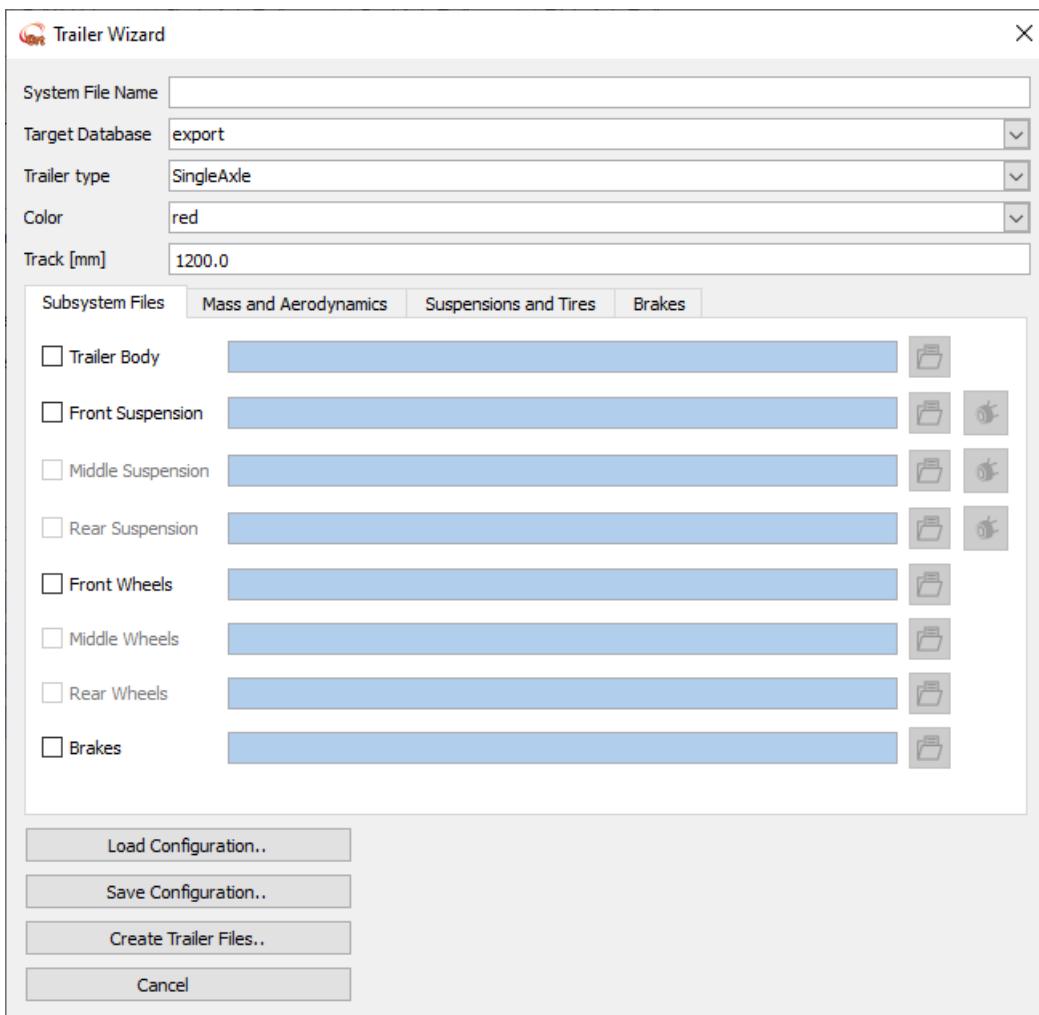
- **Create Trailer Files**

button starts trailer data creation process. Popup dialog will show the successful/unsuccessful status of process. Created trailer models will **not** be automatically loaded into VI-CarRealTime session;

- **Cancel**

button closes the editor.

General Parameters & Subsystem Files



General parameters in Trailer Wizard are:

- **System File Name**

file name for created vehicle model;

- **Target Database**

registered database used to store the model;

- **Trailer type**

Single Axle, Dual Axle or Triple Axle;

- **Color**

chassis graphics color;

- **Track**

trailer track;

- **Subsystem File Names**

name of existing subsystem files to be used in vehicle model (activating a specific subsystem will deactivate related parameter options in related editor tabs).

Mass and Aerodynamics

Subsystem Files	Mass and Aerodynamics	Suspensions and Tires	Brakes
Chassis Mass [Kg]	160.0	Payload Mass [Kg]	30.0
Front Wheel Mass [Kg]	30.0		
Hitch-Ball Axle Offset [mm]	850.0	Hitch-Ball Graphics Offset [mm]	400.0
Vehicle Frontal Area [mm ²]	96001.0	Aero Drag Coefficient	0.3
	X [mm]	Y [mm]	Z [mm]
Chassis CG Loc	-375.0	0.0	460.0
Payload CG Loc	-375.0	0.0	460.0
Hitch-Ball Loc	3100.0	0.0	260.0
Graphics Dimensions	1100.0	1300.0	600.0

Mass and aerodynamics properties are defined through the following data (rear parameters aren't available for SingleAxle trailers)

- **Chassis Mass**

chassis part mass;

- **Payload Mass**

payload part mass;

- **Front/Rear Wheel Mass**

unsprung masses values, Rear Mass;

- **Wheelbase**

distance between front and rear axle;

- **Hitch-Ball Axle Offset**

Distance between the hitch ball and the front axle in [Global Reference](#) X direction;

- **Hitch-Ball Graphics Offset**

Distance between the hitch ball and **graphic** solid representing the trailer sprung mass;

- **Vehicle Frontal Area**

vehicle area needed for aerodynamic force computation;

- **Aero Drag Coefficient**

coefficient for aero drag force;

- **Chassis CG Loc**
chassis CG location;
- **Payload CG Loc**
payload CG location;
- **Hitch-Ball**
Hitch Ball location;
- **Graphic Dimensions**
Graphic dimensions of the solid representing the trailer sprung mass.

Suspensions and Tires

Subsystem Files		Mass and Aerodynamics	Susensions and Tires	Brakes																																			
<table border="1"> <thead> <tr> <th colspan="2">Front Suspension and Tires</th> <th>Middle Suspension and Tires</th> <th>Rear Suspension and Tires</th> </tr> </thead> <tbody> <tr> <td>Front Spring Stiffness [N/mm]</td> <td>150.0</td> <td></td> <td></td> </tr> <tr> <td>Front Spring Preload [N]</td> <td>4000.0</td> <td></td> <td></td> </tr> <tr> <td>Front ARB Stiffness [N/mm]</td> <td>0.0</td> <td></td> <td></td> </tr> <tr> <td>Front Damper Damping [N sec/mm]</td> <td>8.0</td> <td></td> <td></td> </tr> <tr> <td>Front Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> <tr> <td><input type="checkbox"/> Front Twin Wheels Offset</td> <td>300.0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Front Twin Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>					Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires	Front Spring Stiffness [N/mm]	150.0			Front Spring Preload [N]	4000.0			Front ARB Stiffness [N/mm]	0.0			Front Damper Damping [N sec/mm]	8.0			Front Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir				<input type="checkbox"/> Front Twin Wheels Offset	300.0				Front Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir			
Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires																																				
Front Spring Stiffness [N/mm]	150.0																																						
Front Spring Preload [N]	4000.0																																						
Front ARB Stiffness [N/mm]	0.0																																						
Front Damper Damping [N sec/mm]	8.0																																						
Front Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						
<input type="checkbox"/> Front Twin Wheels Offset	300.0																																						
Front Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						
<table border="1"> <thead> <tr> <th colspan="2">Front Suspension and Tires</th> <th>Middle Suspension and Tires</th> <th>Rear Suspension and Tires</th> </tr> </thead> <tbody> <tr> <td>Middle Spring Stiffness [N/mm]</td> <td>150.0</td> <td></td> <td></td> </tr> <tr> <td>Middle Spring Preload [N]</td> <td>4000.0</td> <td></td> <td></td> </tr> <tr> <td>Middle ARB Stiffness [N/mm]</td> <td>25.0</td> <td></td> <td></td> </tr> <tr> <td>Middle Damper Damping [N sec/mm]</td> <td>8.0</td> <td></td> <td></td> </tr> <tr> <td>Middle Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> <tr> <td><input type="checkbox"/> Middle Twin Wheels Offset</td> <td>300.0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Middle Twin Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>					Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires	Middle Spring Stiffness [N/mm]	150.0			Middle Spring Preload [N]	4000.0			Middle ARB Stiffness [N/mm]	25.0			Middle Damper Damping [N sec/mm]	8.0			Middle Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir				<input type="checkbox"/> Middle Twin Wheels Offset	300.0				Middle Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir			
Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires																																				
Middle Spring Stiffness [N/mm]	150.0																																						
Middle Spring Preload [N]	4000.0																																						
Middle ARB Stiffness [N/mm]	25.0																																						
Middle Damper Damping [N sec/mm]	8.0																																						
Middle Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						
<input type="checkbox"/> Middle Twin Wheels Offset	300.0																																						
Middle Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						
<table border="1"> <thead> <tr> <th colspan="2">Front Suspension and Tires</th> <th>Middle Suspension and Tires</th> <th>Rear Suspension and Tires</th> </tr> </thead> <tbody> <tr> <td>Rear Spring Stiffness [N/mm]</td> <td>150.0</td> <td></td> <td></td> </tr> <tr> <td>Rear Spring Preload [N]</td> <td>4000.0</td> <td></td> <td></td> </tr> <tr> <td>Rear ARB Stiffness [N/mm]</td> <td>0.0</td> <td></td> <td></td> </tr> <tr> <td>Rear Damper Damping [N sec/mm]</td> <td>8.0</td> <td></td> <td></td> </tr> <tr> <td>Rear Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> <tr> <td><input type="checkbox"/> Rear Twin Wheels Offset</td> <td>300.0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Rear Twin Tire Property File</td> <td>mdids://carrealtime_shared/tires.tbl/trailer_tire.tir</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>					Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires	Rear Spring Stiffness [N/mm]	150.0			Rear Spring Preload [N]	4000.0			Rear ARB Stiffness [N/mm]	0.0			Rear Damper Damping [N sec/mm]	8.0			Rear Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir				<input type="checkbox"/> Rear Twin Wheels Offset	300.0				Rear Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir			
Front Suspension and Tires		Middle Suspension and Tires	Rear Suspension and Tires																																				
Rear Spring Stiffness [N/mm]	150.0																																						
Rear Spring Preload [N]	4000.0																																						
Rear ARB Stiffness [N/mm]	0.0																																						
Rear Damper Damping [N sec/mm]	8.0																																						
Rear Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						
<input type="checkbox"/> Rear Twin Wheels Offset	300.0																																						
Rear Twin Tire Property File	mdids://carrealtime_shared/tires.tbl/trailer_tire.tir																																						

The parameters needed to define suspension characteristics are:

- **Front/Middle/Rear Spring Stiffness**
spring component linear stiffness (symmetric);

- **Front/Middle/Rear Spring Preload**
spring component preload;
- **Front/Middle/Rear ARB Stiffness**
anti roll bar device ground stiffness;
- **Front/Middle/Rear Damper Damping**
damper component damping coefficient;
- **Front/Middle/Rear Tire Property File**
property file needed for tire force;
- **Front/Middle/Rear Twin Wheels Offset**
Distance of the external wheel center from the internal wheel one;
- **Front/Middle/Rear Tire Property File**
If twin tires are activated (checking the box near Front/Rear Twin Wheels Offset), then select the 2nd tire property file.

Suspension curves and motion ratios are considered flat; user can eventually create or modify them using VI-SuspensionGen.

Brakes

Parameter	Value
Friction Coefficient	0.4
<input checked="" type="checkbox"/> Overrun Brakes	
Overrun Lower Force Threshold [N]	1000.0
Overrun Upper Force Threshold [N]	5000.0
Front Effective Radius [mm]	180.0
Front Maximum Force [N]	2000.0
Middle Effective Radius [mm]	180.0
Middle Maximum Force [N]	2000.0
Rear Effective Radius [mm]	180.0
Rear Maximum Force [N]	2000.0

Brake system parameters are:

- **Friction Coefficient**
- **Overrun Brakes** flag
If checked, the Overrun Lower/Upper Force thresholds can be set.
- **Overrun Lower Force Threshold**
- **Overrun Upper Force Threshold**
- **Front/Rear Piston Effective Radius**
- **Front/Rear Maximum Force**

Property File Obfuscation

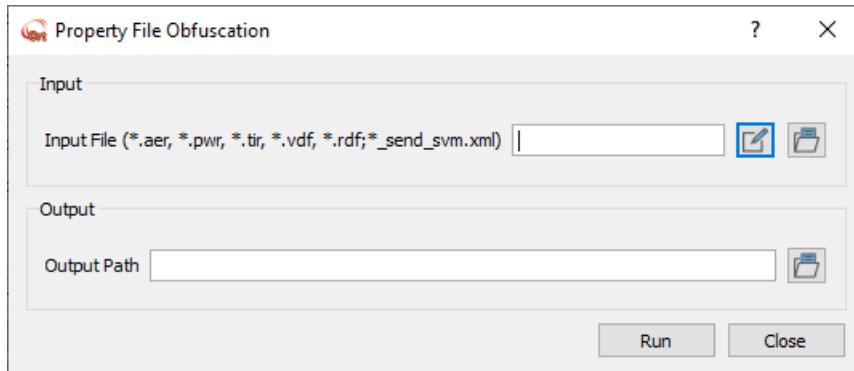
The property files may contain sensitive information that cannot be delivered or distributed without some sort of protection.

VI-CarRealTime supports native data obfuscation algorithms to allow distribution of some of those files to customers or partners, making data unreadable to human eyes, without loosing efficiency or causing additional workload to the user.

The obfuscated files will be unreadable to human eyes, but completely transparent to VI-CarRealTime, which works with them exactly as if they were written in plain text.

The Property File Obfuscation panel allows the user to obfuscate one of the following property files:

- **aer**
Aerodynamic map property file.
- **pwr**
Engine map property file.
- **tir**
Tire property file.
- **vdf**
VI-Driver property file.
- **rdf**
Road Data File property file.
- **send_svm.xml**
xml file which summarizes the whole vehicle model data. The obfuscated send file can also be generated by submitting an event in [files only \(obfuscated\)](#) mode.



The obfuscation editor features a single section where the user is required to provide the following input data:

- **Input File (*.aer, *.pwr, *.tir, *.vdf, *.rdf, *_send.svm.xml)**
Input property file the user wants to obfuscate.
- **Output Path**
Destination folder where the obfuscated property file will be written.

Notes:

1. no readable information is left on the obfuscated files, so their names must define their contents in a clear way.
2. the obfuscation process can be reverted, so make sure to keep the original property file when creating an obfuscated version.
3. any "manual" modification to the obfuscated file, by means of text or hex editor, will corrupt the file causing unpredictable VI-CarRealTime results.

VI-CarRealTime

4. the current core for property file obfuscation supports input property files (for aer, pwr, tir, vdf, rdf, send_svm.xml) not bigger than 18 MB (4 MB for DS1006) except for RDF of type MESH.

Obfuscate send_svm.xml

The obfuscation of send_svm.xml file allows the user to generate a binary version of his vehicle model. Such file, embedding the whole vehicle data and the maneuver event to be simulated, can be shared with customers or partners without revealing any data about the vehicle.

The customer, which might not be allowed to know such sensitive vehicle data, can either use such file to [run the simulation in batch mode](#) or, more interestingly, he can use the send_svm.xml file in Simulink to run a co-simulation: in particular, relying on the whole set of runtime model [input](#) and [output](#), he can integrate his own external component with the obfuscated vehicle model.

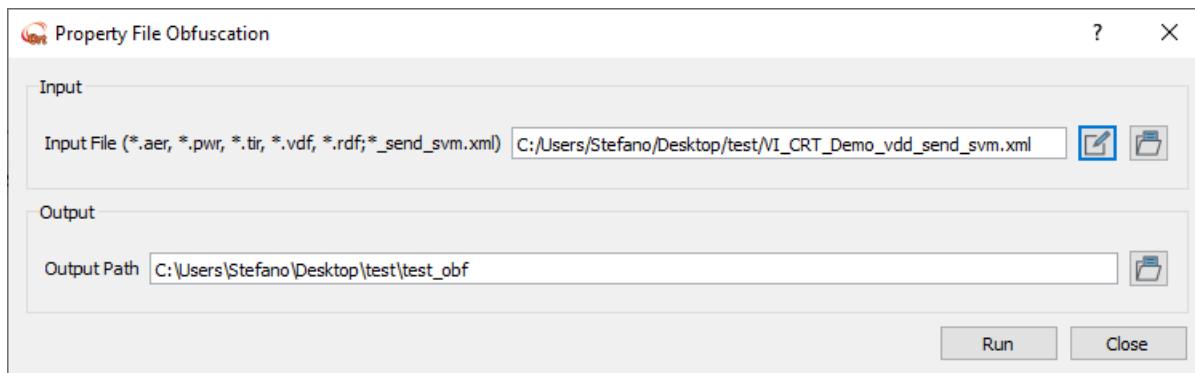
Note: in order for the obfuscated send_svm.xml file to be used, the [vicrt_cdb.cfg](#) file embedding the definition of all the databases references by the send_svm.xml file must be present in the same directory of the obfuscated file.

Here an example about how to generate an obfuscated send_svm.xml file by using [Property File Obfuscation](#) utility and how to use it in Simulink.

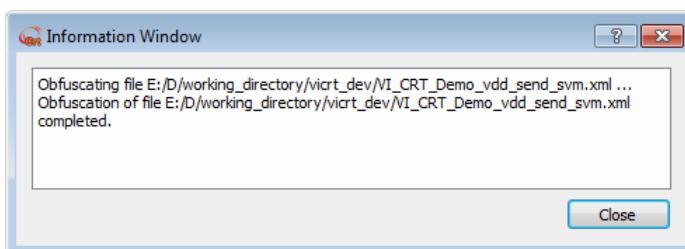
Note: it is also possible to create an obfuscated send_svm.xml file by submitting an event in [files only \(obfuscated\)](#) mode.

Obfuscate send_svm.xml

- Click on *Utilities* menu and select *Property File Obfuscation*
- Browse for an existing send_svm.xml file
- Select an *Output Path* (destination folder of generated obfuscated send_svm.xml file)



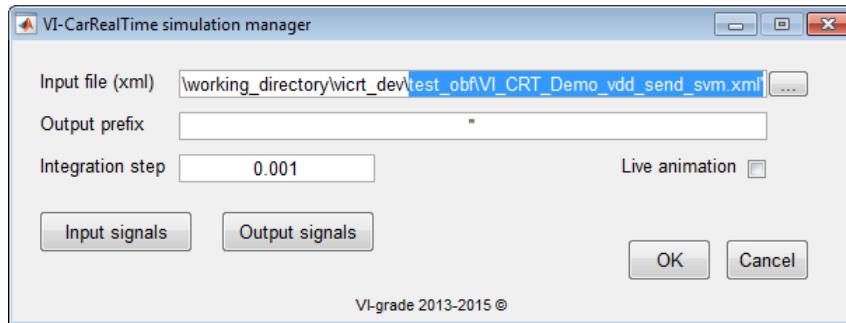
- Click *Run* button to generate the obfuscated version of send_svm.xml file.



Use the obfuscated send_svm.xml in a co-simulation

- open Matlab and select as working directory the same directory where the obfuscated send_svm.xml file is stored.

- assure that also the [*vicrt_cdb.cfg*](#) file with the list of all the databases referenced in the send_svm.xml file is present in such folder.
- type in Matlab Command Window the command `addpath_vicrt_20` to add all the required paths.
- type `vicrt_passive` to open the mdl model.
- select the obfuscated send_svm.xml file as Input File in VI-CarRealTime simulation manager panel.



- run the co-simulation.

At the end of the simulation the result file will be generated in the working directory.

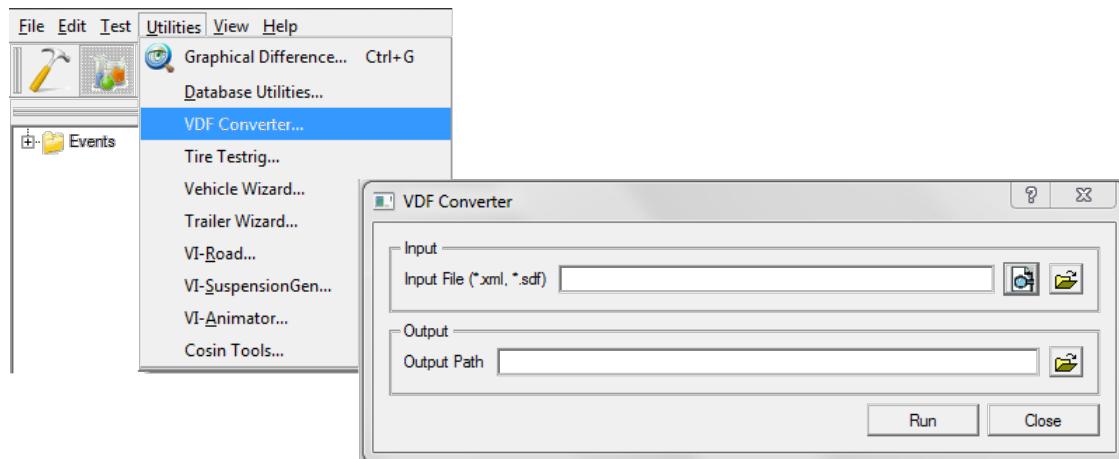
VDF Converter

The VDF Converter, is a tool that allow the conversion of Smart Driver event files (XML/SDF) in the equivalent VI-Driver files (VDF).

The tool is available in VI-CarRealTime under **VDF Converter...** entry of the **Utilities** menu.

Using the displayed window is possible to set an existing **\$filename\$.xml/sdf** SmartDriver file and convert it into the **\$filename\$.vdf** placed in the directory selected in the **Output Path** field.

Is also possible to create an input file using the button of the mask.



In the following sections the conversion rules of SDF/XML files into the VDF are presented, putting in evidence the unsupported feature and the mapping between parameters.

For an easier finding of them, the features are grouped using the SDF block in which they are contained.

- [**MANEUVERS**](#)
- [**STEERING**](#)
- [**THROTTLE**](#)
- [**BRAKING**](#)

- [GEAR](#)
- [CLUTCH](#)
- [MACHINE_CONTROL](#)

Maneuvers

Smart driver **MANEUVERS** block information are entirely converted into **STARTUP** and **MANEUVERS_LIST** blocks of the VDF file.

The only critical entry for this block is the **STATIC_SETUP** key that is converted into the **INITIAL_SETUP** as shown by following table.

STATIC SETUP	INITIAL_SETUP
'NONE'	'OFF'
'NORMAL'	'OFF'
'SETTLE'	'OFF'
'STRAIGHT'	'STRAIGHT'
'SKIDPAD'	UNSUPPORTED

Note: VI-Driver require that SI units to work properly. The XML/SDF values are automatically converted into SI values if possible.

Longitudinal controllers

For each maneuver, the **STEERING**, **THROTTLE**, **BRAKING**, **GEAR** and **CLUTCH** controller sub-blocks are translated into the same blocks of the VDF file.

For each sub-block, the **METHOD** key is set as follows.

METHOD [XML/SDF]	METHOD [VDF]
'MACHINE'	'MACHINE'
'OPEN'	'OPENLOOP'
'USER'	UNSUPPORTED

According to the Smart Driver formulation, for each maneuver the smoothing time of each controller of VI-Driver is set to the **SMOOTING_TIME** specified under the related maneuver block of the SDF/XML file.

Regarding to the **STEERING** controller sub-block, an additional key (**ACTUATOR_TYPE**) is available in XML/SDF file w.r.t. the VDF file.

The following table shows the supported ACTUATOR_TYPE in VI-Driver.

ACTUATOR_TYPE	
'ROTATION'	SUPPORTED
'TRANSLATION'	UNSUPPORTED
'TORQUE'	UNSUPPORTED

Note: VI-Driver require that SI units to work properly. The XML/SDF values are automatically converted into SI values if possible.

Openloop signals

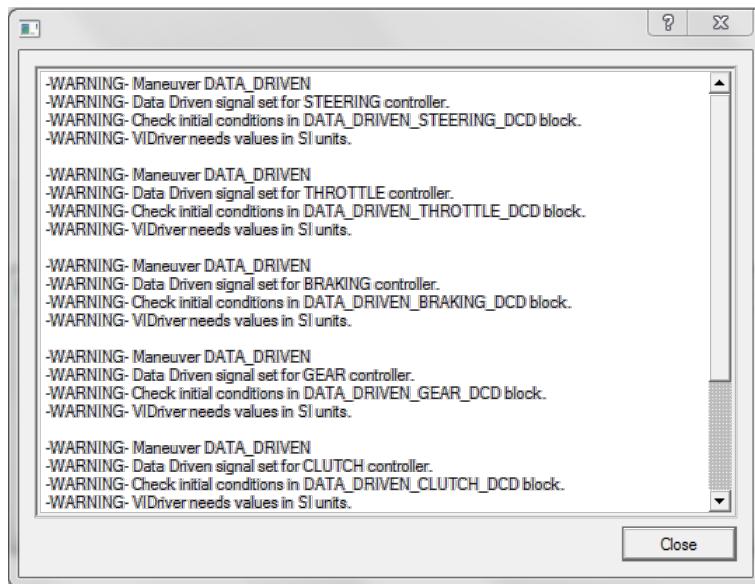
If a controller is set to OPEN, the information related to the signal are stored in a specific block of the VDF file. The name of that block is set as value of the **SIGNAL** key for the controller. Depending to the value of the **CONTROL_TYPE** key, the block is named as follows.

CONTROL_TYPE	SIGNAL
'CONSTANT'	'\$maneuver_name\$_\$controller_name\$_CONSTANT'
'STEP'	'\$maneuver_name\$_\$controller_name\$_STEP'
'RAMP'	'\$maneuver_name\$_\$controller_name\$_RAMP'

'IMPULSE'	'\$maneuver_name\$__\$controller_name\$_IMPULSE'
'SINE'	'\$maneuver_name\$__\$controller_name\$_SINE'
'SWEPT_SINE'	'\$maneuver_name\$__\$controller_name\$_SWEPT_SINE'
'DATA_DRIVEN'	'\$maneuver_name\$__\$controller_name\$_DATA_DRIVEN'
'DATA_MAP'	UNSUPPORTED

If '**DATA_DRIVEN**' control type is set, the tool displays a warning message for each controller that use this type of signal in order to inform that the initial condition have to be manually checked.

Note: VI-Driver require that SI units to work properly. The XML/SDF values are automatically converted into SI values if possible.



Machine controller

For each maneuver, the **MACHINE_CONTROL** sub-block is converted into the **MACHINE** sub-block of the VDF file, translating the **STEERING_CONTROLLER** and the **SPEED_CONTROL** keys into the **PATH** and the **SPEED** key.

The information related to the **STEERING_CONTROLLER** (i.e. path parameters, .drd file, etc.) are stored inside a specific block of the VDF file. The name of that block is set as value of the **PATH** key.

Depending to the type of the **STEERING_CONTROLLER**, the block is named as follows.

STEERING CONTROLLER	PATH
'PATH'	Value of DRIVER PATH if exist, else 'PATH'
'STRAIGHT'	'\$maneuver_name\$ STRAIGHT'
'SKIDPAD'	'\$maneuver_name\$ SKIDPAD'

Regarding to path orientation mode, for initial maneuver and the following ones, the **INITIAL_PATH** and **REF_SYSTEM** of the **MODEL** block are used to set the **REFSYS_TYPE** key of each maneuver. That key is set to **INITIAL_PATH** for the first maneuver and to **REF_SYSTEM** for the other. The information related to the interpolation function are set into the specific path block of the VDF according to the source file setup.

VI-CarRealTime

The information related to the SPEED_CONTROL key are stored in a specific block of the VDF file. The name of that block is set as value of the SPEED key.

Depending to the value of the SPEED_CONTROL key, the block is named as follows.

SPEED_CONTROL	SPEED
'MAP'	Value of SPEED_MAP
'ACCMAP'	Value of SPEED_MAP
'ACCMAP2'	Value of SPEED_MAP
'LATACCMAP'	UNSUPPORTED
'MAINTAIN'	'\$manuver_name\$_MAINTAIN'

In order to reach the same behavior between Smart Driver and VI-Driver, the LONGITUDINAL_SPEED_CONTROL have to be set in the VDF file.

Depending to the value of SPEED_CONTROL key, the LONGITUDINAL_SPEED_CONTROL is set as follows.

SPEED_CONTROL	LONGITUDINAL_SPEED_CONTROL
'MAP'	'LONVEL'
'ACCMAP'	'LONACC'
'ACCMAP2'	'LONVEL'
'MAINTAIN'	'LONVEL'

Note: VI-Driver require that SI units to work properly. The XML/SDF values are automatically converted into SI values if possible.

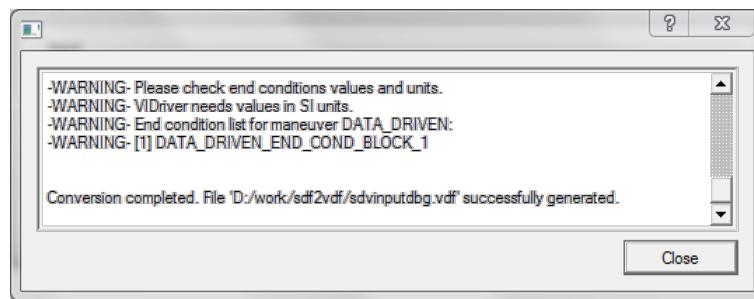
End conditions block

For each maneuver, the end conditions table placed in the END_CONDITIONS sub-block is converted into a list of block names placed in the END_CONDITIONS sub-block of the VDF file.

Each row of the end conditions table is translated into a block named \$manuver_name\$_END_COND_BLOCK_\$row_index\$.

During the conversion process, the tool prints a warning message for each maneuver, reporting the list of end conditions that have to be manually checked in order to set the reference value in SI units.

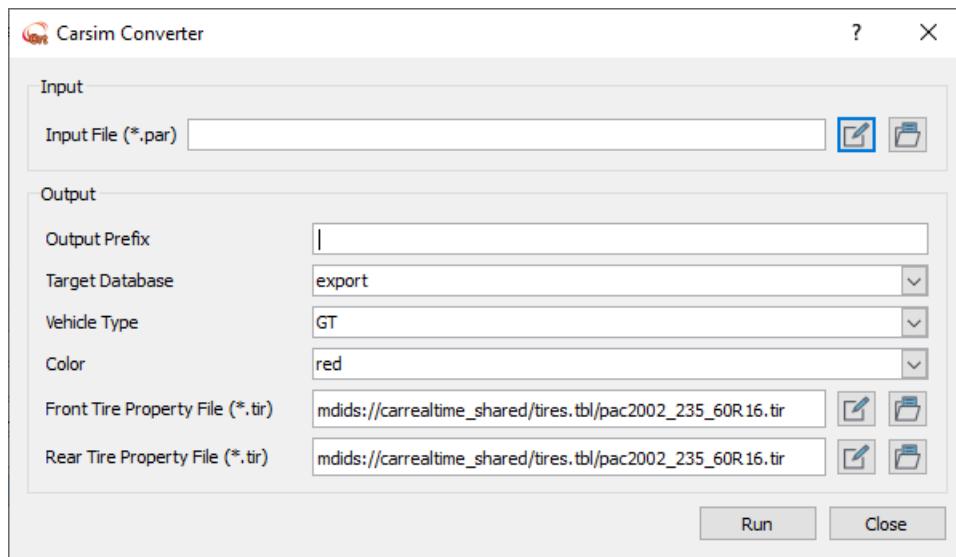
Due to the fact that the end conditions could be user defined, the conversion tool cannot apply an automatic value conversion.

**CarSim converter**

CarSim converter is a tool that allows to convert CarSim model files (*.par) into VI-CarRealTime ones.

CarSim converter requires a special feature key (VI_CarRealTime_CarSim_Converter). If such license key is not defined in the license file, the tool is not available.

The tool is supporting the most common vehicle configuration (Front/Rear independent suspensions, Front independent/Rear Rigid Axle, Rear Twist Beam suspension); due to the differences in tire model supported by CarSim and VI-CarRealTime, tire data are not converted and must be supplied by the user as tire property file.



The converter editor features a single section where the user is required to provide the following input data:

- **Input File (*.par)**
CarSim input file (.par).
- **Output Prefix**
Name for VI-CarRealTime system file.
- **Target Database**
registered database to store converted vehicle model.
- **Vehicle type**
type of vehicle (used to generate chassis graphics); available options are: Compact, GT, Sedan, Open Wheels, SUV, Spider.
- **Color**
chassis graphics color.
- **Front Tire Property File**
tire property file for front wheels.
- **Rear Tire Property File**
tire property file for front wheels.

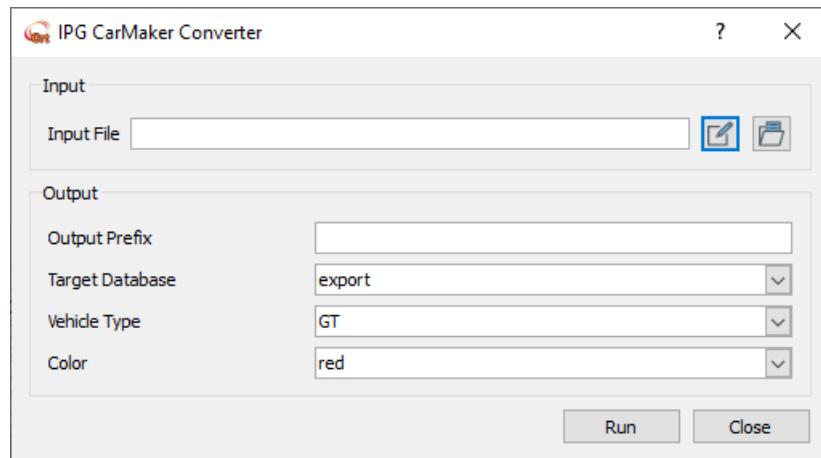
When starting the conversion process the procedure populates the VI-CarRealTime vehicle data and stores them into the selected target database. Accessory data such as vehicle graphics are also created.

Note: Supported CarSim version: 8.1.1 and 9.0

CarMaker converter

CarMaker converter is a tool that allows to convert CarMaker model files into VI-CarRealTime ones. The tool takes as input the CarMaker-Car file and automatically retrieves all the files eventually referenced inside it.

The tool, at the moment, is supporting only a particular configuration of the model (see Supported Features table at the end of this chapter).

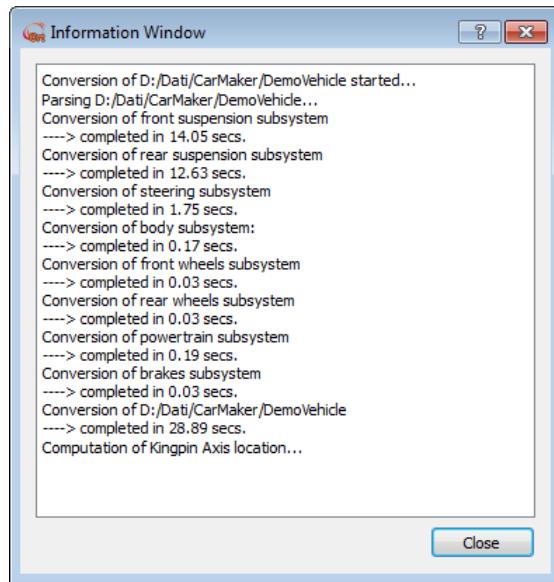


The converter editor features a single section where the user is required to provide the following input data:

- **Input File (CarMaker-Car)**
CarMaker input file (CarMaker-Car).
- **Output Prefix**
Name for VI-CarRealTime system file.
- **Target Database**
registered database to store converted vehicle model.
- **Vehicle Type**
type of vehicle (used to generate chassis graphics); available options are: Compact, GT, Sedan, Open Wheels, SUV, Spider.
- **Color**
chassis graphics color.

When starting the conversion process, the procedure populates the VI-CarRealTime vehicle data and stores them into the selected target database. Accessory data such as vehicle graphics are also created in accordance with the inputs of the editor.

At the end of the procedure, a report appears with the status of the export and, eventually, warning and error messages.



Notes:

Supported Features	
Category	Supported types
Body	Rigid Body, TrimLoads, Elastically mounted Engine
Aerodynamics	Simple aerodynamics (not fully implemented at the moment)
Suspension Kinematics	SKC external files
Suspension Compliance	SKC external files
Suspension Force Elements	Springs, Dampers, Bumpers, Anti-Roll Bar (linear rate only)
Steering	Static Steer Ratio, Mechanical Module (Pfeffer), uniform column, simplified EPS with constant tbar stiffness
Powertrain	Generic and BEV (only DL and Motors on front and rear axles) powertrain
Brake	Pressure Distribution and HydBrakeSystem (not fully implemented at the moment)
Tire	MF52, MF61, TNO_MF61, TameTire

1.3 Model Interface

VI-CarRealTime provides an Input/Output Interface which can be used to exchange data with external tools in order to perform a co-simulation.

It is possible to perform a co-simulation with VI-CarRealTime by using:

- [Matlab/Simulink](#)
- [FMI](#)
- [plug-in dll](#)

During a co-simulation process, a channel (evaluated using an external tool) can be passed to VI-CarRealTime model and the solver will use it during the equations of motion evaluation. By default, unless otherwise stated, all the external input channels are summed to the internal ones. As an example, if an [external](#) spring force is passed

VI-CarRealTime

to VI-CarRealTime model by using [Matlab/Simulink Interface](#), the resulting spring force seen by the solver will be the sum between the external force and the actual force generated by the internal spring element.

A list of the **Input Channels** available in VI-CarRealTime can be found [here](#).

The output channels generated by VI-CarRealTime can be post-processed when the simulation is completed or passed run-time to an external tool.

All the output channels are available run-time and can be used in a co-simulation process, whilst an output channel is written in the output file only if it is defined in the [output map xml](#) file used for the simulation.

A list of the **Output Channels** available in VI-CarRealTime can be found [here](#).

Furthermore VI-CarRealTime offers a set of [solver API](#) that can be used to interface the internal model with external modules.

1.3.1 Input Channels

This section shows all the input channels available in VI-CarRealTime.

For easier finding the desired input, all the variables are grouped in the following categories:

- [Driver Demands](#)
- [Subsystem activation flags](#)
- [ABS](#)
- [TCS](#)
- [Brake system general](#)
- [Road friction](#)
- [Brake system](#)
- [Hook](#)
- [Rear steering actuator](#)
- [Steering](#)
- [Engine](#)
- [Gearbox](#)
- [Clutch](#)
- [Differential Central](#)
- [Differential Front](#)
- [Differential Rear](#)
- [Generic Engine to Clutch](#)
- [Generic Engine to Gearbox](#)
- [Generic Engine to Central Differential](#)
- [Generic Engine to Front Differential](#)
- [Generic Engine to Rear Differential](#)
- [Generic Engine to Front Right Wheel](#)
- [Generic Engine to Front Left Wheel](#)
- [Generic Engine to Rear Right Wheel](#)
- [Generic Engine to Rear Left Wheel](#)

- [Anti-roll bar](#)
- [Aerodynamic](#)
- [Road irregularity](#)
- [Main damper](#)
- [Main spring](#)
- [Main bumpstop](#)
- [Main reboundstop](#)
- [Auxiliary vertical](#)
- [Third element](#)
- [Mono element](#)
- [Testrig Loadcase](#)
- [Tire force](#)
- [Tire moment](#)
- [Wheel driving](#)
- [External body](#)
- [External ground](#)
- [External tracking](#)
- [External suspension](#)
- [Road External](#)
- [Steering Assist](#)

Note: All inputs must be expressed in SI units.

Driver Demands

Here the list of all Driver Demands inputs:

str_swa

"Steering-wheel angle input (hand wheel for driver - radian)"

rack_displacement

"Rack displacement (meter)"

throttle

"Engine throttle demand (normalized from 0 to 100) "

gear

"Gear demand"

clutch

"Clutch demand (normalized from 0 to 1)"

brake

"Brake demand (normalized from 0 to 100)"

target_speed

"Runtime target speed for vi-driver control (m/s). In order to use such input correctly, the vdf event must have a target speed map of type = 'USER_INPUT' (user_long_speed.vdf in VI-CarRealTime shared database)."

target_acc

"Runtime target acceleration for vi-driver control (m/s^2). In order to use such input correctly, the vdf event must have a target speed map of type = 'USER_INPUT' (user_long_acc.vdf in VI-CarRealTime shared database)."

user_maneuver_ID

"Maneuver id switch (if 0 driver will maintain its maneuvers switch logic, otherwise the input will represent the maneuver id that will be performed)"

user_input_steering

"Steering-wheel angle input (hand wheel for driver) [user input driver signal [USER_INPUT_STEERING](#)]"

user_input_throttle

"Engine throttle input [user input driver signal [USER_INPUT_THROTTLE](#)]"

user_input_braking

"Brake input [user input driver signal [USER_INPUT_BRAKING](#)]"

user_input_gear

"Gear demand input [user input driver signal [USER_INPUT_GEAR](#)]"

user_input_clutch

"Clutch demand input [user input driver signal [USER_INPUT_CLUTCH](#)]"

Subsystem Activation Flags

Here the list of all Subsystem Activation Flags inputs:

Internal_Steering_Column_Activity

"Internal steering column activation flag 0-1"

Internal_Aero_Activity

"internal aero activation flag 0-1"

Internal_Brake_System_Activity.L1

"Front left internal brake system activation flag 0-1"

Internal_Brake_System_Activity.R1

"Font right internal brake system activation flag 0-1"

Internal_Brake_System_Activity.L2

"Rear left internal brake system activation flag 0-1"

Internal_Brake_System_Activity.R2

"Rear right internal brake system activation flag 0-1"

Internal_Clutch_Activity

"Internal clutch activation flag 0-1"

Internal_Engine_Activity

"Internal engine engine flag 0-1"

Internal_Motor_to_GearBox_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with gearbox"

Internal_Motor_to_CentralDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with central differential"

Internal_Motor_to_FrontDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with front differential"

Internal_Motor_to_RearDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with rear differential"

Internal_Motor_to_FrontRightWheel_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with front right wheel"

Internal_Motor_to_FrontLeftWheel_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with front left wheel"

Internal_Motor_to_RearRightWheel_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with rear right wheel"

Internal_Motor_to_RearLeftWheel_Activity

"Activation flag which switches off (0) or on (1) the internal Motor connected with rear left wheel"

Internal_Gearbox_Activity

"Internal gearbox activation flag 0-1"

Internal_CentralDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal central differential model"

Internal_FrontDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal front differential model"

Internal_RearDifferential_Activity

"Activation flag which switches off (0) or on (1) the internal rear differential model"

Internal_Engine_Idle_MaxRev_Controls_Activity

"Activation flag which switches off (0) or on (1) the internal engine maximum rpm control system"

Tire_Force_x.L1

"Front left tire Fx activation flag 0-1"

Tire_Force_x.R1

"Front right tire Fx activation flag 0-1"

Tire_Force_x.L2

"Rear left tire Fx activation flag 0-1"

Tire_Force_x.R2

"Rear right tire Fx activation flag 0-1"

Tire_Force_y.L1

"Front left tire Fy activation flag 0-1"

Tire_Force_y.R1

"Front right tire Fy activation flag 0-1"

Tire_Force_y.L2

"Rear left tire Fy activation flag 0-1"

Tire_Force_y.R2

"Rear right tire Fy activation flag 0-1"

Tire_Force_z.L1

"Front left tire Fz activation flag 0-1"

Tire_Force_z.R1

"Front right tire Fz activation flag 0-1"

Tire_Force_z.L2

"Rear left tire Fz activation flag 0-1"

Tire_Force_z.R2

"Rear right tire Fz activation flag 0-1)"

Tire_Moment_x.L1

"Front left tire Mx activation flag 0-1"

Tire_Moment_x.R1

"Front right tire Mx activation flag 0-1"

Tire_Moment_x.L2

"Rear left tire Mxactivation flag 0-1"

Tire_Moment_x.R2

"Rear right tire Mx activation flag 0-1"

Tire_Moment_y.L1

"Front left tire My activation flag 0-1"

Tire_Moment_y.R1

"Front right tire My activation flag 0-1"

Tire_Moment_y.L2

"Rear left tire My activation flag 0-1"

Tire_Moment_y.R2

"Rear right tire My activation flag 0-1"

Tire_Moment_z.L1

"Front left tire Mz activation flag 0-1"

Tire_Moment_z.R1

"Front right tire Mz activation flag 0-1"

Tire_Moment_z.L2

"Rear left tire Mz activation flag 0-1"

Tire_Moment_z.R2

"Rear right tire Mz activation flag 0-1"

Brake System General

Here the list of all Brake System General inputs:

brk_mast_cyl_pres

"Brake master cylinder pressure (pascal)"

brk_bias_front

"Brake Additional Bias Front"

ABS

Here the list of all ABS inputs:

Internal_ABS_Activity

"internal ABS activation flag 0-1"

ABS_threshold

"ABS activation threshold (longitudinal slip). This value is added to the value specified for the threshold in the system properties"

TCS

Here the list of all TCS inputs:

Internal_TCS_Activity

"internal TCS activation flag 0-1"

TCS_threshold

"TCS activation threshold (longitudinal slip). This value is added to the value specified for the threshold in the system properties"

Friction

Here the list of all Friction inputs:

tire_mu.L1

"Front left factor by which tire/ground friction coefficient is scaled. Ignored unless > 1.0e-5"

tire_mu.R1

"Font right factor by which tire/ground friction coefficient is scaled. Ignored unless > 1.0e-5"

tire_mu.L2

"Rear left factor by which tire/ground friction coefficient is scaled. Ignored unless > 1.0e-5"

tire_mu.R2

"Rear right factor by which tire/ground friction coefficient is scaled. Ignored unless > 1.0e-5"

Brake System

Here the list of all Brake System inputs:

brk_mmag_sign.L1

"Front left brake moment magnitude, sign to be adjusted by vehicle model to oppose wheel motion (newton-meter)"

brk_mmag_unsign.L1

VI-CarRealTime

"Front left brake moment magnitude, sign not adjusted by vehicle model (newton-meter)"

brk_pres.L1

"Front left fluid pressure in brake chamber (pascal)"

brk_pres_gain.L1

"Front left brake chamber-pressure gain"

brk_mu.L1

"Front left additional friction coefficient"

brk_mmag_sign.R1

"Font right brake moment magnitude, sign to be adjusted by vehicle model to oppose wheel motion (newton-meter)"

brk_mmag_unsign.R1

"Font right brake moment magnitude, sign not adjusted by vehicle model (newton-meter)"

brk_pres.R1

"Font right fluid pressure in brake chamber (pascal)"

brk_pres_gain.R1

"Font right brake chamber-pressure gain"

brk_mu.R1

"Front right additional friction coefficient"

brk_mmag_sign.L2

"Rear left brake moment magnitude, sign to be adjusted by vehicle model to oppose wheel motion (newton-meter)"

brk_mmag_unsign.L2

"Rear left brake moment magnitude, sign not adjusted by vehicle model (newton-meter)"

brk_pres.L2

"Rear left fluid pressure in brake chamber (pascal)"

brk_pres_gain.L2

"Rear left brake chamber-pressure gain"

brk_mu.L2

"Rear left additional friction coefficient"

brk_mmag_sign.R2

"Rear right brake moment magnitude, sign to be adjusted by vehicle model to oppose wheel motion (newton-meter)"

brk_mmag_unsign.R2

"Rear right brake moment magnitude, sign not adjusted by vehicle model (newton-meter)"

brk_pres.R2

"Rear right fluid pressure in brake chamber (pascal)"

brk_pres_gain.R2

"Rear right brake chamber-pressure gain"

brk_mu.R2

"Rear right additional friction coefficient"

Hook

Here the list of all Hook inputs:

Fx.rear

"Applied force at sprung-mass attach to car in ground X-direction (newton)"

Fy.rear

"Applied force at sprung-mass attach to car in ground Y-direction (newton)"

Fz.rear

"Applied force at sprung-mass attach to car in ground Z-direction (newton)"

Mx.rear

"Applied moment on sprung mass in ground X-direction (newton-meter)"

My.rear

"Applied moment on sprung mass in ground Y-direction (newton-meter)"

Mz.rear

"Applied moment on sprung mass in ground Z-direction (newton-meter)"

Rear Steering Actuator Disp

Here the list of all Rear Steering Actuator Disp inputs:

str_swa.left

"Rear Left steering input, used as second independent variable to lookup rear kinematics curve (meter)"

str_swa.right

"Rear Right steering input, used as second independent variable to lookup rear kinematics curve (meter)"

Steering

Here the list of all Steering inputs:

steer_at_spindle.L1

"Front left wheel steer at spindle (radian)"

steer_at_spindle.R1

"Front right wheel steer at spindle (radian)"

steer_at_spindle.L2

"Rear left wheel steer at spindle (radian)"

steer_at_spindle.R2

"Rear right wheel steer at spindle (radian)"

Internal_Hand_Wheel_Torque_Computation

"Input to deactivate the internal steering torque output calculation (values: 1 -> standard internal output, 0->output coming from input channel Hand_Wheel_Torque.)"

Hand_Wheel_Torque

"Input to overwrite internal hand steering torque output.In case of advanced steering, the input will be neglected."

Engine

Here the list of all Engine inputs:

engine_omega

"Engine omega from external powertrain (radian/second)"

engine_max_trq

"Maximum engine torque from external powertrain (newton-meter)"

engine_min_trq

"Minimum engine torque from external powertrain (newton-meter)"

engine_trq

"Actual engine torque from external powertrain (newton-meter)"

clutch_inputplate_torque

" (newton-meter)"

non_termic_engine_trq

"input torque of a further engine in parallel with the thermic internal engine (newton-meter)"

efficiency

"engine efficiency [0-1]"

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

control_mode

"choose how to control the engine, between torque (control_mode=0) and throttle (control_mode=1)"

GearBox

Here the list of all Gearbox inputs:

efficiency

"engine efficiency [0-1]"

transmission_ratio

"total transmission ratio (primary * gear)"

diff_central_inputshaft_torque

"central differential input shaft torque (newton-meter)"

clutch_outputshaft_angle

"clutch output shaft rotation angle (radian)"

clutch_outputshaft_omega

"clutch output shaft angular velocity (radian/second**2)"

clutch_outputshaft_alpha

"clutch output shaft angular acceleration (radian/second**2)"

automatic_gearbox_shifting_mode

"switch between comfort and sport gear shifting mode. 0=comfort, 1=sport"

Clutch

Here the list of all Clutch inputs:

clutch_trq

"Actual clutch torque from external powertrain (newton-meter)"

gearbox_inputshaft_torque

"gearbox input shaft torque (newton-meter)"

inputplate_angle

"clutch input plate rotation angle (radian)"

inputplate_omega

"clutch input plate angular velocity (radian/second**2)"

inputplate_alpha

"clutch input plate angular acceleration (radian/second**2)"

efficiency

"engine efficiency [0-1]"

Differential Central

Here the list of all Differential Central inputs:

efficiency

"central differential efficiency [0-1]"

diff_front_inputshaft_torque

"front input shaft torque (newton-meter)"

diff_rear_inputshaft_torque

"rear input shaft torque (newton-meter)"

gearbox_outputshaft_angle

"gearbox output shaft rotation angle (radian)"

gearbox_outputshaft_omega

"gearbox output shaft angular velocity (radian/second**2)"

gearbox_outputshaft_alpha

"gearbox output shaft angular acceleration (radian/second**2)"

Differential Front

Here the list of all Differential Front inputs:

efficiency

"front differential efficiency [0-1]"

wheel_front_left_torque

"front left half shaft torque (newton-meter)"

wheel_front_right_torque

"front right half shaft torque (newton-meter)"

diff_central_front_outputshaft_angle

VI-CarRealTime

"central differential front output shaft angle (radian)"

diff_central_front_outputshaft_omega

"central differential front output shaft angular velocity (radian/second)"

diff_central_front_outputshaft_alpha

"central differential front output shaft angular acceleration (radian/second**2)"

Differential Rear

Here the list of all Differential Rear inputs:

efficiency

"rear differential efficiency [0-1]"

wheel_rear_left_torque

"rear left half shaft torque (newton-meter)"

wheel_rear_right_torque

"rear right half shaft torque (newton-meter)"

diff_central_rear_outputshaft_angle

"central differential rear output shaft angle (radian)"

diff_central_rear_outputshaft_omega

"rear differential front output shaft angular velocity (radian/second)"

diff_central_rear_outputshaft_alpha

"rear differential front output shaft angular acceleration (radian/second**2)"

Generic Engine to Clutch

Here the list of all Generic Engine to Clutch inputs:

throttle_scaling

"throttle scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"throttle demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

Generic Engine to Gearbox

Here the list of all Generic Engine to Gearbox inputs:

throttle_scaling

"throttle scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"throttle demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

Generic Engine to Central Differential

Here the list of all Generic Engine to Central Differential inputs:

throttle_scaling

"throttle scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"throttle demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Generic Engine to Front Differential

Here the list of all Generic Engine to Front Differential inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Generic Engine to Rear Differential

Here the list of all Generic Engine to Rear Differential inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details."

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Generic Engine to Front Right Wheel

Here the list of all Generic Engine to Front Right Wheel inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details."

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Generic Engine to Front Left Wheel

Here the list of all Generic Engine to Front Left Wheel inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details."

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."

Generic Engine to Rear Right Wheel

Here the list of all Generic Engine to Rear Right Wheel inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Generic Engine to Rear Left Wheel

Here the list of all Generic Engine to Rear Left Wheel inputs:

throttle_scaling

"[throttle](#) scaling factor [0-1]. Used only when control_mode is equal to 1."

throttle_offset

"[throttle](#) demand offset [0-1]. Used only when control_mode is equal to 1."

efficiency

"engine efficiency [0-1]"

engine_torque

"actual engine torque from generic engine (newton-meter)"

engine_max_torque

"maximum engine torque from generic engine (newton-meter)"

engine_min_torque

"minimum engine torque from generic engine (newton-meter)"

control_mode

"choose how to control the motor, between torque (control_mode=0) and throttle (control_mode=1)"

regeneration_factor

"regeneration coefficient [0-1] of the electric engine. Please look at [regenerative braking factor](#) for the details.

transmission_ratio

"engine runtime transmission ratio (default value 1.0)"

reverse_mode

"reverse mode (standard run = 0 backward run = 1)."'

Antirollbar

Here the list of all Antirollbar inputs:

activity_flag.front

"Internal front antiroll bar activation flag 0-1"

activity_flag.rear

"Internal rear antiroll bar activation flag 0-1"

force.front

"Suspension front antiroll force (positive left) (newton)"

force.rear

"Suspension rear antiroll force (positive left) (newton)"

Aerodynamic

Here the list of all Aerodynamic inputs:

fx_front

"Aerodynamic force component in tracking frame x direction applied at front aero application point (newton)"

fy_front

"Aerodynamic force component in tracking frame y direction applied at front aero application point (newton)"

fz_front

"Aerodynamic force component in tracking frame z direction applied at front aero application point (newton)"

fx_rear

"Aerodynamic force component in tracking frame x direction applied at rear aero application point (newton)"

fy_rear

"Aerodynamic force component in tracking frame y direction applied at rear aero application point (newton)"

fz_rear

"Aerodynamic force component in tracking frame z direction applied at rear aero application point (newton)"

fx_center

"Aerodynamic force component in tracking frame x direction applied at center aero application point (newton)"

fy_center

"Aerodynamic force component in tracking frame y direction applied at center aero application point (newton)"

fz_center

"Aerodynamic force component in tracking frame z direction applied at center aero application point (newton)"

mx

"Aerodynamic torque component in tracking frame x direction (newton-meter)"

my

"Aerodynamic torque component in tracking frame y direction (newton-meter)"

mz

"Aerodynamic torque component in tracking frame z direction (newton-meter)"

Note: tracking frame is [Road Reference Frame](#) (default) or [Chassis Reference Frame](#) depending on the value of [use_road_plane_direction](#) flag.

Irregularity

Here the list of all Irregularity inputs:

road.L1

"Front left wheel road vertical perturbation (meter)"

road.R1

"Front right wheel road vertical perturbation (meter)"

road.L2

"Rear left wheel road vertical perturbation (meter)"

road.R2

"Rear right wheel road vertical perturbation (meter)"

Main Damper

Here the list of all Main Damper inputs:

force.L1

"Front left suspension damper force (at damper) (newton)"

force.R1

"Front right suspension damper force (at damper) (newton)"

force.L2

"Rear left suspension damper force (at damper) (newton)"

force.R2

"Rear right suspension damper force (at damper) (newton)"

activation_flag.L1

"Front left suspension damper force (at damper) internal activity flag"

activation_flag.R1

"Front right suspension damper force (at damper) internal activity flag"

activation_flag.L2

"Rear left suspension damper force (at damper) internal activity flag"

activation_flag.R2

"Rear right suspension damper force (at damper) internal activity flag"

Main Spring

Here the list of all Main Spring inputs:

force.L1

"Front left suspension spring force (at spring) (newton)"

force.R1

"Front right suspension spring force (at spring) (newton)"

force.L2

"Rear left suspension spring force (at spring) (newton)"

force.R2

"Rear right suspension spring force (at spring) (newton)"

activity_flag.L1

"Front left suspension spring force (at spring) internal activity flag"

activity_flag.R1

"Front right suspension spring force (at spring) internal activity flag"

activity_flag.L2

"Rear left suspension spring force (at spring) internal activity flag"

activity_flag.R2

"Rear right suspension spring force (at spring) internal activity flag"

Main Bumpstop

Here the list of all Main Bumpstop inputs:

force.L1

"Front left suspension jounce-bumper force (at bumper) (newton)"

force.R1

"Front right suspension jounce-bumper force (at bumper) (newton)"

force.L2

"Rear left suspension jounce-bumper force (at bumper) (newton)"

force.R2

"Rear right suspension jounce-bumper force (at bumper) (newton)"

activity_flag.L1

"Front left suspension jounce-bumper force (at bumper) internal activity flag"

activity_flag.R1

"Front right suspension jounce-bumper force (at bumper) internal activity flag"

activity_flag.L2

"Rear left suspension jounce-bumper force (at bumper) internal activity flag"

activity_flag.R2

"Rear right suspension jounce-bumper force (at bumper) internal activity flag"

Main Reboundstop

Here the list of all Main Reboundstop inputs:

force.L1

"Front left suspension rebound-bumper force (at bumper) (newton)"

force.R1

"Front right suspension rebound-bumper force (at bumper) (newton)"

force.L2

"Rear left suspension rebound-bumper force (at bumper) (newton)"

force.R2

"Rear right suspension rebound-bumper force (at bumper) (newton)"

activity_flag.L1

"Front left suspension rebound-bumper force (at bumper) internal activity flag"

activity_flag.R1

"Front right suspension rebound-bumper force (at bumper) internal activity flag"

activity_flag.L2

"Rear left suspension rebound-bumper force (at bumper) internal activity flag"

activity_flag.R2

"Rear right suspension rebound-bumper force (at bumper) internal activity flag"

Auxiliary Vertical

Here the list of all Auxiliary Vertical inputs:

force.L1

"Front left suspension auxiliary spring force (at wheel) (newton)"

force.R1

"Front right suspension auxiliary spring force (at wheel) (newton)"

force.L2

"Rear left suspension auxiliary spring force (at wheel) (newton)"

force.R2

"Rear right suspension auxiliary spring force (at wheel) (newton)"

activity_flag.L1

"Front left suspension auxiliary spring force (at wheel) internal activity flag"

activity_flag.R1

"Front right suspension auxiliary spring force (at wheel) internal activity flag"

activity_flag.L2

"Rear left suspension auxiliary spring force (at wheel) internal activity flag"

activity_flag.R2

"Rear right suspension auxiliary spring force (at wheel) internal activity flag"

Main Inerter

Here the list of all Main Inerter inputs:

force.L1

"Front left suspension inerter force (at inerter) (newton)"

force.R1

"Front right suspension inerter force (at inerter) (newton)"

force.L2

"Rear left suspension inerter force (at inerter) (newton)"

force.R2

"Rear right suspension inerter force (at inerter) (newton)"

activation_flag.L1

"Front left suspension inerter force (at inerter) internal activity flag"

activation_flag.R1

"Front right suspension inerter force (at inerter) internal activity flag"

activation_flag.L2

"Rear left suspension inerter force (at inerter) internal activity flag"

activation_flag.R2

"Rear right suspension inerter force (at inerter) internal activity flag"

Third Element

Here the list of all Third Element inputs:

Damper_force.front

"Front third damper force (at damper) (newton)"

Damper_force.rear

"Rear third damper force (at damper) (newton)"

Damper_activity_flag.front

"Front third damper force (at damper) activity flag"

Damper_activity_flag.rear

"Rear third damper force (at damper) activity flag"

Spring.front

"Front third spring force (at spring) (newton)"

Spring.rear

"Rear third spring force (at spring) (newton)"

Spring_activity_flag.front

"Front third spring force (at spring) activity flag"

Spring_activity_flag.rear

"Rear third spring force (at spring) activity flag"

Mono Element

Here the list of all Mono Element inputs:

Damper.front

"Front single damper force (at damper) (newton)"

Damper.rear

"Rear single damper force (at damper) (newton)"

Damper_activity_flag.front

"Front single damper force (at damper) activity flag"

Damper_activity_flag.rear

"Rear single damper force (at damper) activity flag"

Inerter.front

"Front inerter force (at inerter) (newton)"

Inerter.rear

"Rear inerter force (at inerter) (newton)"

Inerter_activity_flag.front

"Front inerter force (at inerter) activity flag"

Inerter_activity_flag.rear

"Rear inerter force (at inerter) activity flag"

Spring.front

"Front single spring force (at spring) (newton)"

Spring.rear

"Rear single spring force (at spring) (newton)"

Spring_activity_flag.front

"Front single spring force (at spring) activity flag"

Spring_activity_flag.rear

"Rear single spring force (at spring) activity flag"

Bumpstop_Spring.front

"Front single spring bumpstop force (at bumpstop) (newton)"

Bumpstop_Spring.rear

"Rear single spring bumpstop force (at bumpstop) (newton)"

Bumpstop_Spring_activity_flag.front

"Front single spring bumpstop force (at bumpstop) activity flag"

Bumpstop_Spring_activity_flag.rear

"Rear single spring bumpstop force (at bumpstop) activity flag"

Reboundstop_Spring.front

"Front single reboundstop force (at reboundstop) (newton)"

Reboundstop_Spring.rear

"Rear single reboundstop force (at reboundstop) (newton)"

Reboundstop_Spring_activity_flag.front

"Front single reboundstop force (at reboundstop) activity flag"

Reboundstop_Spring_activity_flag.rear

"Rear single reboundstop force (at reboundstop) activity flag"

Bumpstop_Damper.front

"Front single damper bumpstop force (at bumpstop) (newton)"

Bumpstop_Damper.rear

"Rear single damper bumpstop force (at bumpstop) (newton)"

Bumpstop_Damper_activity_flag.front

"Front single damper bumpstop force (at bumpstop) activity flag"

Bumpstop_Damper_activity_flag.rear

"Rear single damper bumpstop force (at bumpstop) activity flag"

Reboundstop_Damper.front

"Front single reboundstop force (at reboundstop) (newton)"

Reboundstop_Damper.rear

"Rear single reboundstop force (at reboundstop) (newton)"

Reboundstop_Damper_activity_flag.front

"Front single reboundstop force (at reboundstop) activity flag"

Reboundstop_Damper_activity_flag.rear

"Rear single reboundstop force (at reboundstop) activity flag"

Roll_Damper.front

"Front roll damper element force (at damper) (newton)"

Roll_Damper.rear

"Rear roll damper element force (at damper) (newton)"

Roll_Damper_activity_flag.front

"Front roll damper internal force activity flag"

Roll_Damper_activity_flag.rear

"Rear roll damper internal activity flag"

Testrig Loadcase

The following inputs are specific of the Sevenpostrig events and are generated only when [Body Fix Setup Mode](#) is set to *Fix Body at Design* or *Fix Body at Setup*.

These inputs can be used to submit specific dynamic loadcases, providing forces and/or torques inputs at the tire contact patch location.

Here the list of all Testrig Loadcase inputs:

Tire_Force_wc_x.L1

"Longitudinal front left input force applied at hub carrier (newton) in [ISO-C](#) reference system."

Tire_Force_wc_x.L2

"Longitudinal rear left input force applied at hub carrier (newton) in [ISO-C](#) reference system."

Tire_Force_wc_x.R1

"Longitudinal front right input force applied at hub carrier (newton) in [ISO-C](#) reference system."

Tire_Force_wc_x.R2

"Longitudinal rear right input force applied at hub carrier (newton) in [ISO-C](#) reference system."

Tire_Force_x.L1

"Longitudinal front left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_x.R1

"Longitudinal front right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_x.L2

"Longitudinal rear left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_x.R2

"Longitudinal rear right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_y.L1

"Lateral front left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_y.R1

"Lateral front right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_y.L2

"Lateral rear left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_y.R2

"Lateral rear right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_z.L1

"Vertical front left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_z.R1

"Vertical front right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_z.L2

"Vertical rear left input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Force_z.R2

"Vertical rear right input force applied at hub carrier (newton) in [ISO-W](#) reference system."

Note: Tire_Force_z inputs are used only when the related [Deactivate_Verical_Displacement_Control](#) input flag is set to 0.

Tire_Moment_x.L1

"Overturning front left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_x.R1

"Overturning front right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_x.L2

"Overturning rear left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_x.R2

"Overturning rear right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_y.L1

"Rolling resistance front left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_y.R1

"Rolling resistance front right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_y.L2

"Rolling resistance rear left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_y.R2

"Rolling resistance rear right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_z.L1

"Aligning front left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_z.R1

"Aligning front right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_z.L2

"Aligning rear left input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Tire_Moment_z.R2

"Aligning rear right input moment applied at hub carrier (newton) in [ISO-W](#) reference system."

Deactivate_Vertical_Displacement_Control.L1

"When the flag is set to 0, the front left hub vertical displacement controller is deactivated (internal VI-CarRealTime [PID](#) doesn't work) and it is possible to provide an external *Tire_Force_z.L1* input runtime."

Deactivate_Vertical_Displacement_Control.R1

"When the flag is set to 0, the front right hub vertical displacement controller is deactivated (internal VI-CarRealTime [PID](#) doesn't work) and it is possible to provide an external *Tire_Force_z.R1* input runtime."

Deactivate_Vertical_Displacement_Control.L2

"When the flag is set to 0, the rear left hub vertical displacement controller is deactivated (internal VI-CarRealTime [PID](#) doesn't work) and it is possible to provide an external *Tire_Force_z.L2* input runtime."

Deactivate_Vertical_Displacement_Control.R2

"When the flag is set to 0, the rear right hub vertical displacement controller is deactivated (internal VI-CarRealTime [PID](#) doesn't work) and it is possible to provide an external *Tire_Force_z.R2* input runtime."

Tire Force

Here the list of all Tire Force inputs:

Fx.L1

"Front left tire Fx force at hub carrier (newton) in [ISO-C](#) reference system."

Fy.L1

"Front left tire Fy force at hub carrier (newton) in [ISO-C](#) reference system."

Fz.L1

"Front left tire Fz force at hub carrier (newton) in [ISO-C](#) reference system."

Fx.R1

"Front right tire Fx force at hub carrier (newton) in [ISO-C](#) reference system."

Fy.R1

"Front right tire Fy force at hub carrier (newton) in [ISO-C](#) reference system."

Fz.R1

"Front right tire Fz force at hub carrier (newton) in [ISO-C](#) reference system."

Fx.L2

"Rear left tire Fx force at hub carrier (newton) in [ISO-C](#) reference system."

Fy.L2

"Rear left tire Fy force at hub carrier (newton) in [ISO-C](#) reference system."

Fz.L2

"Rear left tire Fz force at hub carrier (newton) in [ISO-C](#) reference system."

Fx.R2

"Rear right tire Fx force at hub carrier (newton) in [ISO-C](#) reference system."

Fy.R2

"Rear right tire Fy force at hub carrier (newton) in [ISO-C](#) reference system."

Fz.R2

"Rear right tire Fz force at hub carrier (newton) in [ISO-C](#) reference system."

Tire Moment

Here the list of all Tire Moment inputs:

Mx.L1

"Front left tire Mx moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

My.L1

"Front left tire My moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mz.L1

"Front left tire Mz moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mx.R1

"Front right tire Mx moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

My.R1

"Front right tire My moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mz.R1

"Front right tire Mz moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mx.L2

"Rear left tire Mx moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

My.L2

"Rear left tire My moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mz.L2

"Rear left tire Mz moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mx.R2

"Rear right tire Mx moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

My.R2

"Rear right tire My moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Mz.R2

"Rear right tire Mz moment at hub carrier (newton-meter) in [ISO-C](#) reference system."

Wheel Driving

Here the list of all Wheel Driving inputs:

moment.L1

"Front left wheel drive moment (newton-meter)"

moment.R1

"Front right wheel drive moment (newton-meter)"

moment.L2

"Rear left wheel drive moment (newton-meter)"

moment.R2

"Rear right wheel drive moment (newton-meter)"

Ext Body

Here the list of all Ext Body inputs:

force.X

"Applied force at sprung-mass CM in body X-direction (newton)"

force.Y

"Applied force at sprung-mass CM in body Y-direction (newton)"

force.Z

"Applied force at sprung-mass CM in body Z-direction (newton)"

moment.X

"Applied moment on sprung mass in body X-direction (newton-meter)"

moment.Y

"Applied moment on sprung mass in body Y-direction (newton-meter)"

moment.Z

"Applied moment on sprung mass in body Z-direction (newton-meter)"

Ext Ground

Here the list of all Ext Ground inputs:

force.X

"Applied force at sprung-mass CM in global X-direction (newton)"

force.Y

"Applied force at sprung-mass CM in global Y-direction (newton)"

force.Z

"Applied force at sprung-mass CM in global Z-direction (newton)"

moment.X

"Applied moment on sprung mass in global X-direction (newton-meter)"

moment.Y

"Applied moment on sprung mass in global Y-direction (newton-meter)"

moment.Z

"Applied moment on sprung mass in global Z-direction (newton-meter)"

Ext Tracking

Here the list of all Ext Tracking inputs:

force.X

"Applied force at sprung-mass CM in vehicle-tracking frame X-direction (newton)"

force.Y

"Applied force at sprung-mass CM in vehicle-tracking frame Y-direction (newton)"

moment.X

"Applied moment on sprung mass in vehicle-tracking frame X-direction (newton-meter)"

moment.Y

"Applied moment on sprung mass in vehicle-tracking frame Y-direction (newton-meter)"

External Suspension

The following list of inputs is enabled through the [external_suspension](#) flags available in the [Model Properties](#) parameter tree. For each of the suspension characteristic (track, base, side view angle, toe, camber), there is a flag for the activation/deactivation of the internal computation, plus a comprehensive set of inputs that allow to replace the internal suspension elasto-kinematic (as a whole or component by component) with an external model. Each of this input is additive, so if the internal flag is active, the value provided by the input is summed to the value internally computed.

Suspension.[L1/L2/R1/R2].baseKinematicInternalFlag

"flag to activate/deactivate the internal computation of suspension base kinematics [0-1]"

Suspension.[L1/L2/R1/R2].baseDisplacement

"externally computed base variation"

Suspension.[L1/L2/R1/R2].baseVelocity

"externally computed first time derivative of base variation"

Suspension.[L1/L2/R1/R2].baseAcceleration

"externally computed second time derivative of base variation"

Suspension.[L1/L2/R1/R2].baseDerivativeWrtSwa

"externally computed derivative of base variation with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].baseDerivativeWrtJside

"externally computed derivative of base variation, with respect to jounce (this side)"

Suspension.[L1/L2/R1/R2].baseDerivativeWrtJopp

"externally computed derivative of base variation, with respect to jounce (opposite side)"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtSwaSwa

"externally computed second derivative of base variation, with respect to swa / swa"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtSwaJside

"externally computed second derivative of base variation, with respect to swa / jounce"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtSwaJopp

"externally computed second derivative of base variation, with respect to swa / opposite jounce"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtJsideJside

"externally computed second derivative of base variation, with respect to jounce / jounce"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtJsideJopp

"externally computed second derivative of base variation, with respect to jounce / jounce opposite"

Suspension.[L1/L2/R1/R2].baseSecondDerivativeWrtJoppJopp

"externally computed second derivative of base variation, with respect to jounce opposite / jounce opposite"

Suspension.[L1/L2/R1/R2].baseComplianceInternalFlag

"flag to activate/deactivate the internal computation of suspension base compliance [0-1]"

Suspension.[L1/L2/R1/R2].baseCompliance

"externally computed base variation due to compliance"

Suspension.[L1/L2/R1/R2].baseComplianceVelocity

"externally computed first time derivative of base variation due to compliance"

Suspension.[L1/L2/R1/R2].baseComplianceAcceleration

"externally computed second time derivative of base variation due to compliance"

Suspension.[L1/L2/R1/R2].baseComplianceDerivativeWrtSwa

"externally computed first derivative of base variation due to compliance with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].baseComplianceDerivativeWrtJside

"externally computed first derivative of base variation due to compliance with respect to jounce"

Suspension.[L1/L2/R1/R2].baseComplianceDerivativeWrtJopp

"externally computed first derivative of base variation due to compliance with respect to jounce opposite"

Suspension.[L1/L2/R1/R2].trackKinematicInternalFlag

"flag to activate/deactivate the internal computation of suspension track kinematics [0-1]"

Suspension.[L1/L2/R1/R2].trackDisplacement

"externally computed track variation"

Suspension.[L1/L2/R1/R2].trackVelocity

"externally computed first time derivative of track variation"

Suspension.[L1/L2/R1/R2].trackAcceleration

"externally computed second time derivative of track variation"

Suspension.[L1/L2/R1/R2].trackDerivativeWrtSwa

"externally computed derivative of track variation with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].trackDerivativeWrtJside

"externally computed derivative of track variation, with respect to jounce (this side)"

Suspension.[L1/L2/R1/R2].trackDerivativeWrtJopp

"externally computed derivative of track variation, with respect to jounce (opposite side)"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtSwaSwa

"externally computed second derivative of track variation, with respect to swa / swa"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtSwaJside

"externally computed second derivative of track variation, with respect to swa / jounce"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtSwaJopp

"externally computed second derivative of track variation, with respect to swa / opposite jounce"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtJsideJside

"externally computed second derivative of track variation, with respect to jounce / jounce"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtJsideJopp

"externally computed second derivative of track variation, with respect to jounce / jounce opposite"

Suspension.[L1/L2/R1/R2].trackSecondDerivativeWrtJoppJopp

"externally computed second derivative of track variation, with respect to jounce opposite / jounce opposite"

Suspension.[L1/L2/R1/R2].trackComplianceInternalFlag

"flag to activate/deactivate the internal computation of suspension track compliance [0-1]"

Suspension.[L1/L2/R1/R2].trackCompliance

"externally computed track variation due to compliance"

Suspension.[L1/L2/R1/R2].trackComplianceVelocity

"externally computed first time derivative of track variation due to compliance"

Suspension.[L1/L2/R1/R2].trackComplianceAcceleration

"externally computed second time derivative of track variation due to compliance"

Suspension.[L1/L2/R1/R2].trackComplianceDerivativeWrtSwa

"externally computed first derivative of track variation due to compliance with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].trackComplianceDerivativeWrtJside

"externally computed first derivative of track variation due to compliance with respect to jounce"

Suspension.[L1/L2/R1/R2].trackComplianceDerivativeWrtJopp

"externally computed first derivative of track variation due to compliance with respect to jounce opposite"

Suspension.[L1/L2/R1/R2].svaKinematicInternalFlag

"flag to activate/deactivate the internal computation of suspension sva kinematics [0-1]"

Suspension.[L1/L2/R1/R2].svaDisplacement

"externally computed side view angle variation"

Suspension.[L1/L2/R1/R2].svaVelocity

"externally computed first time derivative of side view angle variation"

Suspension.[L1/L2/R1/R2].svaAcceleration

"externally computed second time derivative of side view angle variation"

Suspension.[L1/L2/R1/R2].svaDerivativeWrtSwa

"externally computed derivative of side view angle variation with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].svaDerivativeWrtJside

"externally computed derivative of side view angle variation, with respect to jounce (this side)"

Suspension.[L1/L2/R1/R2].svaDerivativeWrtJopp

"externally computed derivative of side view angle variation, with respect to jounce (opposite side)"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtSwaSwa

"externally computed second derivative of side view angle variation, with respect to swa /swa"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtSwaJside

"externally computed second derivative of side view angle variation, with respect to swa / jounce"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtSwaJopp

"externally computed second derivative of side view angle variation, with respect to swa / opposite jounce"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtJsideJside

"externally computed second derivative of side view angle variation, with respect to jounce / jounce"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtJsideJopp

"externally computed second derivative of side view angle variation, with respect to jounce / jounce opposite"

Suspension.[L1/L2/R1/R2].svaSecondDerivativeWrtJoppJopp

"externally computed second derivative of side view angle variation, with respect to jounce opposite / jounce opposite"

Suspension.[L1/L2/R1/R2].svaComplianceInternalFlag

"flag to activate/deactivate the internal computation of suspension sva compliance [0-1]"

Suspension.[L1/L2/R1/R2].svaCompliance

"externally computed side view angle variation due to compliance"

Suspension.[L1/L2/R1/R2].svaComplianceVelocity

"externally computed first time derivative of side view angle variation due to compliance"

Suspension.[L1/L2/R1/R2].svaComplianceAcceleration

"externally computed second time derivative of side view angle variation due to compliance"

Suspension.[L1/L2/R1/R2].svaComplianceDerivativeWrtSwa

"externally computed first derivative of side view angle variation due to compliance with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].svaComplianceDerivativeWrtJside

"externally computed first derivative of side view angle variation due to compliance with respect to jounce"

Suspension.[L1/L2/R1/R2].svaComplianceDerivativeWrtJopp

"externally computed first derivative of side view angle variation due to compliance with respect to jounce opposite"

Suspension.[L1/L2/R1/R2].toeKinematicInternalFlag

"flag to activate/deactivate the internal computation of suspension toe kinematics [0-1]"

Suspension.[L1/L2/R1/R2].toeDisplacement

"externally computed toe variation, used only if toeKinematicInternalFlag is set to 0"

Suspension.[L1/L2/R1/R2].toeVelocity

"externally computed first time derivative of toe variation"

Suspension.[L1/L2/R1/R2].toeAcceleration

"externally computed second time derivative of toe variation"

Suspension.[L1/L2/R1/R2].toeDerivativeWrtSwa

"externally computed derivative of toe variation with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].toeDerivativeWrtJside

"externally computed derivative of toe variation, with respect to jounce (this side)"

Suspension.[L1/L2/R1/R2].toeDerivativeWrtJopp

"externally computed derivative of toe variation, with respect to jounce (opposite side)"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtSwaSwa

"externally computed second derivative of toe variation, with respect to swa / swa"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtSwaJside

"externally computed second derivative of toe variation, with respect to swa / jounce"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtSwaJopp

"externally computed second derivative of toe variation, with respect to swa / opposite jounce"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtJsideJside

"externally computed second derivative of toe variation, with respect to jounce / jounce"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtJsideJopp

"externally computed second derivative of toe variation, with respect to jounce / jounce opposite"

Suspension.[L1/L2/R1/R2].toeSecondDerivativeWrtJoppJopp

"externally computed second derivative of toe variation, with respect to jounce opposite / jounce opposite"

Suspension.[L1/L2/R1/R2].toeComplianceInternalFlag

"flag to activate/deactivate the internal computation of suspension toe compliance [0-1]"

Suspension.[L1/L2/R1/R2].toeCompliance

"externally computed toe variation due to compliance"

Suspension.[L1/L2/R1/R2].toeComplianceVelocity

"externally computed first time derivative of toe variation due to compliance"

Suspension.[L1/L2/R1/R2].toeComplianceAcceleration

"externally computed second time derivative of toe variation due to compliance"

Suspension.[L1/L2/R1/R2].toeComplianceDerivativeWrtSwa

"externally computed first derivative of toe variation due to compliance with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].toeComplianceDerivativeWrtJside

"externally computed first derivative of toe variation due to compliance with respect to jounce"

Suspension.[L1/L2/R1/R2].toeComplianceDerivativeWrtJopp

"externally computed first derivative of toe variation due to compliance with respect to jounce opposite"

Suspension.[L1/L2/R1/R2].camberKinematicInternalFlag

"flag to activate/deactivate the internal computation of suspension camber kinematics [0-1]"

Suspension.[L1/L2/R1/R2].camberDisplacement

"externally computed camber variation, used only if camberKinematicInternalFlag is set to 0"

Suspension.[L1/L2/R1/R2].camberVelocity

"externally computed first time derivative of camber variation"

Suspension.[L1/L2/R1/R2].camberAcceleration

"externally computed second time derivative of camber variation"

Suspension.[L1/L2/R1/R2].camberDerivativeWrtSwa

"externally computed derivative of camber variation with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].camberDerivativeWrtJside

"externally computed derivative of camber variation, with respect to jounce (this side)"

Suspension.[L1/L2/R1/R2].camberDerivativeWrtJopp

"externally computed derivative of camber variation, with respect to jounce (opposite side)"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtSwaSwa

"externally computed second derivative of camber variation, with respect to swa / swa"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtSwaJside

"externally computed second derivative of camber variation, with respect to swa / jounce"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtSwaJopp

"externally computed second derivative of camber variation, with respect to swa / opposite jounce"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtJsideJside

"externally computed second derivative of camber variation, with respect to jounce / jounce"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtJsideJopp

"externally computed second derivative of camber variation, with respect to jounce / jounce opposite"

Suspension.[L1/L2/R1/R2].camberSecondDerivativeWrtJoppJopp

"externally computed second derivative of camber variation, with respect to jounce opposite / jounce opposite"

Suspension.[L1/L2/R1/R2].camberComplianceInternalFlag

"flag to activate/deactivate the internal computation of suspension camber compliance [0-1]"

Suspension.[L1/L2/R1/R2].camberCompliance

"externally computed camber variation due to compliance"

Suspension.[L1/L2/R1/R2].camberComplianceVelocity

"externally computed first time derivative of camber variation due to compliance"

Suspension.[L1/L2/R1/R2].camberComplianceAcceleration

"externally computed second time derivative of camber variation due to compliance"

Suspension.[L1/L2/R1/R2].camberComplianceDerivativeWrtSwa

"externally computed first derivative of camber variation due to compliance with respect to steering wheel angle"

Suspension.[L1/L2/R1/R2].camberComplianceDerivativeWrtJside

"externally computed first derivative of camber variation due to compliance with respect to jounce"

Suspension.[L1/L2/R1/R2].camberComplianceDerivativeWrtJopp

"externally computed first derivative of camber variation due to compliance with respect to jounce opposite"

Steering.[L1/L2/R1/R2].InternalSteeringColumnFlag

"flag to activate/deactivate the internal computation of steering column elasto-kinematics [0-1]"

Steering.[L1/L2/R1/R2].SteeringCompliance

"externally computed steering column compliance angle"

Steering.[L1/L2/R1/R2].SuspensionSplinesInput

"externally computed input to use in the suspension kinematics lookup table (it can be either steering wheel angle or rack displacement)"

Steering.[L1/L2/R1/R2].SuspensionSplinesInputVelocity

"externally computed time derivative of the input to use in the suspension kinematics lookup table (it can be either steering wheel angle or rack displacement)"

Steering.[L1/L2/R1/R2].RackDisplacement

"externally computed rack displacement"

Steering.[L1/L2/R1/R2].InternalForceAndMomentFeedbackFlag

"flag to activate/deactivate the internal computation of rack force and steering wheel torque[0-1]"

Steering.[L1/L2/R1/R2].SteeringTorque

"externally computed steering torque"

Steering.[L1/L2/R1/R2].RackForce

"externally computed rack force"

Steering.[L1/L2/R1/R2].LeftTierodForceOnRackDirection

"externally computed tierod force left"

Steering.[L1/L2/R1/R2].RightTierodForceOnRackDirection

"externally computed tierod force right"

Steering.[L1/L2/R1/R2].InternalKingpinMomentFlag

"flag to activate/deactivate the internal computation of kingpin moment[0-1]"

Steering.[L1/L2/R1/R2].KingpinMomentLeft

"externally computed kingpin moment left"

Steering.[L1/L2/R1/R2].KingPinMomentRight

"externally computed kingpin moment right"

External Suspension Main Elements User Input

The following list of inputs is enabled through the [external_suspension](#) flags available in the [Model Properties](#) parameter tree. For each of the main suspension force elements (spring, damper, bumpstop, reboundstop, antirollbar), there is an additional set of inputs that allow to modify the internal component.

External_<Components>.[L1/L2/R1/R2].<Components>_Scaling_Factor

"runtime scaling factor to be applied to force element [default value =1]."

External_<Components>.[L1/L2/R1/R2].<Components>_Motion_Ratio_External_Activation

"activation input for user external element motion ratio [0 1]."

External_<Components>.[L1/L2/R1/R2].<Components>_Motion_Ratio

"runtime element motion ratio."

External Suspension Scaling Factors

The following list of inputs is enabled through the [external_suspension](#) flags available in the [Model Properties](#) parameter tree. For each of the suspension characteristic (track, base, side view angle, toe, camber), there are runtime scaling factors to modify suspension kinematic/compliance response

Kinematics:

External_Kinematic_Scaling.[front/rear].Base

"runtime scaling factor to be applied to base movement w.r.t. jounce [default value =1]."

External_Kinematic_Scaling.[front/rear].Track

"runtime scaling factor to be applied to track movement w.r.t. jounce [default value =1]."

External_Kinematic_Scaling.[front/rear].Sva

"runtime scaling factor to be applied to side view angle rotation w.r.t. jounce [default value =1]."

External_Kinematic_Scaling.[front/rear].Camber

"runtime scaling factor to be applied to camber angle rotation w.r.t. jounce [default value =1]."

External_Kinematic_Scaling.[front/rear].Toe

"runtime scaling factor to be applied to toe angle rotation w.r.t. jounce [default value =1]."

Compliance:

External_Compliance_Scaling.[front/rear].Base_vs_ContactPatch_Fx

"runtime scaling factor to be applied to base deformation w.r.t. contact patch Fx [default value =1]."

External_Compliance_Scaling.[front/rear].Track_vs_ContactPatch_Fy

"runtime scaling factor to be applied to track deformation w.r.t. contact patch Fy [default value =1]."

External_Compliance_Scaling.[front/rear].Sva_vs_ContactPatch_FxBraking

"runtime scaling factor to be applied to side view angle deformation w.r.t. contact patch Fx braking [default value =1]".

External_Compliance_Scaling.[front/rear].Camber_vs_ContactPatch_Fy

"runtime scaling factor to be applied to camber angle deformation w.r.t. contact patch Fy [default value =1]".

External_Compliance_Scaling.[front/rear].Toe_vs_ContactPatch_Mz

"runtime scaling factor to be applied to toe angle deformation w.r.t. contact patch Mz [default value =1]".

Road External

The following list of inputs is enabled when submitting an event with an [external road](#) as Road Data File. In order to characterize an external road it is sufficient to provide VI-CarRealTime with the runtime contact patches information (look Input Array below) for all the tires: it can be done by means of a [plug-in dll](#), [FMI interface](#) or by using Simulink (please refer to [Using External Road](#) tutorial).

Road_External.L1

"Input Array defining the road data info for front left tire"

Road_External.R1

"Input Array defining the road data info for front right tire"

Road_External.L2

"Input Array defining the road data info for rear left tire"

Road_External.R2

"Input Array defining the road data info for rear right tire"

Input Array needs the following runtime parameters, to be expressed in [global reference system](#) :

- *Contact Patch Longitudinal Position*
X location of the contact patch.
- *Contact Patch Lateral Position*
Y location of the contact patch.
- *Contact Patch Vertical Position*
Z location of the contact patch.
- *Road Normal X*
X direction cosine of road normal.
- *Road Normal Y*
Y direction cosine of road normal.
- *Road Normal Z*
Z direction cosine of road normal.
- *Friction*
friction coefficient of the road.

Steering Assist

The following list of inputs is available only when [advanced steering system](#) is active:

steer_assist.force

"additional input force which is applied to the rack part."

steer_assist.torque

"additional input torque which is applied to the upper or lower steering column depending on the torsion bar location."

steering_torque.torque

"additional steering wheel angle input torque which can be used to drive the vehicle."

steering_assist.motor_torque

"additional torque applied to the Electric Motor. Such torque is multiplied by [Electric Motor Ratio](#) and then applied to the selected component (rack, pinion, column)."

1.3.2 Output Channels

This section shows all the output channels available in VI-CarRealTime.

For easier finding the desired output, all the variables are grouped in the following categories:

- [Aerodynamic](#)
- [Animator Widget](#)
- [Body Part Acceleration](#)
- [Body Part Displacement](#)
- [Body Part Velocity](#)
- [Brake](#)
- [Bump-stop](#)
- [Rebound-stop](#)
- [Cab Bushing/Beam](#)
- [Cab](#)
- [Chassis Acceleration](#)
- [Chassis Displacement](#)
- [Chassis Velocity](#)
- [Chassis Compliance](#)
- [Clutch](#)
- [Compliance Extra Outputs](#)
- [Condition Sensor](#)
- [Cross Weight](#)
- [Damper](#)
- [Differential Speeds](#)
- [Differential Torques](#)
- [Differential Central](#)
- [Differential Front](#)
- [Differential Rear](#)
- [Differential User Output](#)
- [Driver Demands](#)
- [Driver Monitors](#)
- [Engine](#)
- [Motor](#)
- [Fuel Consumption](#)

- [Gyro](#)
- [Global Pad](#) (7 Post)
- [Global Upright](#) (7 Post)
- [Hook](#)
- [Inerter](#)
- [Kingpin Contributes](#)
- [Spring](#)
- [Rear Steering System](#)
- [Ride Height](#)
- [Road](#)
- [Sensor](#)
- [SevenPostrig](#)
- [Skidplate](#)
- [Steering](#)
- [Suspension](#)
- [Suspension Testrig](#)
- [System Threshold](#)
- [System Mass](#)
- [Tire](#)
- [Torque Converter](#)
- [Transmission](#)
- [Upright Accelerometer](#)
- [Vehicle](#)
- [Vehicle States](#)
- [Wheel](#)
- [Wheel Angles](#)
- [Graphics Part and Force](#)

Aerodynamic

Here the list of all **Aerodynamic** outputs:

aero_balance

"Factor between front downforce and total downforce (no units)"

center_downforce

"Aerodynamic down force at drag application point (newton) Z+ down"

center_sideforce

"Aerodynamic side force at center application point (newton) Y+ left"

drag_arm

"Central force application height with respect to road plane (meter)"

drag_force

"Aerodynamic drag force at drag application point (newton) X+ backward"

drag_force_front

"Aerodynamic drag force at front application point (newton) X+ backward"

drag_force_rear

"Aerodynamic drag force at rear application point (newton) X+ backward"

drag_torque

"Pitch Aerodynamic moment (newton-meter). Positive value represents a positive rear to front load transfer"

front_downforce

"Aerodynamic down force at front application point (newton) Z+ down"

front_sideforce

"Aerodynamic side force at front application point (newton) Y+ left"

height_auxiliary

"Ride height at auxiliary front sensor w.r.t. road plane (meter)"

height_front

"Ride height at front sensor w.r.t. road plane (meter)"

height_rear

"Ride height at rear sensor w.r.t. road plane (meter)"

lateral_velocity

"If aerodynamic forces are applied in the vehicle system (using [road_plane_direction_flag](#) = 0) it is the Y-component of velocity of center aerodynamic application point w.r.t. ground in the sprung mass body frame. If [road_plane_direction_flag](#) = 1, it is the Y-component of velocity of center aerodynamic application point w.r.t. road plane in the runtime road plane frame - z+ up road normal, x+ vehicle forward - (meter/second)"

longitudinal_velocity

"If aerodynamic forces are applied in the vehicle system (using [road_plane_direction_flag](#) = 0) it is the X-component of velocity of center aerodynamic application point w.r.t. ground in the sprung mass body frame. If [road_plane_direction_flag](#) = 1, it is the X-component of velocity of center aerodynamic application point w.r.t. road plane in the runtime road plane frame - z+ up road normal, x+ vehicle forward - (meter/second)"

pitch

"Chassis pitch angle w.r.t road plane (radian)"

rear_downforce

"Aerodynamic down force at rear application point (newton) Z+ down"

rear_sideforce

"Aerodynamic side force at rear application point (newton) Y+ left"

roll

"Negative of chassis roll-rotation angle w.r.t. road plane (radian)"

roll_torque

"Mx Aerodynamic moment (newton-meter) X+ backward"

side_slip

"side slip angle computed as atan([aero_forces_lateral_velocity](#) / [aero_forces_longitudinal_velocity](#)) (radian)"

steer

"Front wheel steer angle at spindle -relative to vehicle body- (radian)"

vertical_velocity

"If aerodynamic forces are applied in the vehicle system (using [road plane direction flag](#) = 0) it is the Z-component of velocity of center aerodynamic application point w.r.t. ground in the sprung mass body frame. If [road plane direction flag](#) = 1, it is the Z-component of velocity of center aerodynamic application point w.r.t. road plane in the runtime road plane frame - z+ up road normal, x+ vehicle forward - (meter/second)"

yaw_torque

"Mz Aerodynamic moment (newton-meter) Z+ down"

Animator Widget

Here the list of all **Animator Widget** outputs:

longitudinal_speed

"X-component of vehicle-mass-center velocity in sprung-mass-body frame (km/h)"

engine_rpm

"Engine rotational velocity (rpm)"

Body Part Acceleration

Here the list of all **Body Part Acceleration** outputs:

Note: These outputs represent the accelerations of [body rigid part](#) center of gravity relative to global frame projected in the body part frame.

Note: The outputs are available only when at least one rigid part is active in [body subsystem](#) and the output channel prefix is composed by using the rigid part name (i.e. for *front_body* part the output channels are *body_front_body_accelerations*).

body_<rigid_body_name>_accelerations:lateral

"Negative of Y-component of rigid body mass center acceleration in rigid body frame (g's)"

body_<rigid_body_name>_accelerations:longitudinal

"X-component of rigid body mass center acceleration in rigid body frame (g's)"

body_<rigid_body_name>_accelerations:pitch

"Negative of Y-component of rigid body angular acceleration in rigid body frame (radian/second**2)"

body_<rigid_body_name>_accelerations:roll

"Negative of X-component of rigid body angular acceleration in rigid body frame (radian/second**2)"

body_<rigid_body_name>_accelerations:vertical

"Z-component of rigid body mass center acceleration in rigid body frame (g's)"

body_<rigid_body_name>_accelerations:yaw

"Z-component of rigid body angular acceleration in rigid body frame (radian/second**2)"

Body Part Displacement

Here the list of all **Body Part Displacement** outputs:

Note: These outputs represent the displacements of [body rigid part](#) center of gravity relative to global frame projected in the body part frame.

Note: The outputs are available only when at least one rigid part is active in [body subsystem](#) and the output channel prefix is composed by using the rigid part name (i.e. for *front_body* part the output channels are *body_front_body_displacements*).

body_<rigid_body_name>_displacements:lateral

"Negative of Y-coordinate of rigid body mass center in global frame (meter)"

body_<rigid_body_name>_displacements:longitudinal

"X-coordinate of rigid body mass center in global frame (meter)"

body_<rigid_body_name>_displacements:vertical

"Z-coordinate of rigid body mass center in global frame (meter)"

body_<rigid_body_name>_displacements:pitch

"Rigid body pitch-rotation angle (2nd rotation in Body/321 sequence) with respect to global frame (radian)"

body_<rigid_body_name>_displacements:roll

"Negative of rigid body roll-rotation angle (3rd rotation in Body/321 sequence) with respect to global frame (radian)"

body_<rigid_body_name>_displacements:yaw

"Rigid body yaw-rotation angle (1st rotation in Body/321 sequence) relative with respect to frame (radian)"

Body Part Velocity

Here the list of all **Body Part Velocity** outputs:

Note: These outputs represent the velocities of [body rigid part](#) center of gravity relative to global frame projected in the body part frame.

Note: The outputs are available only when at least one rigid part is active in [body subsystem](#) and the output channel prefix is composed by using the rigid part name (i.e. for *front_body* part the output channels are *body_front_body_velocities*).

body_<rigid_body_name>_velocities:lateral

"Negative of Y-component of rigid body mass center velocity in rigid body frame (km/h)"

body_<rigid_body_name>_velocities:longitudinal

"X-component of rigid body mass center velocity in rigid body frame (km/h)"

body_<rigid_body_name>_velocities:pitch

"Negative of Y-component of rigid body angular velocity in rigid body frame (radian/second)"

body_<rigid_body_name>_velocities:roll

"Negative of X-component of rigid body angular velocity in rigid body frame (radian/second)"

body_<rigid_body_name>_velocities:vertical

"Z-component of rigid body mass center velocity in rigid body frame (km/h)"

body_<rigid_body_name>_velocities:yaw

"Z-component of rigid body angular velocity in rigid body frame (radian/second)"

Brake

Here the list of all **Brake** outputs:

Brake_Moment:[L,R][1,2]

"Brake moment (newton-meter)"

Chamber_Pressure:[L,R][1,2]

"fluid pressure in brake chamber (MPa)"

Lookup model outputs:

The following outputs will refer to [wheel lookup model under low speed braking](#) conditions and represent the outputs for the lookup algorithm

Locked_Damper_Moment:[L,R][1,2]

"Brake locked damper moment (newton-meter)"

Locked_Moment:[L,R][1,2]

"brake lockup moment (newton-meter)"

Locked_Spring_Moment:[L,R][1,2]

"brake locked spring moment (newton-meter)"

Locked_Switch:[L,R][1,2]

"brake locked switch state (-)"

Lockup_Rotation:[L,R][1,2]

"brake lockup rotation (radian)"

ABS_activity:[L,R][1,2]

"Internal anti-lock braking system runtime activation [0/1]"

Bumpstop

Here the list of all **Bumpstop** outputs:

bu[l,r]_bounce_stop_data:force_[front,rear]

"suspension bumpstop force at bumpstop element (always positive) (newton)"

bu[l,r]_bounce_stop_data:jounce_[front,rear]

"suspension bumpstop gap (positive in elongation) (meter)"

bu[l,r]_bounce_stop_data:velocity_[front,rear]

"suspension bumpstop velocity (positive in elongation) (meter/second)"

Jounce_Bumper:Force_At_Bumper:[L,R][1,2]

"suspension bumpstop force at bumpstop element (always positive) (newton)"

Jounce_Bumper:Force_At_Wheel:[L,R][1,2]

"suspension bumpstop force at suspension jounce-bumper force at wheel (newton)"

Jounce_Bumper:Jounce:[L,R][1,2]

"suspension bumpstop gap (positive in compression) (meter)"

Jounce_Bumper:Jounce_Rate:[L,R][1,2]

"suspension bumpstop compression velocity (meter/second)"

bus_dmp_jounce_stop_data:displacement:[front,rear]

"single damper bumpstop gap (positive in elongation) (meter)"

bus_dmp_jounce_stop_data:force:[front,rear]

"single damper bumpstop force at element (always positive) (newton)"

bus_dmp_jounce_stop_data:velocity:[front,rear]
"single damper bumpstop elongation velocity (meter/second)"

bus_ins_jounce_stop_data:displacement:[front,rear]
"single inerter bumpstop gap (positive in elongation) (meter)"

bus_ins_jounce_stop_data:force:[front,rear]
"single inerter bumpstop force at element (always positive) (newton)"

bus_ins_jounce_stop_data:velocity:[front,rear]
"single inerter bumpstop elongation velocity (meter/second)"

bus_roll_dmp_jounce_stop_data:displacement:[front,rear]
"roll damper bumpstop gap (positive in elongation) (meter)"

bus_roll_dmp_jounce_stop_data:force:[front,rear]
"roll damper bumpstop force at element (newton)"

bus_roll_dmp_jounce_stop_data:velocity:[front,rear]
"roll damper bumpstop elongation velocity (meter/second)"

bus_spr_jounce_stop_data:displacement:[front,rear]
"single spring bumpstop gap (positive in elongation) (meter)"

bus_spr_jounce_stop_data:force:[front,rear]
"single spring bumpstop force at element (newton)"

bus_spr_jounce_stop_data:velocity:[front,rear]
"single spring bumpstop elongation velocity (meter/second)"

Reboundstop

Here the list of all **Reboundstop** outputs:

bu[l,r]_rebound_stop_data:force:[front,rear]
"suspension rebound stop force at element (always negative) (newton)"

bu[l,r]_rebound_stop_data:velocity:[front,rear]
"suspension rebound stop elongation velocity (meter/second)"

Rebound_Bumper:Force_At_Bumper:[L,R][1,2]
"suspension rebound stop force at element (always negative) (newton)"

Rebound_Bumper:Force_At_Wheel:[L,R][1,2]
"suspension rebound stop force at wheel (newton)"

Rebound_Bumper:Jounce:[L,R][1,2]
"suspension rebound stop gap (positive in compression) (meter)"

Rebound_Bumper:Jounce_Acceleration:[L,R][1,2]
"suspension rebound stop compression acceleration (meter/second**2)"

Rebound_Bumper:Jounce_Rate:[L,R][1,2]
"suspension rebound stop compression velocity (meter/second)"

res_dmp_rebound_stop_data_displacement_[front,rear]
"single damper rebound stop gap (positive in elongation) (meter)"

res_dmp_rebound_stop_data:force:[front,rear]

"single damper rebound stop force at element (newton)"

res_dmp_rebound_stop_data:velocity:[front,rear]

"single damper rebound stop elongation velocity (meter/second)"

res_ins_rebound_stop_data:displacement:[front,rear]

"single inerter rebound stop gap (positive in elongation) (meter)"

res_ins_rebound_stop_data:force:[front,rear]

"single inerter rebound stop force at element (newton)"

res_ins_rebound_stop_data:velocity:[front,rear]

"single inerter rebound stop elongation velocity (meter/second)"

res_roll_dmp_rebound_stop_data:displacement:[front,rear]

"single roll damper rebound stop gap (positive in elongation) (meter)"

res_roll_dmp_rebound_stop_data:force:[front,rear]

"single roll damper rebound stop force at element (newton)"

res_roll_dmp_rebound_stop_data:velocity:[front,rear]

"single roll damper rebound stop elongation velocity (meter/second)"

res_spr_rebound_stop_data:displacement:[front,rear]

"single spring rebound stop gap (positive in elongation) (meter)"

res_spr_rebound_stop_data:force:[front,rear]

"single spring rebound stop force at element (newton)"

res_spr_rebound_stop_data:velocity:[front,rear]

"single spring rebound stop elongation velocity (meter/second)"

Cab Bushing/Beam

Here the list of all **Cab Bushing/Beam** outputs:

Note: These outputs refer to the [cab auxiliary subsystem](#), so they are present only when such subsystem is present in the model.

bgs_data_1:AX

"Relative rotation R1 between I and J parts (radian)"

bgs_data_1:AY

"Relative rotation R2 between I and J parts (radian)"

bgs_data_1:AZ

"Relative torsional rotation between I and J parts (radian)"

bgs_data_1:FX

"Bushing force X with respect to connection 2 (Newton)"

bgs_data_1:FY

"Bushing force Y with respect to connection 2 (Newton)"

bgs_data_1:FZ

"Bushing force Z with respect to connection 2 (Newton)"

bgs_data_1:TX

"Bushing moment R1 with respect to connection 2 (Newton-meter)"

bgs_data_1:TY

"Bushing moment R2 with respect to connection 2 (Newton-meter)"

bgs_data_1:TZ

"Bushing moment R3 with respect to connection 2 (Newton-meter)"

bgs_data_1:VX

"Relative velocity X between I and J parts (meter/second)"

bgs_data_1:VY

"Relative velocity Y between I and J parts (meter/second)"

bgs_data_1:VZ

"Relative velocity Z between I and J parts (meter/second)"

bgs_data_1:WX

"Relative angular velocity w1 between I and J parts (radian/second)"

bgs_data_1:WY

"Relative angular velocity w2 between I and J parts (radian/second)"

bgs_data_1:WZ

"Relative angular velocity w3 between I and J parts (radian/second)"

bgs_data_1:X

"Relative displacement X between I and J parts (meter)"

bgs_data_1:Y

"Relative displacement Y between I and J parts (meter)"

bgs_data_1:Z

"Relative displacement Z between I and J parts (meter)"

Cab

Here the list of all **Cab** outputs:

Note: These outputs refer to the [cab auxiliary subsystem](#), so they are present only when such subsystem is present in the model.

cab_accelerations:lateral

"Negative of Y-component of cab mass center acceleration in cab body frame (g's)"

cab_accelerations:longitudinal

"X-component of cab mass center acceleration in cab body frame (g's)"

cab_accelerations:pitch

"Negative of Y-component of cab angular acceleration in cab body frame (radian/second²)"

cab_accelerations:roll

"Negative of X-component of cab angular acceleration in cab body frame (radian/second²)"

cab_accelerations:vertical

"Z-component of cab mass center acceleration in cab body frame (g's)"

cab_accelerations:yaw

"Z-component of cab angular acceleration in cab body frame (radian/second²)"

cab_displacements:lateral

"Negative of Y-coordinate of cab mass center in global frame (meter)"

cab_displacements:longitudinal

"X-coordinate of cab mass center in global frame (meter)"

cab_displacements:pitch

"cab pitch-rotation angle (2nd rotation in Body/321 sequence) relative to global frame (radian)"

cab_displacements:roll

"Negative of cab roll-rotation angle (3rd rotation in Body/321 sequence) relative to global frame (radian)"

cab_displacements:vertical

"Z-coordinate of cab mass center in global frame (meter)"

cab_displacements:yaw

"cab yaw-rotation angle (1st rotation in Body/321 sequence) relative to global frame (radian)"

cab_velocities:lateral

"Negative of Y-component of cab mass center velocity in sprung mass body frame (km/h)"

cab_velocities:longitudinal

"X-component of cab mass center velocity in cab body frame (km/h)"

cab_velocities:pitch

"Negative of Y-component of cab angular velocity in cab body frame (radian/second)"

cab_velocities:roll

"Negative of X-component of cab angular velocity in cab body frame (radian/second)"

cab_velocities:vertical

"Z-component of cab mass center velocity in cab body frame (km/h)"

cab_velocities:yaw

"Z-component of vehicle angular velocity in cab body frame (radian/second)"

Chassis Acceleration

Here the list of all **Chassis Acceleration** outputs:

Note: These outputs represent the displacement of [Adams Car cg point](#) relative to global frame projected in the sprung mass frame

chassis_accelerations:lateral

"Negative of Y-component of vehicle-mass-center acceleration in sprung-mass-body frame (g's)"

chassis_accelerations:longitudinal

"X-component of vehicle-mass-center acceleration in sprung-mass-body frame (g's)"

chassis_accelerations:pitch

"Negative of Y-component of vehicle angular acceleration in sprung-mass-body frame (radian/second**2)"

chassis_accelerations:roll

"Negative of X-component of vehicle angular acceleration in sprung-mass-body frame (radian/second**2)"

chassis_accelerations:vertical

"Z-component of vehicle-mass-center acceleration in sprung-mass-body frame (g's)"

chassis_accelerations:yaw

"Z-component of vehicle angular acceleration in sprung-mass-body frame (radian/second**2)"

Chassis Displacement

Here the list of all **Chassis Displacement** outputs:

Note: These outputs represent the displacement of [Adams Car cg point](#) relative to global frame projected in the sprung mass frame

chassis_displacements:lateral

"Negative of Y-coordinate of vehicle mass center in global frame (meter)"

chassis_displacements:longitudinal

"X-coordinate of vehicle mass center in global frame (meter)"

chassis_displacements:vertical

"Z-coordinate of vehicle mass center in global frame (meter)"

chassis_displacements:pitch

"Vehicle pitch-rotation angle (2nd rotation in Body/321 sequence) with respect to global frame (radian)"

chassis_displacements:roll

"Negative of vehicle roll-rotation angle (3rd rotation in Body/321 sequence) with respect to global frame (radian)"

chassis_displacements:yaw

"Vehicle yaw-rotation angle (1st rotation in Body/321 sequence) relative with respect to frame (radian)"

chassis_displacements:pitch_wrt_road

"Vehicle pitch-rotation angle (2nd rotation in Body/321 sequence) with respect to road_plane (radian)"

chassis_displacements:roll_wrt_road

"Negative of vehicle roll-rotation angle (3rd rotation in Body/321 sequence) with respect to road plane (radian)"

chassis_displacements:vertical_wrt_road

"Z-coordinate of vehicle mass center with respect to road plane (meter)"

Chassis Velocity

Here the list of all **Chassis Velocity** outputs:

Note: These outputs represent the displacement of [Adams Car cg point](#) relative to global frame projected in the sprung mass frame

chassis_velocities:lateral

"Negative of Y-component of vehicle-mass-center velocity in sprung-mass-body frame (km/h)"

chassis_velocities:longitudinal

"X-component of vehicle-mass-center velocity in sprung-mass-body frame (km/h)"

chassis_velocities:pitch

"Negative of Y-component of vehicle angular velocity in sprung-mass-body frame (radian/second)"

chassis_velocities:roll

"Negative of X-component of vehicle angular velocity in sprung-mass-body frame (radian/second)"

chassis_velocities:vertical

"Z-component of vehicle-mass-center velocity in sprung-mass-body frame (km/h)"

chassis_velocities:yaw

"Z-component of vehicle angular velocity in sprung-mass-body frame (radian/second)"

Chassis Compliance

Here the list of all **Chassis Compliance** outputs:

jfs_torsional_stiffness_data:AX_1

"Relative rotation around x between front and rear chassis parts (with respect to rear chassis part) (radian)"

jfs_torsional_stiffness_data:AY_1

"Relative rotation around y between front and rear chassis parts (with respect to rear chassis part) (radian)"

jfs_torsional_stiffness_data:AZ_1

"Relative rotational around z between front and rear chassis parts (with respect to rear chassis part) (radian)"

jfs_torsional_stiffness_data:X_1

"Relative displacement X between front and rear chassis parts (with respect to rear chassis part) (meter)"

jfs_torsional_stiffness_data:Y_1

"Relative displacement Y between front and rear chassis parts (with respect to rear chassis part) (meter)"

jfs_torsional_stiffness_data:Z_1

"Relative displacement Z between front and rear chassis parts (with respect to rear chassis part) (meter)"

jfs_torsional_stiffness_data:FX_1

"Chassis compliance force X (with respect to rear chassis part) (newton)"

jfs_torsional_stiffness_data:FY_1

"Chassis compliance force Y (with respect to rear chassis part) (newton)"

jfs_torsional_stiffness_data:FZ_1

"Chassis compliance force Z (with respect to rear chassis part) (newton)"

jfs_torsional_stiffness_data:TX_1

"Chassis compliance torque X (with respect to rear chassis part) (newton-meter)"

jfs_torsional_stiffness_data:TY_1

"Chassis compliance torque Y (with respect to rear chassis part) (newton-meter)"

jfs_torsional_stiffness_data:TZ_1

"Chassis compliance torque Z (with respect to rear chassis part) (newton-meter)"

Clutch

Here the list of all **Clutch** outputs:

slip_angle

"clutch slip angle (radian)"

torque

"torque transmitted by the clutch (newton-meter)"

clutch_rpm

"clutch output shaft rotational speed (rpm). "

gearbox_inputshaft_alpha
"gearbox input shaft angular acceleration (radian/second**2)"

gearbox_inputshaft_angle
"gearbox input shaft rotation angle (radian)"

gearbox_inputshaft_omega
"gearbox input shaft angular velocity (radian/second)"

inputplate_torque
"input plate torque (newton-meter)"

effective_clutch
"effective clutch demand value [0-1]"

Compliance Extra Outputs

These outputs are available only when [compliance_extra_output](#) flag is set to 1 in the model system tree. Furthermore, compliance velocity is computed only when [compliance_velocity_computation_flag](#) is active.

base_compliance:[L1,R1,L2,R2]
"Wheelbase variation due to compliance contribute (meter)"

base_compliance_velocity:[L1,R1,L2,R2]
"Wheelbase velocity variation due to compliance contribute (meter/second)"

camber_compliance:[L1,R1,L2,R2]
"Camber variation due to compliance contribute (meter)"

base_compliance_velocity:[L1,R1,L2,R2]
"Camber velocity variation due to compliance contribute (meter/second)"

side_view_compliance:[L1,R1,L2,R2]
"Side View Angle variation due to compliance contribute (radian)"

side_view_compliance_velocity:[L1,R1,L2,R2]
"Side View Angle velocity variation due to compliance contribute (radian/second)"

toe_compliance:[L1,R1,L2,R2]
"Toe variation due to compliance contribute (radian)"

toe_compliance_velocity:[L1,R1,L2,R2]
"Toe velocity variation due to compliance contribute (radian/second)"

track_compliance:[L1,R1,L2,R2]
"Track variation due to compliance contribute (meter)"

track_compliance_velocity:[L1,R1,L2,R2]
"Track velocity variation due to compliance contribute (meter/second)"

Condition Sensor

Here the list of all **Condition Sensor** outputs:

condition_sensors_side_slip_angle
"Vehicle side-slip angle: ATAN2(VY,VX) , where VX and VY are the velocities of [Adams Car cg point](#) expressed in the tracking system (radians)"

Cross Weight

Here the list of all **Cross Weight** outputs:

cross_weight:crossL1R2_ratio

"cross weight ratio as defined : (FrontLeft_TireFz + RearRight_TireFz) /Sum (TireFz)"

cross_weight:crossR1L2_ratio

"cross weight ratio as defined : (FrontRight_TireFz + RearLeft_TireFz) /Sum (TireFz)"

cross_weight:nose_ratio

"cross weight ratio as defined : (FrontRight_TireFz + FrontLeft_TireFz) /Sum (TireFz)"

cross_weight:side_right_ratio

"cross weight ratio as defined : (FrontRight_TireFz + RearRight_TireFz) /Sum (TireFz)"

Damper

Here the list of all **Damper** outputs:

da[l,r]_ride_damper_data:force:[front,rear]

"suspension damper force at element (newton)"

da[l,r]_ride_damper_data:velocity:[front,rear]

"suspension damper extension rate (positive in elongation) (meter/second)"

Damper:Force_At_Damper:[L,R][1,2]

"suspension damper force at damper (newton)"

Damper:Force_At_Wheel:[L,R][1,2]

"suspension damper force at wheel (newton)"

Damper:Jounce:[L,R][1,2]

"suspension damper jounce (positive in compression) (meter)"

Damper:Jounce_Acceleration:[L,R][1,2]

"suspension damper jounce acceleration (positive in compression) (meter/second**2)"

Damper:Jounce_Rate:[L,R][1,2]

"suspension damper jounce rate (positive in compression) (meter/second)"

das_ride_damper_data:displacement:[front,rear]

"central damper jounce (positive in elongation) (meter)"

das_ride_damper_data:force:[front,rear]

"central damper force at element (newton)"

das_ride_damper_data:velocity:[front,rear]

"central damper extension rate (positive in elongation) (meter/second)"

das_roll_damper_data:displacement:[front,rear]

"roll damper jounce (positive in elongation) (meter)"

das_roll_damper_data:force_at_wheel:[front,rear]

"roll damper force at wheel (newton)"

das_roll_damper_data:force:[front,rear]

"roll damper force at element (newton)"

das_roll_damper_data:velocity:[front,rear]

"roll damper extension rate (positive in elongation) (meter/second)"

das_third_damper_data:displacement:[front,rear]

"third-damper jounce (positive in elongation) (meter)"

das_third_damper_data:force:[front,rear]

"third-damper force at damper (newton)"

das_third_damper_data:velocity:[front,rear]

"third-damper extension rate (positive in elongation) (meter/second)"

secondary_damper:displacement:[front_left,front_right,rear_left,rear_right]

"[auxiliary main damper](#) jounce (positive in elongation) (meter)"

secondary_damper:force:[front_left,front_right,rear_left,rear_right]

"[auxiliary main damper](#) force at damper (newton)"

secondary_damper:velocity:[front_left,front_right,rear_left,rear_right]

"[auxiliary main damper](#) extension rate (positive in elongation) (meter/second)"

Differential Speeds

Here the list of all **Differential Speeds** outputs:

halfshaft_omega_delta_[front,rear]

"delta angular velocity between right and left shaft (radian/second)"

halfshaft_omega_delta_central

"delta angular velocity between front shaft and rear shaft (radian/second)"

halfshaft_omega_[left,right]_[front,rear]

"angular velocity of single shaft (radian/second)"

halfshaft_omega_central_[front,rear]

"angular velocity of single shaft (radian/second)"

Differential Torques

Here the list of all **Differential Torques** outputs:

differential_torque_[front,rear]

"differential torque (right - left) split by differential (Newton-meter)"

differential_torque_central

"differential torque (front - rear) split by differential (Newton-meter)"

output_torque_[left,right]_[front,rear]

"output torque coming from differential (Newton-meter)"

output_torque_central_[front,rear]

"output torque coming from differential (Newton-meter)"

Differential Central

Here the list of all **Central Differential** outputs:

diff_front_inputshaft_alpha

"central differential front input shaft angular acceleration (radian/second**2)"

diff_front_inputshaft_angle

"central differential front input shaft rotation angle (radian)"

diff_front_inputshaft_omega

"central differential front input shaft angular velocity (radian/second)"

diff_rear_inputshaft_alpha

"central differential rear input shaft angular acceleration (radian/second**2)"

diff_rear_inputshaft_angle

"central differential rear input shaft rotation angle (radian)"

diff_rear_inputshaft_omega

"central differential rear input shaft angular velocity (radian/second)"

gearbox_outputshaft_torque

"gearbox output shaft torque (Newton-meter)"

Differential Front

Here the list of all **Front Differential** outputs:

wheel_front_left_alpha

"front left wheel angular acceleration (radian/second**2)"

wheel_front_left_angle

"front left wheel rotation angle (radian)"

wheel_front_left_omega

"front left wheel angular velocity (radian/second)"

wheel_front_right_alpha

"front right wheel angular acceleration (radian/second**2)"

wheel_front_right_angle

"front right wheel rotation angle (radian)"

wheel_front_right_omega

"front right wheel angular velocity (radian/second)"

diff_central_front_outputshaft_torque

"center differential front output shaft torque (Newton-meter)"

Differential Rear

Here the list of all **Rear Differential** outputs:

wheel_rear_left_alpha

"rear left wheel angular acceleration (radian/second**2)"

wheel_rear_left_angle

"rear left wheel rotation angle (radian)"

wheel_rear_left_omega

"rear left wheel angular velocity (radian/second)"

wheel_rear_right_alpha

"rear right wheel angular acceleration (radian/second**2)"

wheel_rear_right_angle

"rear right wheel rotation angle (radian)"

wheel_rear_right_omega

"rear right wheel angular velocity (radian/second)"

diff_central_rear_outputshaft_torque

"center differential rear output shaft torque (Newton-meter)"

Differential User Output

Here the list of all **Differential User Outputs**:

Note: These outputs are available only when [enable_extra_output](#) flag is set to 1.

UserOut_differential_[front/rear].X_Outer_[left/right]

"X location in [Vehicle Reference System](#), of the drive shaft [outer convel](#)"

UserOut_differential_[front/rear].Y_Outer_[left/right]

"Y location in [Vehicle Reference System](#), of the drive shaft [outer convel](#)"

UserOut_differential_[front/rear].Z_Outer_[left/right]

"Z location in [Vehicle Reference System](#), of the drive shaft [outer convel](#)"

UserOut_differential_[front/rear].X_Inner_[left/right]

"X location in [Vehicle Reference System](#), of the drive shaft [inner convel](#)"

UserOut_differential_[front/rear].Y_Inner_[left/right]

"Y location in [Vehicle Reference System](#), of the drive shaft [inner convel](#)"

UserOut_differential_[front/rear].Z_Inner_[left/right]

"Z location in [Vehicle Reference System](#), of the drive shaft [inner convel](#)"

UserOut_differential_[front/rear].FX_Outer_[left/right]

"X-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].FY_Outer_[left/right]

"Y-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].FZ_Outer_[left/right]

"Z-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].MX_Outer_[left/right]

"X-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].MY_Outer_[left/right]

"Y-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].MZ_Outer_[left/right]

"Z-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the wheel center location"

UserOut_differential_[front/rear].FX_Outer_Convel_[left/right]

"X-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].FY_Outer_Convel_[left/right]

"Y-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].FZ_Outer_Convel_[left/right]

"Z-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].MX_Outer_Convel_[left/right]

"X-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].MY_Outer_Convel_[left/right]

"Y-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].MZ_Outer_Convel_[left/right]

"Z-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [outer convel location](#)"

UserOut_differential_[front/rear].FX_Inner_Convel_[left/right]

"X-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].FY_Inner_Convel_[left/right]

"Y-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].FZ_Inner_Convel_[left/right]

"Z-component of the force in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].MX_Inner_Convel_[left/right]

"X-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].MY_Inner_Convel_[left/right]

"Y-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].MZ_Inner_Convel_[left/right]

"Z-component of the torque in [Vehicle Reference System](#), acting on the drive shaft at the [inner convel location](#)"

UserOut_differential_[front/rear].FX_Inner_Chassis_[left/right]

"X-component of the force in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convel ones"

UserOut_differential_[front/rear].FY_Inner_Chassis_[left/right]

"Y-component of the force in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convel ones"

UserOut_differential_[front/rear].FZ_Inner_Chassis_[left/right]

"Z-component of the force in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convel ones"

UserOut_differential_[front/rear].MX_Inner_Chassis_[left/right]

"X-component of the torque in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].MY_Inner_Chassis_[left/right]

"Y-component of the torque in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].MZ_Inner_Chassis_[left/right]

"Z-component of the torque in [Vehicle Reference System](#), acting on a chassis point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].FX_Inner_Engine_[left/right]

"X-component of the force in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].FY_Inner_Engine_[left/right]

"Y-component of the force in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].FZ_Inner_Engine_[left/right]

"Z-component of the force in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].MX_Inner_Engine_[left/right]

"X-component of the torque in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].MY_Inner_Engine_[left/right]

"Y-component of the torque in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

UserOut_differential_[front/rear].MZ_Inner_Engine_[left/right]

"Z-component of the torque in [Vehicle Reference System](#), acting on an engine point having y=0 and x and z equal to inner convol ones"

Note: if [engine part](#) is not defined in the model, the related outputs will be 0.

Driver Demand

Here the list of all **Driver Demand** outputs:

driver_demands:brake

"Brake input pressure at master cylinder (0-100)"

driver_demands:brake_pressure

"Brake input pressure at master cylinder (MPa)"

driver_demands:clutch

"Clutch input (1-open 0-close)"

driver_demands:gear

"Actual gear (-)"

driver_demands:steering

"Steering input: angle of the hand wheel (positive counterclockwise) (radian)"

driver_demands:steering_velocity

"angular velocity of the hand wheel (radian/second)"

driver_demands:throttle

"Engine throttle input (-)"

Driver Monitor

Here the list of all **Driver Monitor** outputs:

driving_machine_monitor:maneuver_ID

"ID number of Driving Machine mini-maneuver that is currently being executed (-)"

driving_machine_monitor:path_curvature

"Curvature of Driving Machine target path evaluated at path_s (1/m)"

driving_machine_monitor:path_distance

"Lateral displacement of vehicle from Driving Machine target path evaluated at path_s (meter)"

driving_machine_monitor:path_s

"Distance traveled along Driving Machine target path (meter)"

driving_machine_monitor:target_ax

"Target Ax for Driving Machine (g's)"

driving_machine_monitor:target_vx

"Target Vx for Driving Machine (km/h)"

driving_machine_monitor:path_x

"Path x position (meter)"

driving_machine_monitor:path_y

"Path y position (meter)"

driving_machine_monitor:path_z

"Path z position (meter)"

driving_machine_monitor:driver_brake

"brake demand applied by the driver (0-100)"

driving_machine_monitor:driver_clutch

"clutch demand applied by the driver (0-1)"

driving_machine_monitor:driver_gear

"gear demand applied by the driver"

driving_machine_monitor:driver_steering

"steering angle demand applied by the driver"

driving_machine_monitor:driver_throttle

"throttle demand applied by the driver (0-100)"

driving_machine_monitor:driver_gear

"gear demand applied by the driver"

driving_machine_monitor:target_engine_torque

"engine torque computed by the driver to follow the target speed/acceleration profile (Newton*meter)"

Engine

Here the list of all **Engine** outputs:

effective_throttle

"Engine effective throttle, including effects of max and idle rpm control (0-100)"

engine_rpm

"Engine rotational velocity (rpm)"

Max_Engine_torque

"Max engine torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"Min engine torque at the current rotational velocity (newton-meter)"

Power_HP

"Engine Power (Hp)"

Power_KW

"Engine Power (KW)"

torque

"Engine torque (newton-meter)"

clutch_inputplate_alpha

"clutch input plate angular acceleration (radian/second**2)"

clutch_inputplate_angle

"clutch input plate rotation angle (radian)"

clutch_inputplate_omega

"clutch input plate angular velocity (radian/second)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Motor

The following outputs are available for each additional motor:

- [Motor to Central Differential](#)
- [Motor to Clutch](#)
- [Motor to Front Differential](#)
- [Motor to Rear Differential](#)
- [Motor to Gearbox](#)
- [Motor to Front Left Wheel](#)
- [Motor to Front Right Wheel](#)
- [Motor to Rear Left Wheel](#)
- [Motor to Rear Right Wheel](#)

Note: For each motor the outputs are present only when such motor is active in powertrain model.

Motor to Central Differential

Here the list of all **Motor to Central Differential** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Clutch

Here the list of all **Motor to Clutch** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Front Differential

Here the list of all **Motor to Front Differential** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Rear Differential

Here the list of all **Motor to Rear Differential** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Gearbox

Here the list of all **Motor to Gearbox** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Front Left Wheel

Here the list of all **Motor to Front Left Wheel** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Front Right Wheel

Here the list of all **Motor to Front Right Wheel** outputs:

differential_inputshaft_alpha

VI-CarRealTime

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Rear Left Wheel

Here the list of all **Motor to Rear Left Wheel** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Motor to Rear Right Wheel

Here the list of all **Motor to Rear Right Wheel** outputs:

differential_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

differential_inputshaft_angle

"central differential input shaft rotation angle (radian)"

differential_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

effective_throttle

"effective throttle demand of motor [0-1]"

engine_rpm

"motor rotational velocity (rpm)"

Max_Engine_torque

"maximum motor torque at the current rotational velocity (newton-meter)"

Min_Engine_torque

"minimum motor torque at the current rotational velocity (newton-meter)"

Power_KW

"motor power (kW)"

torque

"motor torque (newton-meter)"

TCS_activity

"Internal traction control system runtime activation [0/1]"

Theoretical_Power_KW

"theoretical power provided by the motor, neglecting the efficiency coefficient (kW)"

Fuel Consumption

Here the list of all **Fuel Consumption** outputs:

fuel_consumption:BSFC

"Instantaneous fuel consumption according to Brake specific fuel consumption (BSFC) method "

fuel_consumption:instantaneous

"Instantaneous fuel consumption (liter/sec)"

fuel_consumption:instantaneous_Liter_100Km

VI-CarRealTime

"Instantaneous fuel consumption (liter/100Km)"

fuel_consumption:overall

"Overall fuel consumption (liters)"

Gyro

Here the list of all **Gyro** outputs:

Look at [Gyro Reference](#) to know how gyro reference frame is defined.

It is possible to add 4 additional sensor named: *Driver* , *Sensor_1* , *Sensor_2* , *Sensor_3* . Please refer to [sensor advanced topics](#) to know how adding extra sensor points.

Gyro:Driver_X

"Measures the x component of driver marker displacement relative to ground expressed in gyro (meter)"

Gyro:Driver_Z

"Measures the y component of driver marker displacement relative to ground expressed in gyro (meter)"

Gyro:gyro_ACCX

"Measures the x component of gyro acceleration relative to ground expressed in gyro (meter/second^{**2})"

Gyro:gyro_ACCY

"Measures the y component of gyro acceleration relative to ground expressed in gyro (meter/second^{**2})"

Gyro:gyro_VX

"Measures the x component of gyro velocity relative to ground expressed in gyro (meter/second)"

Gyro:gyro_X

"Measures the x component of gyro displacement relative to ground expressed in ground/global coordinates (meter)"

Gyro:gyro_Y

"Measures the y component of gyro displacement relative to ground expressed in ground/global coordinates (meter)"

Gyro:sensor_1_Z

"Measures the z component of sensor #1 marker displacement relative to ground expressed in gyro (meter)"

Gyro:sensor_2_Z

"Measures the z component of sensor #2 marker displacement relative to ground expressed in gyro (meter)"

Gyro:sensor_3_Z

"Measures the z component of sensor #3 marker displacement relative to ground expressed in gyro (meter)"

Global Pad

Here the list of all **Global Pad** outputs:

Note: These outputs are present only in sevenpost analysis. They represent states of test-rig actuator pads.

Global_pad_acc:[front,rear]_[left,right]

"z-actuator acceleration with respect to [std tire reference](#) (meter/second^{**2})".

Global_pad_vel:[front,rear]_[left,right]

"z-actuator velocity with respect to [std tire reference](#) (meter/second)".

Global_pad_disp:[front,rear]_[left,right]

"z-actuator displacement with respect to [std tire reference](#) (meter)".

Global Upright

Here the list of all **Global Upright** outputs:

Note: These outputs are present only in sevenpost analysis. They represent vertical behavior of wheel hubs.

Global_upright_acc:[front,rear]_[left,right]

"z-upright acceleration with respect to [std tire reference](#) (meter/second^{**2})".

Global_upright_vel:[front,rear]_[left,right]

"z-upright velocity with respect to [std tire reference](#) (meter/second)".

Global_upright_disp:[front,rear]_[left,right]

"z-upright displacement with respect to [std tire reference](#) (meter)".

Hook

Here the list of all **Hook** outputs:

Hook_rear:Global_force_X

"X hook force in global frame (newton)"

Hook_rear:Global_force_Y

"Y hook force in global frame (newton)"

Hook_rear:Global_force_Z

"Z hook force in global frame (newton)"

Hook_rear:Global_Moment_X

"X hook moment in global frame (newton*meter)"

Hook_rear:Global_Moment_Y

"Y hook moment in global frame (newton*meter)"

Hook_rear:Global_Moment_Z

"Z hook moment in global frame (newton*meter)"

Hook_rear:Hook_Phi

"hook phi-rotation angle (3rd rotation in Body/313 sequence) relative to global frame (radian)"

Hook_rear:Hook_Psi

"hook psi-rotation angle (1st rotation in Body/313 sequence) relative to global frame (radian)"

Hook_rear:Hook_Theta

"hook theta-rotation angle (2nd rotation in Body/313 sequence) relative to global frame (radian)"

Hook_rear:Hook_X

"X displacement of attachment in global frame (meter)"

Hook_rear:Hook_Y

"Y displacement of attachment in global frame (meter)"

Hook_rear:Hook_Z

"Z displacement of attachment in global frame (meter)"

Hook_rear:Local_force_X

"X attachment force in hook frame (newton)"

Hook_rear:Local_force_Y

"Y attachment force in hook frame (newton)"

Hook_rear:Local_force_Z

"Z attachment force in hook frame (newton)"

Hook_rear:Local_Moment_X

"X attachment moment in hook frame (newton*meter)"

Hook_rear:Local_Moment_Y

"Y attachment moment in hook frame (newton*meter)"

Hook_rear:Local_Moment_Z

"Z attachment moment in hook frame (newton*meter)"

Inerter

Here the list of all **Inerter** outputs:

ins_inerter_data:acceleration:[front,rear]

"central inerter extension acceleration (positive in elongation) (meter/second**2)"

ins_inerter_data:velocity:[front,rear]

"central inerter extension rate (positive in elongation) (meter/second)"

ins_inerter_data:displacement:[front,rear]

"central inerter extension with respect to design condition (positive in elongation) (meter)"

ins_inerter_data:force_at_wheel:[front,rear]

"central inerter force at wheel (newton)"

ins_inerter_data:force:[front,rear]

"central inerter force at inerter (newton)"

Inerter:Force_At_Inerter:[L,R][1,2]

"inerter force at element (newton)"

Inerter:Force_At_Wheel:[L,R][1,2]

"inerter force at wheel (newton)"

Inerter:Jounce:[L,R][1,2]

"inerter deformation (positive in compression) (meter)"

Inerter:Jounce_Acceleration:[L,R][1,2]

"inerter deformation acceleration (positive in compression) (meter/second**2)"

Inerter:Jounce_Rate:[L,R][1,2]

"inerter deformation (positive in compression) (meter/second)"

Kingpin Contributes

These outputs are available only when [kingpin extra output activity](#) flag is set to 1 in the model system tree.

Kingpin_Contribute_Front_[Left,Right].Tire_Force_Fx

"Kingpin contribute due to the tire longitudinal force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Tire_Force_Fy

"Kingpin contribute due to the tire lateral force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Tire_Force_Fz

"Kingpin contribute due to the tire vertical force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Tire_Force_Mx

"Kingpin contribute due to the tire overturning moment (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Tire_Force_My

"Kingpin contribute due to the tire rolling resistance moment (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Tire_Force_Mz

"Kingpin contribute due to the tire aligning moment (Newton-meter)"

Note: tire forces and moments are expressed in [ISO-C](#) reference system.

Kingpin_Contribute_Front_[Left,Right].Gyroscopic

"Kingpin contribute due to the gyroscopic effects (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Weight

"Kingpin contribute due to the unsprung mass weight (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Auxiliary

"Kingpin contribute due to the driveshaft reaction force (Newton-meter)"

The following outputs are computed only when [kingpin inertial contribute activity](#) flag is set to 1, otherwise their value is 0.

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_Fx

"Kingpin contribute due to the longitudinal inertia force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_Fy

"Kingpin contribute due to the lateral inertia force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_Fz

"Kingpin contribute due to the vertical inertia force (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_Mx

"Kingpin contribute due to the overturning inertia moment (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_My

"Kingpin contribute due to the rolling inertia moment (Newton-meter)"

Kingpin_Contribute_Front_[Left,Right].Inertia_Force_Mz

"Kingpin contribute due to the aligning inertia moment (Newton-meter)"

Spring

Here the list of all **Spring** outputs:

ns[l,r]_ride_spring_data:displacement:[front,rear]

"suspension spring deformation (positive in elongation) (meter)"

ns[l,r]_ride_spring_data:force:[front,rear]

"suspension spring force at element (newton)"

ns[l,r]_ride_spring_data:velocity:[front,rear]

"suspension spring elongation velocity (meter/second)"

Spring:Aggregate_Force_At_Wheel:[L,R][1,2]

"sum of suspension main spring, suspension auxiliary spring, bumpstop, and rebound stop forces at wheel (newton)"

Spring:Force_At_Spring:[L,R][1,2]

"suspension spring force at element (newton)"

Spring:Force_At_Wheel:[L,R][1,2]

"suspension spring force at wheel (newton)"

Spring:Force_Aux_At_Wheel:[L,R][1,2]

"suspension auxiliary spring force at wheel (newton)"

Spring:Jounce:[L,R][1,2]

"suspension spring deformation (positive in compression) (meter)"

Spring:Jounce_Acceleration:[L,R][1,2]

"suspension spring deformation acceleration (positive in compression) (meter/second**2)"

Spring:Jounce_Rate:[L,R][1,2]

"suspension spring deformation (positive in compression) (meter/second)"

Spring:PID_CW_At_Spring:[L,R][1,2]

"suspension spring force preload at element for cross weight setup (newton)"

Spring:PID_PH_At_Spring:[L,R][1,2]

"suspension spring force preload at element for pin height setup (newton)"

Spring:PID_RH_At_Spring:[L,R][1,2]

"suspension spring force preload at element for ride height setup (newton)"

nss_ride_spring_data:displacement:[front,rear]

"central single spring deformation (positive in elongation) (meter)"

nss_ride_spring_data:force_at_wheel:[front,rear]

"central single spring force at wheel (Newton)"

nss_ride_spring_data:force:[front,rear]

"central single spring force at element (newton)"

nss_ride_spring_data:PID_PH_At_Spring:[front,rear]

"central spring force preload at element for pin height setup (newton)"

nss_ride_spring_data:velocity:[front,rear]

"central spring elongation velocity (positive in elongation) (meter/second)"

ues_third_spring_data:displacement:[front,rear]

"third spring deformation (positive in elongation) (meter)"

ues_third_spring_data:force:[front,rear]

"third spring force at element (newton)"

ues_third_spring_data:velocity:[front,rear]

"third spring elongation velocity (positive in elongation) (meter/second)"

Rear Steering System

These outputs are available only when auxiliary [rear steering subsystem](#) has been added:

Rear_Steering_System:Kingpin_Moment.total

"Total kingpin moment defined as the sum of left and right kingpin moment contributes (Newton-meter)"

Rear_Steering_System:Kingpin_Moment.left

"Left moment about the left steer axis of the rear suspension (Newton-meter)"

Rear_Steering_System:Kingpin_Moment.right

"Right moment about the right steer axis of the rear suspension (Newton-meter)"

Rear_Steering_System:Rear_TieRod_Force_Left:X

"Force at tierod left to achieve actuator displacement input projected on chassis x direction (newton)"

Rear_Steering_System:Rear_TieRod_Force_Left:Y

"Force at tierod left to achieve actuator displacement input projected on chassis y direction (newton)"

Rear_Steering_System:Rear_TieRod_Force_Left:Z

"Force at tierod left to achieve actuator displacement input projected on chassis z direction (newton)"

Rear_Steering_System:Rear_TieRod_Force_Right:X

"Force at tierod right to achieve actuator displacement input projected on chassis x direction (newton)"

Rear_Steering_System:Rear_TieRod_Force_Right:Y

"Force at tierod right to achieve actuator displacement input projected on chassis y direction (newton)"

Rear_Steering_System:Rear_TieRod_Force_Right:Z

"Force at tierod right to achieve actuator displacement input projected on chassis z direction (newton)"

Rear_Steering_System:Steering_Instant_Axis_Rotation:left

"Rotation angle of the left instant axis. It's computed in initialization by the solver depending on the kinematic of the left wheel. (rad)"

Rear_Steering_System:Steering_Instant_Axis_Rotation:right

"Rotation angle of the right instant axis. It's computed in initialization by the solver depending on the kinematic of the right wheel. (rad)"

Rear_Steering_System:actuator_displacement:left

"Displacement of the left steering actuator (meter)"

Rear_Steering_System:actuator_displacement:right

"Displacement of the right steering actuator (meter)"

Rear_Steering_System:actuator_velocity:left

"Velocity of the left steering actuator (meter)"

Rear_Steering_System:actuator_velocity:right

"Velocity of the right steering actuator (meter)"

Ride Height

Here the list of all **Ride Height** outputs:

rhl_front_setup_sensor_data:ride_height

"Vertical displacement of front left ride height sensor with respect to the road plane (meter)"

rhs_front_setup_sensor_data:ride_height

"Vertical displacement of front right ride height sensor with respect to the road plane (meter)"

rhs_rear_setup_sensor_data:ride_height

"Vertical displacement of rear ride height sensor with respect to the road plane (meter)"

Road

Here the list of all **Road** outputs:

Road.Contact_Flag.[L,R][1,2]

"flag which detects tire/road contact: 1 = contact, 0 = no contact"

Road.Contact_Patch_Global_X.[L,R][1,2]

"X coordinate of contact patch expressed in global frame (meter)"

Road.Contact_Patch_Global_Y.[L,R][1,2]

"Y coordinate of contact patch expressed in global frame (meter)"

Road.Contact_Patch_Global_Z.[L,R][1,2]

"Z coordinate of contact patch expressed in global frame (meter)"

Road.Off_Road_Flag.[L,R][1,2]

"flag which detects when a tire contact patch is out of the road: 1 = off road, 0 = on road"

Sensor

Here the list of all **Sensor** outputs:

Note: These output are relative to vehicle body [sensor point](#).

Sensor:AccX_gyro

"Measure of the x component of sensor point acceleration relative to ground expressed in [gyro_reference](#) (meter/second**2)"

Sensor:AccY_gyro

"Measure of the y component of sensor point acceleration relative to ground expressed in [gyro_reference](#) (meter/second**2)"

Sensor:Ax

"Measure of the x component of s acceleration (including gravity) relative to ground expressed in sensor point frame (meter/second**2)"

Sensor:Ay

"Measure of the y component of s acceleration (including gravity) relative to ground expressed in sensor point frame (meter/second**2)"

Sensor:Az

"Measure of the z component of s acceleration (including gravity) relative to ground expressed in sensor point frame (meter/second**2)"

Sensor:Ax_Without_Gravity

"Measure of the x component of sensor acceleration relative to ground expressed in [gyro_reference](#) (meter/second**2)"

Sensor:Ay_Without_Gravity

"Measure of the y component of sensor acceleration relative to ground expressed in [gyro_reference](#) (meter/second**2)"

Sensor:Az_Without_Gravity

"Measure of the z component of sensor acceleration relative to ground expressed in [gyro_reference](#) (meter/second^{**2})"

Sensor:Global_X

"Measure of the x component of sensor displacement relative to ground expressed in ground/global coordinates (meter)"

Sensor:Global_Y

"Measure of the y component of sensor displacement relative to ground expressed in ground/global coordinates (meter)"

Sensor:Global_Z

"Measure of the z component of sensor displacement relative to ground expressed in ground/global coordinates (meter) (meter)"

Sensor:Vx

"Measure of the x component of sensor point velocity relative to ground expressed in sensor point frame (meter/second)"

Sensor:Vy

"Measure of the y component of sensor point velocity relative to ground expressed in sensor point frame (meter/second)"

Sensor:Vz

"Measure of the z component of sensor point velocity relative to ground expressed in sensor point frame (meter/second)"

Sensor:Vx_gyro

"Measure of the x component of sensor point velocity relative to ground expressed in [gyro_reference](#) (meter/second)"

Sensor:YawRate_gyro

"Measure of the z component of angular velocity relative to ground expressed in [gyro_reference](#) (radian/second)"

Seven Postrig

Here the list of all **Seven Postrig** outputs:

Note: These outputs are available only in [7Post events](#) and represent drive signals for test-rig.

sevenpost:[FL,FR,RL,RR]_preprocessed

"Pad / upright displacement (meter)"

sevenpost:[FL,FR,RL,RR]_target

"Pad / upright target (meter - meter/sec - meter/sec^{**2})"

sevenpost:[F,RL,RR]_aero

"Aero actuator force (including compensation) (newton)"

sevenpost:[F,RL,RR]_aero_target

"Aero actuator force (newton)"

testrig:jack_vertical_force:[front_left/front_right/rear_left/rear_right]

"Resulting vertical force acting on pad (Newton)." "

testrig:wheel_travel:[front_left/front_right/rear_left/rear_right]

"Wheel jounce (positive in compression) (meter)"

testrig:wheel_travel_wrt_design:[front_left/front_right/rear_left/rear_right]

"Wheel jounce with respect to vehicle design condition (positive in compression) (meter)"

Skidplate

Here the list of all **Skidplate** outputs:

Note: These outputs are available only when at least one [skidplate](#) contact point sensor is active.

<contact_point_name>:DX

"Measure of the x component of contact point displacement evaluated as the displacement of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

<contact_point_name>:DY

"Measure of the y component of contact point displacement evaluated as the displacement of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

<contact_point_name>:DZ

"Measure of the z component of contact point displacement evaluated as the displacement of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

<contact_point_name>:VX

"Measure of the x component of contact point velocity evaluated as the velocity of the point in the chassis, with respect to the ground projected in the [road reference system](#)."

<contact_point_name>:VY

"Measure of the y component of contact point velocity evaluated as the velocity of the point in the chassis, with respect to the ground projected in the [road reference system](#)."

<contact_point_name>:VZ

"Measure of the z component of contact point velocity evaluated as the velocity of the point in the chassis, with respect to the ground projected in the [road reference system](#)."

<contact_point_name>:FX

"Measure of the x component of contact point force evaluated as the force of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

<contact_point_name>:FY

"Measure of the y component of contact point force evaluated as the force of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

<contact_point_name>:FZ

"Measure of the z component of contact point force evaluated as the force of the point in the chassis, with respect to the equivalent road point projected in the [road reference system](#)."

Steering

Here the list of all **Steering** outputs:

Steering_System:Rack_Displacement

"Rack displacement to achieve steering input (meter)"

Steering_System:Rack_Force

"Force required at rack to achieve steering input (newton)"

Steering_System:Steering_Wheel_Torque

"Torque required at steering hand wheel to achieve steering input (newton-meter)"

Steering_System:Steering_column_deformation

"relative column deformation (positive counterclockwise) (radian)"

Steering_System:DriveSim_Steering_Feedback_Torque

"Torque value used by VI-DriveSim to control the steering wheel motor (newton-meter).

For Standard Steering System it corresponds to *Steering_System:Steering_Wheel_Torque*.

For Rack Pinion Steering it is equal to *Steering_System:Steering_Wheel_Torque* neglecting the friction and inertia of the modeled flexible elements; these components are computed by the hardware interface of VI-DriveSim."

Steering_System:Steering_System_TieRod_Force_Left:x

"Force at tierod left to achieve steering input projected on chassis x direction (newton)"

Steering_System:Steering_System_TieRod_Force_Left:y

"Force at tierod left to achieve steering input projected on chassis y direction (newton)"

Steering_System:Steering_System_TieRod_Force_Left:z

"Force at tierod left to achieve steering input projected on chassis z direction (newton)"

Steering_System:Steering_System_TieRod_Force_Right:x

"Force at tierod right to achieve steering input projected on chassis x direction (newton)"

Steering_System:Steering_System_TieRod_Force_Right:y

"Force at tierod right to achieve steering input projected on chassis y direction (newton)"

Steering_System:Steering_System_TieRod_Force_Right:z

"Force at tierod right to achieve steering input projected on chassis z direction (newton)"

These outputs are available only when advanced steering system is active:

Steering_column:steering_assist_force/moment

"Steering assist force (moment) powered by electric servo assistance module.

The force, applied to rack, is positive when rack is moving according to a positive hand wheel angle (counterclockwise) (newton)

The torque, applied to column, is positive according to a positive hand wheel angle (newton*meter)"

Steering_column:lower_column_rotation

"absolute steering lower column rotation (positive counterclockwise) (radian)"

Steering_column:lower_column_rotational_velocity

"absolute steering lower column rotational velocity (positive counterclockwise) (radian/second)"

Steering_column:friction_torque

"Upper steering column friction torque (newton*meter)"

Steering_column:upper_cardan_shaft_ratio

"transmission ratio across upper hooke joint (connecting upper and middle column)"

Steering_column:lower_cardan_shaft_ratio

"transmission ratio across lower hooke joint (connecting middle and lower column)"

Steering_column:overall_cardan_shaft_ratio

"overall transmission ratio due to hooke joints (upper and lower column)"

Steering_column:inertial_torque

"Steering column inertial torque: positive according to a positive hand wheel angle (newton*meter)"

Steering_column:steering_wheel_torque

"Steering hand wheel torque: positive according to a positive hand wheel angle (newton*meter)"

Steering_column:lower_column_deformation

"relative lower column deformation (positive counterclockwise) (radian)"

Steering_column:upper_column_deformation

"relative upper column deformation (positive counterclockwise) (radian)"

Steering_column:lower_column_deformation_velocity

"relative lower column deformation velocity (positive counterclockwise) (radian/second)"

Steering_column:upper_column_deformation_velocity

"relative upper column deformation velocity (positive counterclockwise) (radian/second)"

Steering_column:lower_column_stiffness_torque

"overall stiffness torque due to lower column deformation (newton/meter)"

Steering_column:lower_column_damping_torque

"overall damping torque due to lower column deformation velocity (newton/meter)"

Steering_column:upper_column_stiffness_torque

"overall stiffness torque due to upper column deformation (newton/meter)"

Steering_column:upper_column_damping_torque

"overall damping torque due to upper column deformation velocity (newton/meter)"

Steering_column:total_torque_upper_column

"overall torque due to upper column deformation and friction (newton/meter)"

Steering_column:total_torque_lower_column

"overall torque due to lower column deformation and friction (newton/meter)"

Steering_Rack_Pinion:rack_displacement

"Rack displacement, positive when rack is moving according to a positive hand wheel angle (counterclockwise) (meter)"

Steering_Rack_Pinion:rack_displacement_saturated

"Rack displacement saturated according to rack limits, positive when rack is moving according to a positive hand wheel angle (counterclockwise) (meter)"

Steering_Rack_Pinion:rack_velocity

"Rack velocity, positive when rack is moving according to a positive hand wheel angle (counterclockwise) (meter/second)"

Steering_Rack_Pinion:rack_acceleration

"Rack acceleration, positive when rack is moving according to a positive hand wheel angle (counterclockwise) (meter/second²)"

Steering_Rack_Pinion:pinion_rotation

"Pinion rotation, positive according to a positive hand wheel angle (counterclockwise) (meter)"

Steering_Rack_Pinion:pinion_rotational_velocity

"Pinion rotational velocity, positive according to a positive hand wheel angle (counterclockwise) (meter/second)"

Steering_Rack_Pinion:rack_force_from_lower_column

"Force along rack displacement direction coming from steering column, positive when rack is moving according to a positive hand wheel angle (counterclockwise) (newton)"

Steering_Rack_Pinion:rack_friction_force

"Rack friction force (newton)"

Steering_Rack_Pinion:rack_force

"Rack force coming from suspension tie-rods (newton)"

Steering_Torsion_Bar:torsion_bar_twist_angle

"torsion bar deformation (positive counterclockwise) (radian)"

Steering_Torsion_Bar:torsion_bar_twist_angle_saturated

"torsion bar deformation saturated according to torsion bar limits (positive counterclockwise) (radian)"

Steering_Torsion_Bar:friction_torque

"Friction torque applied at torsion bar element (newton*meter)"

Suspension

Here the list of all **Suspension** outputs:

Suspension:Auxiliary_Roll_deformation_at_element:SUSP[1,2]

"Suspension anti roll bar deformation on element (output available only when anti roll bar contribute is not expressed using force table) (rad)"

Suspension:Auxiliary_Roll_Force_at_element:SUSP[1,2]

"Suspension anti roll bar torque at element (output available only when anti roll bar contribute is not expressed using force table) (newton-meter)"

Suspension:Auxiliary_Roll_Force:SUSP[1,2]

"Suspension anti roll bar force at wheel (newton)"

Suspension:Inst_Stiff_Deflection:[front,rear]_[left,right]

"Installation stiffness suspension jounce deflection at wheel (meter)"

Suspension:Inst_Stiff_Force:[front,rear]_[left,right]

"Installation stiffness force at wheel front left (newton)"

Suspension:Jounce:[L,R][1,2]

"Suspension jounce at wheel (positive in compression) (meter)"

Suspension:Jounce_Acceleration:[L,R][1,2]

"Suspension jounce acceleration at wheel (positive in compression) (meter/second**2)"

Suspension:Jounce_Rate:[L,R][1,2]

"Suspension jounce velocity at wheel (positive in compression) (meter/second)"

Suspension:Kingpin_Moment:SUSP[1,2]

"Sum of left- and right-wheel kingpin moments due to tire forces and moments (newton-meter)"

Suspension:Roll:SUSP[1,2]

"Suspension relative roll, positive when sprung mass leans to the right (radian)"

Suspension:base_deformation:[L,R][1,2]

"Base deformation due to compliance (meter)"

Suspension:base_velocity:[L,R[1,2]

"Base deformation velocity due to compliance (meter/second)"

Suspension:base_acceleration:[L,R[1,2]

"Base deformation acceleration due to compliance (meter/second**2)"

Suspension:Roll_Center_Height:[front,rear]

"Roll center height wrt road plane (positive vertical) (meter)"

Suspension:Roll_Lateral_Displacement:[front,rear]

"Lateral displacement of roll center footprint on road plane wrt vehicle midpoint (positive left) (meter)"

Suspension Testrig

Here the list of all **Suspension Testrig** outputs:

Note: These outputs are available only in [Suspension Testrig events](#).

testrig:jack_vertical_force:[front_left/front_right/rear_left/rear_right]

"Resulting vertical force acting on pad (Newton)." "

testrig:wheel_load_align:[front_left/front_right/rear_left/rear_right]

"Wheel Aligning Moment in [ISO-W](#) reference system (Newton-meter)." "

testrig:wheel_load_lat:[front_left/front_right/rear_left/rear_right]

"Wheel Lateral Force in [ISO-W](#) reference system (Newton)." "

testrig:wheel_load_long:[front_left/front_right/rear_left/rear_right]

"Wheel Longitudinal Force in [ISO-W](#) reference system (Newton)." "

testrig:wheel_travel:[front_left/front_right/rear_left/rear_right]

"Wheel jounce (positive in compression) (meter)"

testrig:wheel_travel_wrt_design:[front_left/front_right/rear_left/rear_right]

"Wheel jounce with respect to vehicle design condition (positive in compression) (meter)"

System Threshold

Here the list of all **System Threshold** outputs:

Note: These outputs represent the entities monitored during statics. Static equilibrium will not be achieved until their value will be below the relative threshold value expressed in [statics block of system parameters](#).

System:Angular_Acceleration_Norm

"The square root of the sum of the squares of all rigid-body angular accelerations (radian/second**2)"

System:Angular_Speed_Norm

"The square root of the sum of the squares of all rigid-body angular speeds (radian/second)"

System:Translational_Acceleration_Norm

"The square root of the sum of the squares of all rigid-body translational accelerations (meter/second**2)"

System:Translational_Speed_Norm

"The square root of the sum of the squares of all rigid-body translational speeds (meter/second)"

System Mass

Here the list of all **System Mass** outputs:

Following outputs are relative to the vehicle overall center of mass.

System_Mass_Center:Global_Ax

"The global X-component of the absolute acceleration of the system mass center (with respect to global frame) (g's)"

System_Mass_Center:Global_Ay

"The global Y-component of the absolute acceleration of the system mass center (with respect to global frame) (g's)"

System_Mass_Center:Global_Az

"The global Z-component of the absolute acceleration of the system mass center (with respect to global frame) (g's)"

System_Mass_Center:Global_Vx

"The global X-component of the absolute velocity of the system mass center (with respect to global frame) (km/h)"

System_Mass_Center_Global_Vy

"The global Y-component of the absolute velocity of the system mass center (with respect to global frame) (km/h)"

System_Mass_Center:Global_Vz

"The global Z-component of the absolute velocity of the system mass center (with respect to global frame) (km/h)"

System_Mass_Center:Global_X

"X-coordinate of the system mass center in the global frame (meter)"

System_Mass_Center:Global_Y

"Y-coordinate of the system mass center in the global frame (meter)"

System_Mass_Center:Global_Z

"Z-coordinate of the system mass center in the global frame (meter)"

System_Mass_Center:Normal_Acceleration

"The component of the absolute acceleration of the system mass center that is normal (towards the center of curvature) to the path of the system mass center (g's)"

System_Mass_Center:Path_Curvature

"The instantaneous curvature of the path of the system mass center (1/m)"

System_Mass_Center:Speed

"The magnitude of the absolute velocity of the system mass center (km/h)"

System_Mass_Center:Tangential_Acceleration

"The component of the absolute acceleration of the system mass center that is tangent to the path of the system mass center (g's)"

Tire

Here the list of all **Tire** outputs:

ti[i,r]_wheel_tire_forces:lateral_[front,rear]

"Y-component of tire force in [ISO-W](#) reference system (newton)"

ti[i,r]_wheel_tire_forces:longitudinal_[front,rear]

"X-component of tire force in [ISO-W](#) reference system (newton)"

ti[l,r]_wheel_tire_forces:normal_[front,rear]

"Z-component of tire force in [ISO-W](#) reference system (newton)"

ti[l,r]_wheel_tire_forces:aligning_torque_[front,rear]

"Z-component of tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_forces:overturning_moment_[front,rear]

"X-component of tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_forces:rolling_resistance_[front,rear]

"Y-component of tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_kinematics:inclination_angle_[front,rear]

"Wheel inclination angle relative with respect to ground-surface (radian)"

ti[l,r]_wheel_tire_kinematics:lateral_slip_[front,rear]

"Tire lateral slip in ground-surface frame, with relaxation effects (radian)"

ti[l,r]_wheel_tire_kinematics:longitudinal_slip_[front,rear]

"Tire longitudinal slip in ground-surface frame, with relaxation effects (-)"

ti[l,r]_wheel_tire_rolling_states:omega_actual_[front,rear]

"Wheel angular speed relative to hub carrier, i.e. tire spinning velocity (radian/second)"

ti[l,r]_wheel_tire_rolling_states:omega_free_[front,rear]

"Equivalent angular speed of wheel center. It is computed as hub longitudinal velocity divided by effective rolling radius (radian/second)"

ti[l,r]_wheel_tire_rolling_states:rolling_radius_[front,rear]

"Tire effective rolling radius (meter)"

ti[l,r]_wheel_tire_forces:lateral_twin_rear

"Y-component of twin tire force in [ISO-W](#) reference system (newton)"

ti[l,r]_wheel_tire_forces:longitudinal_twin_rear

"X-component of twin tire force in [ISO-W](#) reference system (newton)"

ti[l,r]_wheel_tire_forces:normal_twin_rear

"Z-component of twin tire force in [ISO-W](#) reference system (newton)"

ti[l,r]_wheel_tire_forces:aligning_torque_twin_rear

"Z-component of twin tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_forces:overturning_moment_twin_rear

"X-component of twin tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_forces:rolling_resistance_twin_rear

"Y-component of twin tire moment in [ISO-W](#) reference system (newton-meter)"

ti[l,r]_wheel_tire_kinematics:inclination_angle_twin_rear

"Twin wheel inclination angle relative with respect to ground-surface (radian)"

ti[l,r]_wheel_tire_kinematics:lateral_slip_twin_rear

"Twin tire lateral slip in ground-surface frame, with relaxation effects (radian)"

ti[l,r]_wheel_tire_kinematics:longitudinal_slip_twin_rear

"Twin tire longitudinal slip in ground-surface frame, with relaxation effects (-)"

ti[l,r]_wheel_tire_rolling_states:omega_actual_twin_rear

"Twin wheel angular speed relative to hub carrier (radian/second)"

ti[l,r]_wheel_tire_rolling_states:omega_free_twin_rear

"Equivalent angular speed of twin wheel center (radian/second)"

ti[l,r]_wheel_tire_rolling_states:rolling_radius_twin_rear

"Twin tire effective rolling radius (meter)"

Tire:Actual_Tread_Speed:[L,R][1,2]

"Tire actual forward speed of tread - Wheel Omega * Loaded Radius - (km/h)"

Tire:Contact_Patch_Normal_X:[L,R][1,2]

"X component of road normal in the standard tire ref system (-)"

Tire:Contact_Patch_Normal_Y:[L,R][1,2]

"Y component of road normal in the standard tire ref system (-)"

Tire:Contact_Patch_Normal_Z:[L,R][1,2]

"Z component of road normal in the standard tire ref system (-)"

Tire:Contact_Patch_Global_X:[L,R][1,2]

"X coordinate of contact patch expressed in global frame (meter). The value becomes 0 when tire contact with road is lost."

Tire:Contact_Patch_Global_Y:[L,R][1,2]

"Y coordinate of contact patch expressed in global frame (meter). The value becomes 0 when tire contact with road is lost."

Tire:Contact_Patch_Global_Z:[L,R][1,2]

"Z coordinate of contact patch expressed in global frame (meter). The value becomes 0 when tire contact with road is lost."

Tire:Effective_Closing_Speed:[L,R][1,2]

"Tire effective vertical speed (meter/second)"

Tire:Effective_Penetration:[L,R][1,2]

"Tire effective penetration (meter)"

Tire:Effective_Rolling_Radius:[L,R][1,2]

"Tire effective rolling radius (meter)"

Tire:Equivalent_Tread_Speed:[L,R][1,2]

"Tire equivalent forward speed of tread -Effective Rolling Radius * Wheel Omega - (km/h)"

Tire:External_Friction_Scale_Factor:[L,R][1,2]

"External factor by which tire/ground friction coefficient is scaled. Ignored unless > 1.0e-5. (-)"

Tire:Ground_Surface_Contact_Patch_Vx:[L,R][1,2]

"X-component of contact-patch velocity expressed in road plane frame (km/h)"

Tire:Ground_Surface_Contact_Patch_Vy:[L,R][1,2]

"Y-component of contact-patch velocity expressed in road plane frame (km/h)"

Tire:Ground_Surface_Force_X:[L,R][1,2]

"X-component of tire force at center of tire contact expressed in road plane frame (newton)"

Tire:Ground_Surface_Force_Y:[L,R][1,2]

"Y-component of tire force at center of tire contact expressed in road plane frame (newton)"

Tire:Ground_Surface_Force_Z:[L,R][1,2]

"Z-component of tire force at center of tire contact expressed in road plane frame (newton)"

Tire:Ground_Surface_Moment_X:[L,R][1,2]

"X-component of tire moment at center of tire contact expressed in road plane frame (newton-meter)"

Tire:Ground_Surface_Moment_Y:[L,R][1,2]

"Y-component of tire moment at center of tire contact expressed in road plane frame (newton-meter)"

Tire:Ground_Surface_Moment_Z:[L,R][1,2]

"Z-component of tire moment at center of tire contact expressed in road plane frame (newton-meter)"

Tire:Ground_Surface_Gyroscopic_Moment:[L,R][1,2]

"Gyroscopic-effects portion of Z-component of tire moment expressed in road plane frame (newton-meter)"

Tire:Ground_Surface_Kappa_Fy:[L,R][1,2]

"Lateral force due to longitudinal slip expressed in road plane frame (newton)"

Tire:Ground_Surface_Peak_Friction_X:[L,R][1,2]

"Peak friction coefficient in X-direction of road plane frame (-). For MF_05, PAC2002 and PAC_CT tire model, this output represents Mux coefficient. In case of VI-Tire model output is computed as Abs(Longitudinal Force / Normal Force)".

Tire:Ground_Surface_Peak_Friction_Y:[L,R][1,2]

"Peak friction coefficient in Y-direction of road plane frame (-). For MF_05, PAC2002 and PAC_CT tire model, this output represents Muy coefficient. In case of VI-Tire model output is computed as Abs(Lateral Force / Normal Force)".

Tire:Ground_Surface_Residual_Torque:[L,R][1,2]

"Residual portion (due to conicity and ply steer) of Z-component of tire moment expressed in road plane frame (newton-meter)"

Tire:Hub_Carrier_Force_Global_X:[L,R][1,2]

"X-component of tire force at wheel center expressed in ground global frame (newton)"

Tire:Hub_Carrier_Force_Global_Y:[L,R][1,2]

"Y-component of tire force at wheel center expressed in ground global frame (newton)"

Tire:Hub_Carrier_Force_Global_Z:[L,R][1,2]

"Z-component of tire force at wheel center expressed in ground global frame (newton)"

Tire:Hub_Carrier_Moment_Global_X:[L,R][1,2]

"X-component of tire moment at wheel center expressed in ground global frame (newton-meter)"

Tire:Hub_Carrier_Moment_Global_Y:[L,R][1,2]

"Y-component of tire moment at wheel center expressed in ground global frame (newton-meter)"

Tire:Hub_Carrier_Moment_Global_Z:[L,R][1,2]

"Z-component of tire moment at wheel center expressed in ground global frame (newton-meter)"

Tire:Hub_Carrier_Force_Local_X:[L,R][1,2]

"X-component of tire force at wheel center expressed in hub carrier frame (newton)"

Tire:Hub_Carrier_Force_Local_Y:[L,R][1,2]

"Y-component of tire force at wheel center expressed in hub carrier frame (newton)"

Tire:Hub_Carrier_Force_Local_Z:[L,R][1,2]

"Z-component of tire force at wheel center expressed in hub carrier frame (newton)"

Tire:Hub_Carrier_Moment_Local_X:[L,R][1,2]

"X-component of tire moment at wheel center expressed in hub carrier frame (newton-meter)"

Tire:Hub_Carrier_Moment_Local_Y:[L,R][1,2]

"Y-component of tire moment at wheel center expressed in hub carrier frame (newton-meter)"

Tire:Hub_Carrier_Moment_Local_Z:[L,R][1,2]

"Z-component of tire moment at wheel center expressed in hub carrier frame (newton-meter)"

Tire:Lateral_Relaxation_Length:[L,R][1,2]

"Tire lateral relaxation length (meter)"

Tire:Lateral_Slip_Without_Lag:[L,R][1,2]

"Tire lateral slip expressed in road plane frame, without relaxation effects (radian)"

Tire:Lateral_Slip_With_Lag:[L,R][1,2]

"Tire lateral slip expressed in road plane frame, with relaxation effects (radian)"

Tire:Loaded_Radius:[L,R][1,2]

"Tire loaded radius (meter)"

Tire:Longitudinal_Relaxation_Length:[L,R][1,2]

"Tire longitudinal relaxation length (meter)"

Tire:Longitudinal_Slip_Without_Lag:[L,R][1,2]

"Tire longitudinal slip in road plane frame, without relaxation effects (-)"

Tire:Longitudinal_Slip_With_Lag:[L,R][1,2]

"Tire longitudinal slip expressed in road plane frame, with relaxation effects (-)"

Tire:Moment_Arm_Of_Fx:[L,R][1,2]

"Moment arm of Fx, expressed in road plane frame, due asymmetric plies in the tire belt. This causes an additional portion of Mz at hub carrier. (meter)"

Tire:Road_Friction:[L,R][1,2]

"Friction coefficient of the road in tire contact patch location"

Tire:Pneumatic_Trail:[L,R][1,2]

"Tire pneumatic trail expressed in road plane frame (meter)"

Tire:Temperature_1[_2 / _3]:[L,R][1,2]

"Output for tire thermic model "

Tire:Pressure:[L,R][1,2]

"Output for tire pressure model "

Tire:User_Input_Contact_Patch_Global_Z_Perturbation:[L,R][1,2]

"Z-coordinate of user input contact patch perturbation expressed in global frame (meter)"

Torque Converter

Here the list of all **Torque Converter** outputs:

capacity_factor

"torque converter K factor (radian/(second * SQRT(Newton-meter)))"

efficiency

"efficiency of the torque converter, defined as the product between speed ratio and torque ratio"

gearbox_inputshaft_alpha

"gearbox input shaft angular acceleration (radian/second**2)"

gearbox_inputshaft_angle

"gearbox input shaft rotation angle (radian)"

gearbox_inputshaft_omega

"gearbox input shaft angular velocity (radian/second)"

inputplate_torque

"torque converter input plate torque (Newton-meter)"

speed_ratio

"ratio of the turbine speed over impeller speed"

torque

"turbine output shaft torque (Newton-meter)"

torque_ratio

"ratio of the turbine torque over impeller torque"

Transmission

Here the list of all **Transmission** outputs:

gear

"Current gear id (-)"

input_rpm

"Transmission input shaft rotational velocity entering gearbox (RPM)"

output_rpm

"Differential input shaft rotational velocity exiting gearbox (RPM)"

ratio

"Total engine-to-differential ratio (primary*gear)"

torque

"torque transmitted by the transmission (newton-meter)"

clutch_outputplate_torque

"torque transmitted by the clutch (newton-meter)"

diff_central_inputshaft_alpha

"central differential input shaft angular acceleration (radian/second**2)"

diff_central_inputshaft_angle

"central differential input shaft rotation angle (radian)"

diff_central_inputshaft_omega

"central differential input shaft angular velocity (radian/second)"

automatic_gearbox_shifting_mode

"detect the gearbox shifting mode. 0=comfort, 1=sport"

Upright Accelerometer

Here the list of all **Upright Accelerometer** outputs:

upright_accelerometer:Z_[left,right]_[front,rear]

"Vertical global acceleration of suspension hub expressed in the global frame (meter/second**2)"

Vehicle

Here the list of all **Vehicle** outputs:

Vehicle:Body_Fixed_Alpha_X

"X-component of chassis vehicle angular acceleration expressed in vehicle reference frame (radian/second**2)"

Vehicle:Body_Fixed_Alpha_Y

"Y-component of chassis vehicle angular acceleration expressed in vehicle reference frame (radian/second**2)"

Vehicle:Body_Fixed_Alpha_Z

"Z-component of chassis vehicle angular acceleration expressed in vehicle reference frame (radian/second**2)"

Vehicle:Body_Fixed_Omega_X

"X-component of chassis angular velocity expressed in vehicle reference frame (radian/second)"

Vehicle:Body_Fixed_Omega_Y

"Y-component of vehicle angular velocity expressed in vehicle reference frame (radian/second)"

Vehicle:Body_Fixed_Omega_Z

"Z-component of vehicle angular velocity expressed in vehicle reference frame (radian/second)"

Vehicle:CM_Body_Fixed_Ax

"X-component of chassis mass center acceleration expressed in vehicle reference frame (g's)"

Vehicle:CM_Body_Fixed_Ay

"Y-component of chassis mass center acceleration expressed in vehicle reference frame (g's)"

Vehicle:CM_Body_Fixed_Az

"Z-component of chassis mass center acceleration expressed in vehicle reference frame (g's)"

Vehicle:CM_Body_Fixed_Vx

"X-component component of chassis mass center velocity expressed in vehicle reference frame (km/h)"

Vehicle:CM_Body_Fixed_Vy

"Y-component component of chassis mass center velocity expressed in vehicle reference frame (km/h)"

Vehicle:CM_Body_Fixed_Vz

"Z-component component of chassis mass center velocity expressed in vehicle reference frame (km/h)"

Vehicle:CM_Global_And_Tracking_Az

"Z-component of chassis mass center acceleration in global and vehicle-tracking frames (relative to Z ground direction) (g's)"

Vehicle:CM_Global_And_Tracking_Vz

"Z-component of chassis mass center velocity in global and vehicle-tracking frames (relative to Z ground direction) (km/h)"

Vehicle:CM_Global_Ax

"X-component of chassis mass center acceleration in global frame (g's)"

Vehicle:CM_Global_Ay

"Y-component of chassis mass center acceleration in global frame (g's)"

Vehicle:CM_Global_Vx

"X-component of chassis mass center velocity in global frame (km/h)"

Vehicle:CM_Global_Vy

"Y-component of chassis mass center velocity in global frame (km/h)"

Vehicle:CM_Global_X

"X-coordinate of chassis mass center in global frame (meter)"

Vehicle:CM_Global_Y

"Y-coordinate of chassis mass center in global frame (meter)"

Vehicle:CM_Global_Z

"Z-coordinate of chassis mass center in global frame (meter)"

Vehicle:CM_Tracking_Ax

"X-component of chassis mass center acceleration in vehicle-tracking frame - Z positive up X positive forward - (g's)"

Vehicle:CM_Tracking_Ay

"Y-component of chassis mass center acceleration in vehicle-tracking frame - Z positive up X positive forward - (g's)"

Vehicle:CM_Tracking_Vx

"X-component of chassis mass center velocity in vehicle-tracking frame - Z positive up X positive forward - (km/h)"

Vehicle:CM_Tracking_Vy

"Y-component of chassis mass center velocity in vehicle-tracking frame - Z positive up X positive forward - (km/h)"

Vehicle:cross_weight_PID

"Preload value of the PID controller used to achieve cross weight adjustment (Newton)"

Vehicle:Euler_Roll

"Vehicle roll-rotation angle (3rd rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle:front_ride_height_at_center_point

"Height with respect to road plane of the intersection of line through the ride height front sensor and the vehicle plan at y=0 (meter)"

Vehicle:Global_Alpha_X

"X-component of vehicle angular acceleration in global frame (radian/second**2)"

Vehicle:Global_Alpha_Y

"Y-component of vehicle angular acceleration in global frame (radian/second**2)"

Vehicle:Global_Alpha_Z

"Z-component of vehicle angular acceleration in global frame (radian/second**2)"

Vehicle:Global_Omega_X

"X-component of vehicle angular velocity in global frame (radian/second)"

Vehicle:Global_Omega_Y

"Y-component of vehicle angular velocity in global frame (radian/second)"

Vehicle:Global_Omega_Z

"Z-component of vehicle angular velocity in global frame (radian/second)"

Vehicle:Inclination_Roll

"Chassis roll angle (inclination from the opposite of the direction of gravity) relative to global frame (radian)"

Vehicle:Origin_Distance_Traveled

"Total distance traveled by vehicle origin. This is the time integral of the velocity. (meter)"

Vehicle:Origin_Global_X

"X-coordinate of [vehicle origin](#) in global frame (meter)"

Vehicle:Origin_Global_Y

"Y-coordinate of [vehicle origin](#) in global frame (meter)"

Vehicle:Origin_Global_Z

"Z-coordinate of [vehicle origin](#) in global frame (meter)"

Vehicle:Pitch

"Vehicle pitch-rotation angle (2nd rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle:Side_Slip_Angle

"Vehicle side-slip angle computed as $\text{Atan}(\text{vy_cm_tracking}, \text{vx_cm_tracking})$ (radian)"

Vehicle:Side_Slip_Angle_Rate

"Time derivative of vehicle side-slip angle (radian/second)"

Vehicle:Understeer_Stability_Factor

"A handling-stability factor related to the vehicle understeer gradient. It is valid only for a constant-speed turn on a flat surface that is normal to gravity.

It is computed as follow:

$$\frac{(\text{Steer At Ground} * \text{cm_longitudinal_vel_in_tracking_system} - \text{wheelbase} * \text{yaw_rate})}{(\text{wheelbase} * \text{yaw_rate} * \text{cm_longitudinal_vel_in_tracking_system}^2)}$$
 (rad-s²/m²)

Vehicle:Understeer_Gradient

"An approximation for the vehicle understeer gradient. It is valid only for a constant-speed turn on a flat surface that is normal to gravity.

It is computed as follows:

$\text{Vehicle:Understeer_Stability_Factor} * \text{wheelbase}$ (deg/g)"

Vehicle:Yaw

"Vehicle yaw-rotation angle (1st rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle States

Here the list of all **Vehicle States** outputs:

Note: The gyro reference system (gyro marker) is located on the body (by default it is located coincident to [sensor_point](#)) and is oriented with the Z axis coincident with global Z during the simulation with the X axis along the vehicle movement direction, so it follows the vehicle X,Y,Z displacements, the yaw angle but has always zero pitch and roll angles.

Vehicle_States:horizontal_vel_wrt_road

"Velocity in the plane XY ($\sqrt{vx^{**2} + vy^{**2}}$) of chassis gyro marker (meter/second)"

Vehicle_States:lateral_acc_wrt_road

"Y-component of chassis gyro marker acceleration in road frame (meter/second **2)"

Vehicle_States:lateral_disp

"Y-coordinate of chassis gyro marker in the ground reference (meter)"

Vehicle_States:lateral_vel_wrt_road

"Y-component of chassis gyro marker velocity in road frame (meter/second)"

Vehicle_States:longitudinal_acc_wrt_road

"X-component of chassis gyro marker acceleration in road frame (meter/second **2)"

Vehicle_States:longitudinal_disp

"X-coordinate of chassis gyro marker in the ground reference (meter)"

Vehicle_States:longitudinal_vel_wrt_road

"X-component of chassis gyro marker velocity in road frame (meter/second)"

Vehicle_States:pitch_angle

"Vehicle pitch-rotation angle (1st rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle_States:pitch_angular_acc_wrt_road

"Y-component of vehicle angular acceleration in road frame (radian/second **2)"

Vehicle_States:pitch_angular_vel_wrt_road

"Y-component of vehicle angular velocity in road frame (radian/second)"

Vehicle_States:roll_angle

"Vehicle roll-rotation angle (1st rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle_States:roll_angular_acc_wrt_road

"X-component of vehicle angular acceleration in road frame (radian/second **2)"

Vehicle_States:roll_angular_vel_wrt_road

"X-component of vehicle angular velocity in road frame (radian/second)"

Vehicle_States:side_slip

"vehicle side slip at chassis gyro marker -
atan(Vehicle_States:lateral_vel_wrt_road, Vehicle_States:longitudinal_vel_wrt_road) - (radian)"

Vehicle_States:vertical_acc_wrt_road

"Z-component of chassis gyro marker acceleration in road frame (meter/second **2)"

Vehicle_States:vertical_disp

"Z-coordinate of chassis gyro marker in the global frame (meter)"

Vehicle_States:vertical_vel_wrt_road

"Z-component of chassis gyro marker velocity in road frame (meter/second)"

Vehicle_States:yaw_angle

"Vehicle yaw rotation angle (1st rotation in Body/321 sequence) relative to global frame (radian)"

Vehicle_States:yaw_angular_acc_wrt_road

"Z-component of vehicle angular acceleration in road frame (radian/second **2)"

Vehicle_States:yaw_angular_vel_wrt_road

"Z-component of vehicle angular velocity in road frame (radian/second)"

Wheel

Here the list of all **Wheel** outputs:

Wheel:Alpha:[L,R][1,2]

"Wheel angular acceleration relative to hub carrier (radian/second**2)"

Wheel:Caster_Angle:[left,right]

"Steer axis caster angle relative to chassis (radian)"

Wheel:Drive_Moment:[L,R][1,2]

"Driving torque applied to wheel (newton-meter)"

Wheel:Euler_Roll:[L,R][1,2]

"Hub carrier roll rotation angle relative to road plane (3rd rotation in Body/321 sequence) (radian)"

Wheel:Global_Omega_X:[L,R][1,2]

"X-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Global_Omega_Y:[L,R][1,2]

"Y-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Global_Omega_Z:[L,R][1,2]

"Z-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Global_Vx:[L,R][1,2]

"X-component of wheel center velocity with respect to [standard tire reference maker](#) in ground (km/h)"

Wheel:Global_Vy:[L,R][1,2]

"Y-component of wheel center velocity with respect to [standard tire reference maker](#) in ground (km/h)"

Wheel:Global_Vz:[L,R][1,2]

"Z-component of wheel center velocity with respect to [standard tire reference maker](#) in ground (km/h)"

Wheel:Global_X:[L,R][1,2]

"X-coordinate of wheel center with respect to [standard tire reference maker](#) in ground (meter)"

Wheel:Global_Y:[L,R][1,2]

"Y-coordinate of wheel center with respect to [standard tire reference maker](#) in ground (meter)"

Wheel:Global_Z:[L,R][1,2]

"Z-coordinate of wheel center with respect to [standard tire reference maker](#) in ground (meter)"

Wheel:Ground_Surface_Wheel_Center_Vx:[L,R][1,2]

"X-component of wheel center velocity with respect to road plane (km/h)"

Wheel:Hub_Carrier_Omega_X:[L,R][1,2]

"X-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Hub_Carrier_Omega_Y:[L,R][1,2]

"Y-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Hub_Carrier_Omega_Z:[L,R][1,2]

"Z-component of suspension hub carrier angular velocity in global frame (radian/second)"

Wheel:Hub_Carrier_Vx:[L,R][1,2]

"X-component of wheel center velocity in hub-carrier frame (km/h)"

Wheel:Hub_Carrier_Vy:[L,R][1,2]

"Y-component of wheel center velocity in hub-carrier frame (km/h)"

Wheel:Hub_Carrier_Vz:[L,R][1,2]

"Z-component of wheel center velocity in hub-carrier frame (km/h)"

Wheel:hub_center_accx:[front,rear]_[left,right]

"Wheel center absolute acceleration, projected in X-axis of [upright frame](#) (meter/second**2)"

Wheel:hub_center_accy:[front,rear]_[left,right]

"Wheel center absolute acceleration, projected in Y-axis of [upright frame](#) (meter/second**2)"

Wheel:hub_center_accz:[front,rear]_[left,right]

"Wheel center absolute acceleration, projected in Z-axis of [upright frame](#) (meter/second**2)"

Wheel:Inclination:[L,R][1,2]

"Wheel inclination angle relative to road plane (radian)"

Wheel:Kingpin_Moment:[L,R]1

"Total moment about kingpin due to tire forces and moments (positive about steer-axis direction from ground to sky) (newton-meter)"

Wheel:Omega:[L,R][1,2]

"Wheel angular speed relative to hub carrier (radian/second)"

Wheel:Pitch:[L,R][1,2]

"Hub carrier pitch rotation angle with respect to road plane (2nd rotation in Body/321 sequence) (radian)"

Wheel:Rotation:[L,R][1,2]

"Wheel rotation angle relative to hub carrier (radian)"

Wheel:Side_View_Angle:[L,R][1,2]

"Hub carrier side view angle with respect to chassis (positive for a counterclockwise rotation as seen from the left side of the vehicle) (radian)"

Wheel:Spindle_Steer:[L,R][1,2]

"Wheel steer angle at spindle with respect to chassis (radian)"

Wheel:wheel_center_dispx:[front,rear]_[left,right]

"Delta X-coordinate of wheel center in chassis frame due to elasto-kinematic contribute (meter)"

Wheel:wheel_center_dispy:[front,rear]_[left,right]

"Delta Y-coordinate of wheel center in chassis frame due to elasto-kinematic contribute (meter)"

Wheel:wheel_center_dispz:[front,rear]_[left,right]

"Delta Z-coordinate of wheel center in chassis frame due to elasto-kinematic contribute (meter)"

Wheel:Yaw:[L,R][1,2]

"Hub carrier yaw rotation angle (1st rotation in Body/321 sequence) (radian)"

Wheel:Phi:[L,R][1,2]

"Vehicle phi-rotation angle (3rd rotation in Body/313 sequence) relative to model animation frame (radian)"

Wheel:Psi:[L,R][1,2]

"Vehicle psi-rotation angle (1st rotation in Body/313 sequence) relative to model animation frame (radian)"

Wheel:Theta:[L,R][1,2]

"Vehicle theta-rotation angle (2nd rotation in Body/313 sequence) relative to model animation frame (radian)"

The following outputs are available only when a [Suspension Testrig events](#) or a [7Post events](#) is run:

Wheel:Wheel_Travel_base:[front_left,front_right,rear_left,rear_right]

"Delta X wheel base variation with respect to design condition. Positive toward negative X axis of [global reference frame](#)."

Wheel:Wheel_Travel_track:[front_left,front_right,rear_left,rear_right]

"Delta Y wheel track variation with respect to design condition. Positive toward negative Y axis of [global reference frame](#)."

Wheel Angles

Here the list of all **Wheel Angles** outputs:

wheel_angles:camber_[L,R][1,2]

"hub-carrier camber rotation angle with respect to chassis part (radian)"

wheel_angles:caster_[L,R][1,2]

"hub-carrier side view angle rotation angle with respect to chassis part (radian)"

wheel_angles:toe_[L,R][1,2]

"hub-carrier toe angle rotation with respect to chassis part (radian)"

wheel_angles:camber_wrt_road_[L,R][1,2]

"hub-carrier camber rotation angle with respect to road plane (radian)"

wheel_angles:caster_wrt_road_[L,R][1,2]

"hub-carrier side view angle rotation angle with respect to road plane (radian)"

wheel_angles:toe_wrt_road_[L,R][1,2]

"hub-carrier toe angle rotation with respect to road plane (radian)"

Graphics Part and Force

Here the list of all **Graphics Part and Force** outputs:

Note: These measure are relative to a marker (animation marker) belonging to ground, with Z vertical upwards, X longitudinal backwards, Y lateral (to right).

ges_chassis_XFORM:Phi

"Vehicle phi-rotation angle (3rd rotation in Body/313 sequence) relative to model animation frame (radian)"

ges_chassis_XFORM:Psi

"Vehicle psi-rotation angle (1st rotation in Body/313 sequence) relative to model animation frame (radian)"

ges_chassis_XFORM:Theta

"Vehicle theta-rotation angle (2nd rotation in Body/313 sequence) relative to model animation frame (radian)"

ges_chassis_XFORM:X

"X-coordinate of vehicle origin in model animation frame (meter)"

ges_chassis_XFORM:Y

"Y-coordinate of vehicle origin in model animation frame (meter)"

ges_chassis_XFORM:Z

"Z-coordinate of vehicle origin in model animation frame (meter)"

wh[l,r][front,rear]_wheel_XFORM:FX

"X-component of tire force expressed in model animation frame (newton)"

wh[l,r][front,rear]_wheel_XFORM:FY

"Y-component of tire force expressed in model animation frame (newton)"

wh[l,r][front,rear]_wheel_XFORM:FZ

"Z-component of tire force expressed in model animation frame (newton)"

wh[l,r][front,rear]_wheel_XFORM:Phi

"rear twin tire spindle marker phi-rotation angle (3rd rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r][front,rear]_wheel_XFORM:Psi

"rear twin tire spindle marker psi-rotation angle (1st rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r][front,rear]_wheel_XFORM:Theta

"rear twin tire spindle marker theta-rotation angle (2nd rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r][front,rear]_wheel_XFORM:X

"X-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

wh[l,r][front,rear]_wheel_XFORM:Y

"Y-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

wh[l,r][front,rear]_wheel_XFORM:Z

"Z-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

wh[l,r]_twin_rear_wheel_XFORM:FX

"X-component of tire force expressed in model animation frame (newton)"

wh[l,r]_twin_rear_wheel_XFORM:FY

"Y-component of tire force expressed in model animation frame (newton)"

wh[l,r]_twin_rear_wheel_XFORM:FZ

"Z-component of tire force expressed in model animation frame (newton)"

wh[l,r]_twin_rear_wheel_XFORM:Phi

"rear twin tire spindle marker phi-rotation angle (3rd rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r]_twin_rear_wheel_XFORM:Psi

"rear twin tire spindle marker psi-rotation angle (1st rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r]_twin_rear_wheel_XFORM:Theta

"rear twin tire spindle marker theta-rotation angle (2nd rotation in Body/313 sequence) relative to animation frame (radian)"

wh[l,r]_twin_rear_wheel_XFORM:X

"X-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

wh[l,r]_twin_rear_wheel_XFORM:Y

"Y-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

wh[l,r]_twin_rear_wheel_XFORM:Z

"Z-coordinate of rear twin tire spindle marker expressed in model animation frame (meter)"

1.3.3 Matlab

VI-CarRealTime allows the user to connect their vehicle models with MATLAB environment and use MATLAB facilities to work together with VI-CarRealTime core in order to build up more complex models and carry out simulation results; otherwise, it is possible to rely on MATLAB language and on ad-hoc MATLAB API to manage the main VI-CarRealTime files, run analyses and review results.

The first approach is the co-Simulation between VI-CarRealTime and Simulink ([Matlab/Simulink Interface](#)).

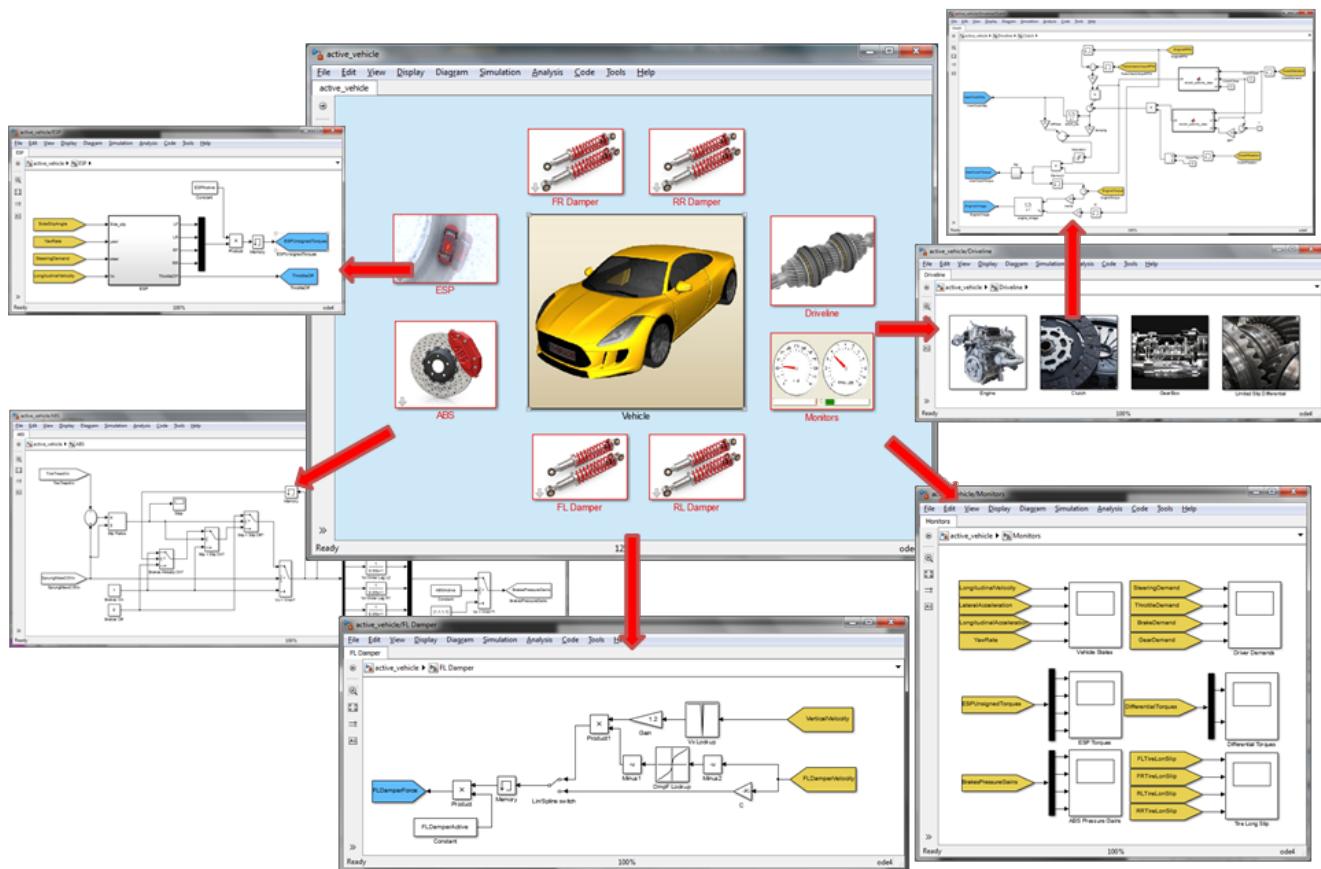
The second approach is the use of [Matlab API Library](#) to interact with VI-CarRealTime models.

The third approach is [Solver Plugin Export from Simulink](#).

A mex function called [resreader](#) is available in order to import result file in MATLAB environment.

Matlab/Simulink Interface

The VI-CarRealTime interface enables user to easily connect their vehicle model with controls systems or accessories developed in MATLAB/Simulink.



```

Command Window
Using Simulation File: VI_CRT_Demo_vdd_send_svm.xml

Model data is being read from input XML file...
WARNING: Problems retrieving Output Map from input xml file, skipped.
Initializing Engine Module

Performing the simulation: VI_CRT_Demo_vdd.

Initializing powertrain / driveline...
Done.

Initializing aero force...
Done.

Initializing tires...
Done.

Initializing driver...
Done.

Solving for static equilibrium...

Static equilibrium has been achieved.
Elapsed Simulation Time: 1.633 (sec).
Elapsed Clock Time: 0.608 (sec).

A Driving-Machine end condition was triggered: simulation terminated prior to simulation abort time.

Writing output to file. Please Wait ...
Elapsed Simulation Time: 10.000 (sec).
Elapsed Clock Time: 5.477 (sec).
Computational Efficiency: 1.828 (sim. sec)/(sec).

Simulation is complete.
f2 >>

```

Introduction

In order to use this interface, an installation of MATLAB and Simulink is needed (no special additional toolbox is required).

The VI-CarRealTime solver interface enables users to easily connect their vehicle model with controls systems or accessories developed in MATLAB/Simulink.

The VI-CarRealTime car model is made available in the MATLAB/Simulink environment as an s-function. The s-function representing the car plant has inputs and outputs that can be connected to other blocks in the Simulink. The VI-CarRealTime car data is retrieved from a file that is passed to the s-function as a parameter.

Some Simulink-VI-CarRealTime example models are available in the VI-CarRealTime installation directory, in the **examples/simulink** sub folder.

In order to include the VI-CarRealTime s-function into an existing MATLAB/Simulink model, open the *vicrt* library included in the distribution by entering "**vicrt_lib**" in the MATLAB command line. Just like with any other Simulink Library you can now copy the s-function and connection blocks into your model.

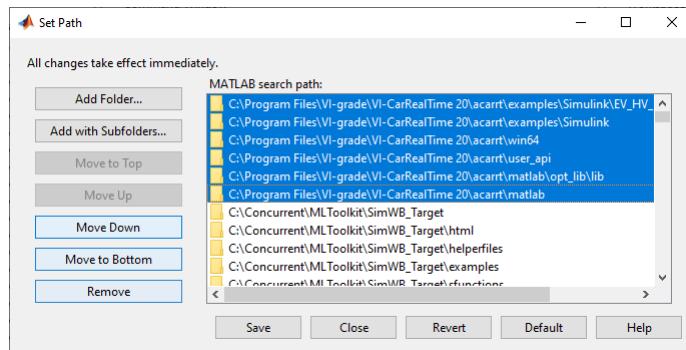
Note: press F5 button to refresh Simulink Library Browser if you don't see the *VI-CarRealTime Library*.

The inputs and the outputs to and from the s-function are completely configurable ([see interface overview](#)).

Setup

In order to connect VI-CarRealTime with MATLAB the user has to add the paths of the VI-CarRealTime libraries to the MATLAB search path. To automate this operation, the installation procedure put in the User\$\UserName\$\Documents\MATLAB directory the "**addpath_vicrt_20.m**" script. This function add to the MATLAB search path the VI-CarRealTime directories only for the current session, and is always available from the MATLAB command window.

In order to keep the VI-CarRealTime paths saved, after calling the "addpath_vicrt_20" function, the MATLAB search path list must be saved using the **Set Path** window from **File** menu.



[Modeling](#)

[Simulation setup](#)

[Running Simulations](#)

[Input File](#)

[Four wheel steering vehicles](#)

Modeling

The VI-CarRealTime vehicle model is made available in the MATLAB/Simulink environment as an s-function. The s-function representing the vehicle plant has inputs and outputs that can be connected to other blocks in Simulink.

The VI-CarRealTime vehicle data is communicated to the s-function at runtime through a socket connection with the VI-CarRealTime GUI or retrieved from files. The two available modes to feed the vehicle s-function are described in more detail further below.

In order to include the VI-CarRealTime s-function into an existing MATLAB/Simulink model open the **vicrt** library included in the VI-CarRealTime distribution by entering "**vicrt_lib**" in the MATLAB command line. Just like with any other Simulink Library you can now copy the s-function and connection blocks into your model.

The **inputs** and the **outputs** to and from the s-function are completely configurable ([see interface overview](#)).

Simulation setup

To setup the vehicle and event you should use the standard approach in the VI-CarRealTime GUI as if you would prepare a standalone simulation.

Note: The interface with MATLAB/Simulink supports the Driving Machine, VI-Driver and Predefined event classes.

Running Simulations

When vehicle model, event and MATLAB/Simulink models are completed it is possible to run the simulation.

Input File

To use this option the simulation input file for a specific event have to be created using the *files only* option in Test Mode.

The file <event_name>_send_svm.xml is created.

The VI-CarRealTime Simulink solver input file `<event_name>_send_svm.xml` can be provided through a string variable called **vicrt_inputfile** or through the mask field **Input file** ([see Interface overview](#)).



When the Simulink computation starts the xml file is used to retrieve the model and event parameters. The simulation is then started from MATLAB/Simulink using the standard methods.

Notes.

It is also possible to define the simulation output prefix independently from what has been set in maneuver definition (fingerprint and send_svm file) through the mask field **Output prefix** ([see Interface overview](#)).

Please note that the `vicrt_inputfile` variable isn't set in the Simulink mdl whenever such variable is defined inside a function. This is because the variables defined inside a function are not part of the base workspace used by Simulink during the simulation. A possible way of act, if a user wants to write the statements for running a co-simulation into a function, is to create a function like the following one:

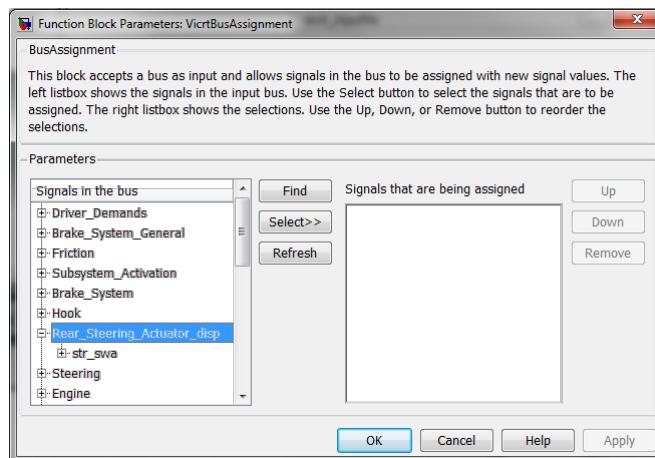
```
function simModel(model, sendFile)
load_system(model)
vicrtMexBlock = get_param(find_system(model,'Look Under Masks','all','Name','vicrt_mex'), 'parent');
vicrtMexBlock=vicrtMexBlock{end};
set_param(vicrtMexBlock, 'vicrt_inputfile', strcat("",sendFile,""));
sim(model, 'Src Workspace','current');
close_system(model, 0);
end
```

where `model` is the Simulink mdl model and `sendFile` is the *send_svm.xml file.

Such function set the desired send_svm.xml file into the VI-CarRealTime Simulink mask and then run the co-simulation by using the input model.

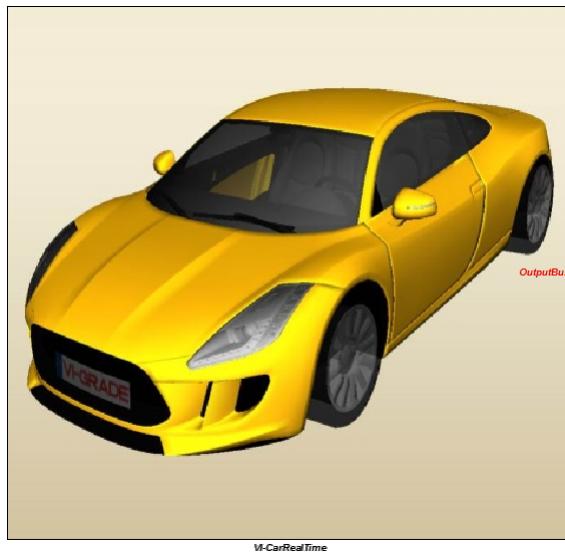
Four wheel steering vehicles

VI-CarRealTime models generated using Adams Car plugin including the four wheel steering option are to be used in conjunction with MATLAB/Simulink by connecting the rear steering controller to the appropriate input channel of Inputs block.



Interface overview

The VI-CarRealTime MATLAB/Simulink interface is now an unique Simulink block that can be completely configured to set up a co-simulation or run the VI-CarRealTime solver directly from Simulink.



The block provides to the user a useful interface through which is possible to set:

- **Input file (xml)**

References `send_svm.xml` file needed by the VI-CarRealTime solver to run the simulation.

- **Output prefix**

When specified, it allows to rename the output file results; otherwise `output_prefix` string parameter defined in the `send_svm.xml` file is used.

- **Integration step**

Sets VI-CarRealTime solver integration step, that must be compatible to the Simulink one by a proportional relation.

- **Remove Algebraic Loop**

When the flag is unchecked, it sets the VI-CarRealTime block mode to direct feed-through. For all the details please refer to Simulink documentation (`direct feedthrough` toggle button in Memory block).

- **Live Animation**

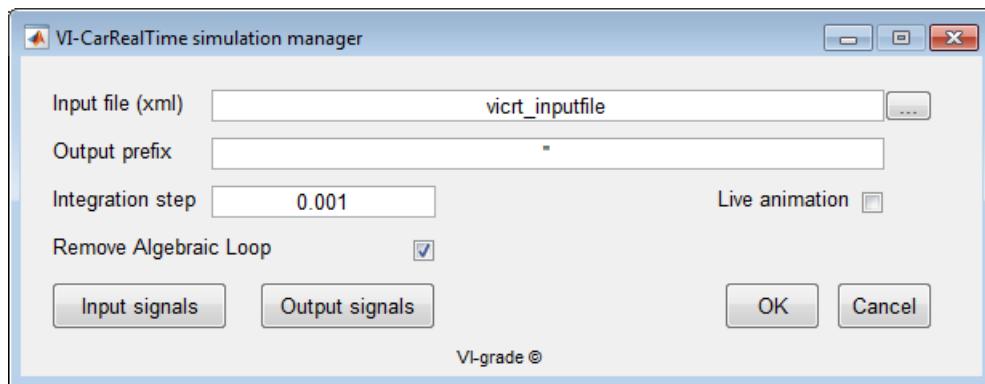
Activation flag for the live animation when running the co-simulation.

- **Input signals**

Button which lists the input channels that will be used for co-simulation. Input file (xml) must be set in order for the button to display the inputs available.

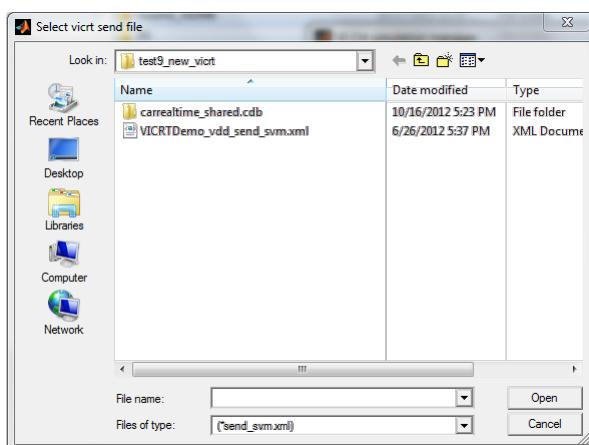
- **Output signals**

Button which lists the output channels that will be used for co-simulation. Input file (xml) must be set in order for the button to display the outputs available.



The simulation **Input file (xml)** can be set in three different ways:

1. using a workspace variable, that must be written in the "input file" text field (`vicrt_inputfile` by default);
2. writing the full path of the input file (or the file name if it is in the working directory) in the "Input file (xml)" text field and pushing the enter button;
3. using the file explorer through the button placed near the text field ().



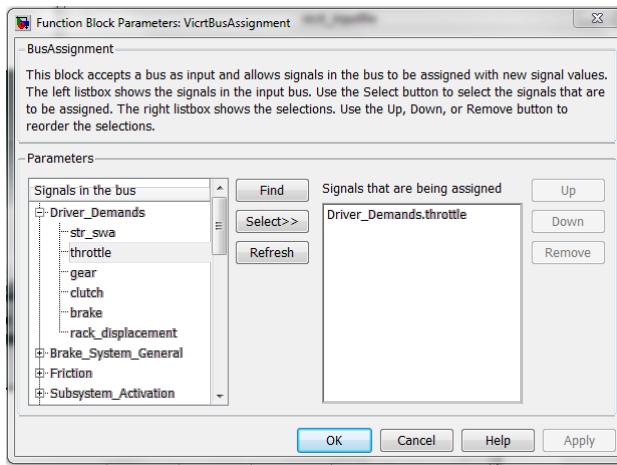
After the input file selection, the interface automatically updates the I/O bus according to the current solver information.

The interface keeps trace of the input file associated to each Simulink model that uses the VI-CarRealTime MATLAB interface and, when it is launched, checks if there are modifications in the input file for the current model.

If the input file is set using a variable, the interface does an additional check in order to retrieve the new input file if the variable value is changed.

Regarding the Input channels selection, using the "input signals" button, it is possible to select from a hierarchical list the input channels to be overwritten ([see input map](#)). After selecting the channels from the left list, using the "Select>>" button the signal is put into the right list. The selected signals can be reordered using the "Up" and "Down" buttons or removed using the "Remove" one.

Clicking on the "OK" button complete the input channels selection and return to the main mask.

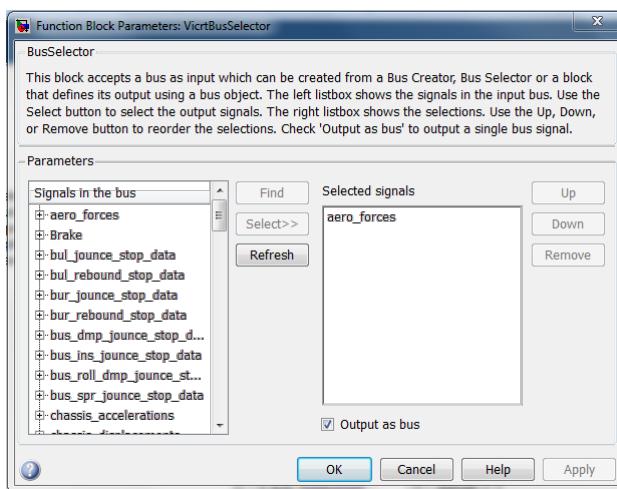


Regarding the output channels selection, using the "Output signal" button, it is possible to select the signals of interest from a list, ordered with the same hierarchy of the result file ([see output map](#)). It must be highlighted that while the signal names reflect what is specified in the [Output Channels](#) map, **Scale Factor** and **Offset** parameters doesn't affect the output of the VI-CarRealTime Simulink block.

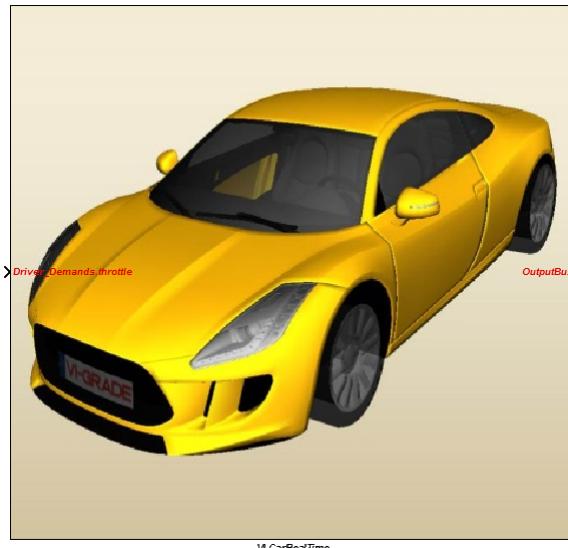
After the signal selection (done in the same way of the inputs), using the "Output as bus" flag, it is possible to set if the output must be provided as a list of channels (one port for each signal named with its name) or by an unique bus channel (an unique port named "Output bus" providing a bus).

Clicking on the "OK" button complete the output channels selection and return to the main mask.

Note: Simulink constraints impose that at least one signal must be selected in the output bus selector. In addition, if the "Output as bus" option is set, the channel names will be generated from the original one removing the prefixes describing the hierarchy. This behavior could generate some warning if the 2 or more signals with the same suffix are selected. In that case, each signal could be identified using its position into the generated bus.



Clicking the "OK" button on the VI-CarRealTime solver mask complete the I/O channels selection, displaying the I/O ports in the VI-CarRealTime block.



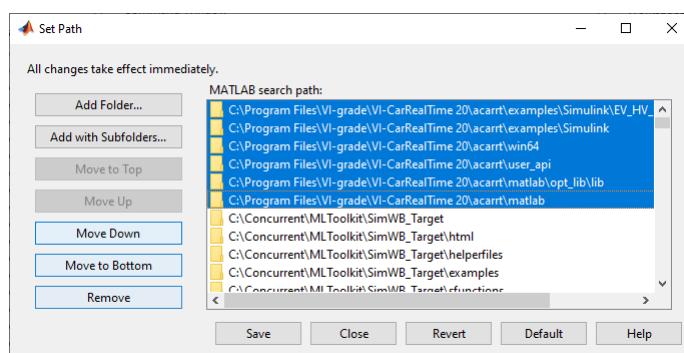
VI-CarRealTime Solver Plugin Export from Simulink

A Simulink model can be exported as a VI-CarRealTime external plug-in to extend/enhance the model functionalities. This section describes how a Simulink model can be exported as a plug-in. Code is generated by Simulink Coder/Real-Time Workshop and is then wrapped in a VI-CarRealTime plug-in.

In order to use this feature, an installation of MATLAB and Simulink is needed, including a Simulink Coder/Real-Time Workshop license.

To make the VI-CarRealTime tools available within MATLAB the user has to add the paths of the VI-CarRealTime libraries to the MATLAB search path. To automate this operation, the installation procedure put in the User\\$UserName\$Documents\MATLAB directory the "**addpath_vicrt_20.m**" script. This function add to the MATLAB search path the VI-CarRealTime directories only for the current session, and is always available from the MATLAB command window.

In order to keep the VI-CarRealTime paths saved, after calling the "addpath_vicrt_20" function, the MATLAB search path list must be saved using the **Set Path** window from **File** menu.

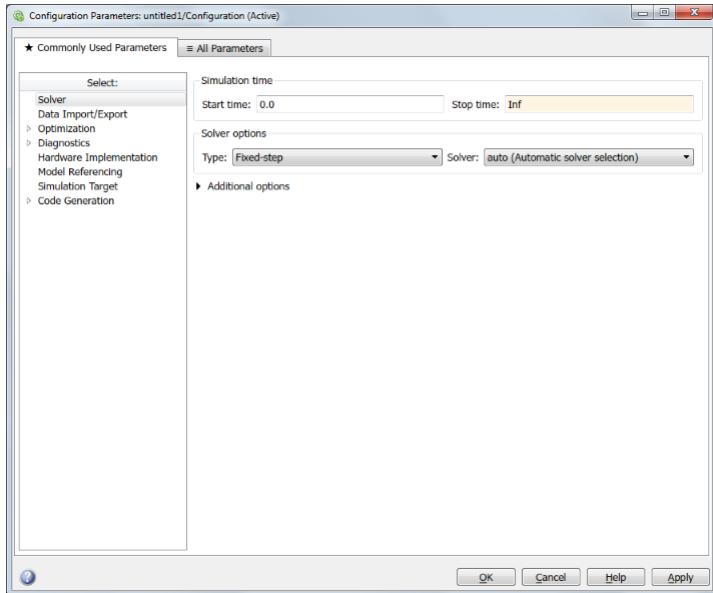


In addition, a MATLAB supported C/C++ compiler needs to be installed and properly setup.

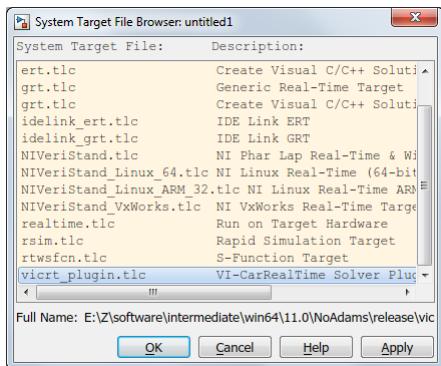
Simulink Model Configuration

To configure the Simulink model for the export of a VI-CarRealTime solver plugin, open the **Configuration Parameters** dialog.

- Go to Solver, make sure to set the Solver type to **Fixed-Step**, and set the Stop time to **Inf**:



- Go to **Code Generation** (or **Real Time Workshop**, dialog may differ depending on MATLAB version).
- Open the System target browser clicking on the **Browse...** button, select **vicrt_plugin.tlc** and press **OK**.

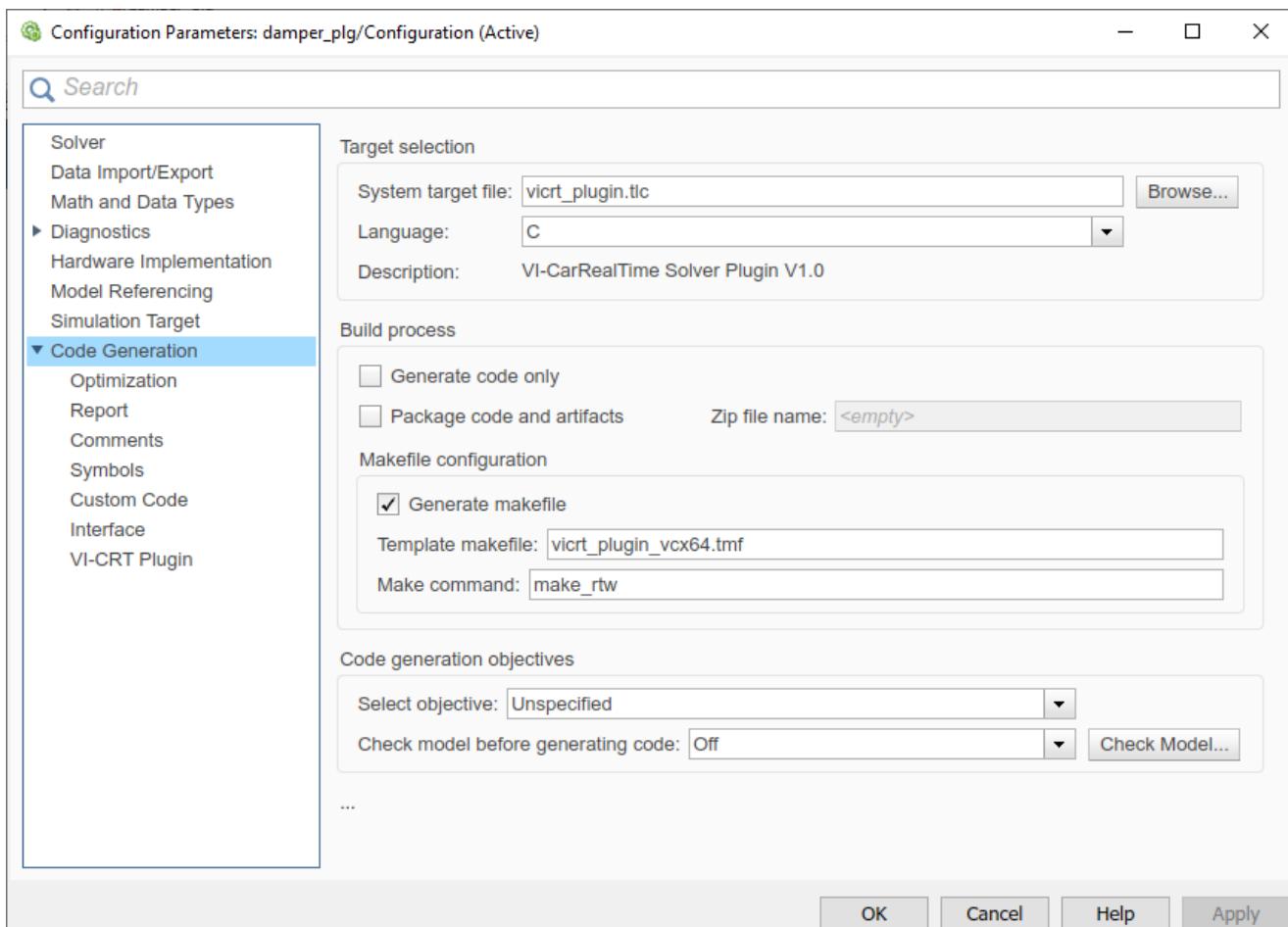


- Choose the proper **.tmf** file in the **Template makefile** field according to the target machine & compiler:
 - vicrt_plugin_vc64.tmf** (for Microsoft Visual C compiler Windows)
 - vicrt_plugin_mingw64.tmf** (for MinGW compiler Windows)
 - vicrt_plugin_ccur.tmf** (for Linux).
- If the build will be performed later on a second machine where MATLAB is not installed select **Generate code only** and **Package code and artifacts**.
- In **Code Generation/VI-CRT Plugin** set the VI-CRT TOPDIR with the VI-CarRealTime installation path (the default installation path is C:\Program Files\VI-grade\VI-CarRealTime 20).
If needed, set also PACKNGO MATLAB ROOT which represents the path on the second machine of the MATLAB folder extracted from the PackNGo zip file (generated when the option **Package code and artifacts** is checked). The PackNGo zip file is arranged with the following folders structure:
 - **working_folder** (same name of the current MATLAB folder)

- *output_folder* (model name plus "_crt_rtw" suffix)
- *R2018* (or similar name for other MATLAB releases)

The folders to extract and copy to the second machine are *output_folder* containing the source files together with the makefile to build them and *R2018* in the path specified by PACKNGO MATLAB ROOT.

- For the meaning of the other **Code Generation** parameters refer to Simulink documentation.
- Press **OK** in the **Configuration Parameters** dialog to apply changes and close the dialog.

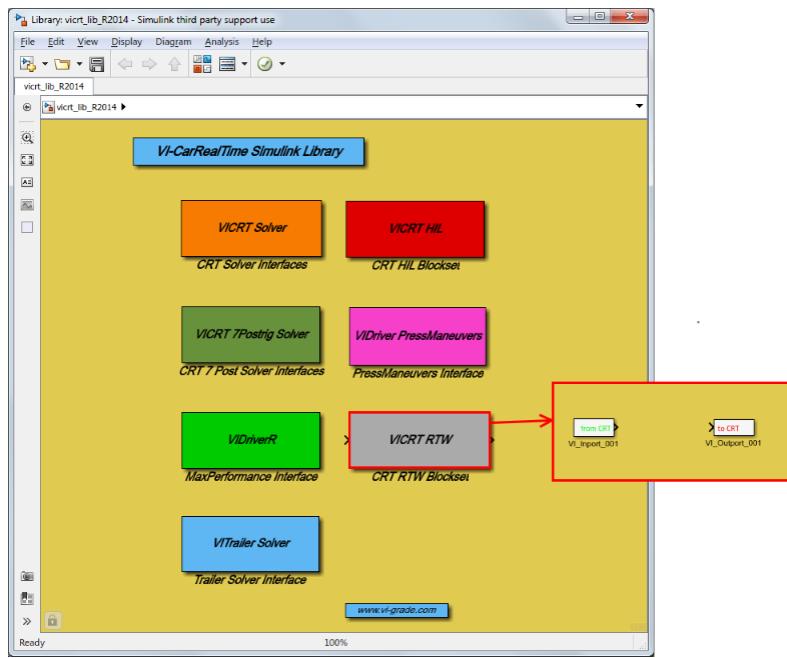


A parameter called "vicrt_inputfile" referencing a valid *_send_svm.xml vehicle model file must be defined in the MATLAB Workspace.

Simulink Model I/O (Obsolete)

The following process is now automated as a pre-build action and therefore this section is now obsolete but still functional.

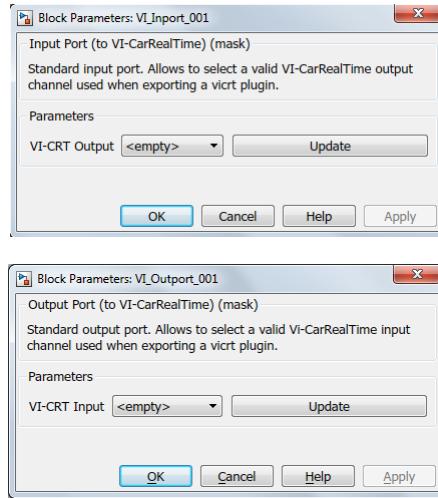
A Simulink model needs to be updated/modified in order to configure the connections to VI-CarRealTime solver inputs/outputs. The VI-CarRealTime Simulink library provides specific input/output blocks to easily setup the connection between the model exported from Simulink and the VI-CarRealTime solver:



The library provides two blocks:

- **toCRT**: this is an output of the Simulink model, and will be connected to a VI-CarRealTime Input.
- **fromCrt**: this is an external input to the Simulink, and will be connected to a VI-CarRealTime Output.

These blocks are standard Simulink Import and Outport, masked as follows:



If the `vicrt_inputfile` MATLAB variable is defined and contains the path to a valid `<event_name>_send_svm.xml` file, pressing the Update button on one of the blocks available in the model all the Input and Output lists are automatically updated to match the input/output list of the selected event file.

When a `VI-CRT Output` or `VI-CRT Input` is selected block name changes to match the VI-CarRealTime channel name. When exporting the model using Simulink Coder/RealtimeWorkshop the name of the block is used to identify the corresponding input/output channel of the model and it is connected accordingly.

It is also possible to register additional Input and Output ports specific of the Simulink model to make them available in VI-CarRealTime. This can be done by using standard Import and Outport from the Simulink library. When these elements are present in the model they are automatically converted to VI-CarRealTime model

VI-CarRealTime

Input/Output. For example if the Simulink model 'my_model' has 2 standard outports named 'my_custom_out_01' and 'my_custom_out_02', the results file will include a results set named 'my_model', with 'my_custom_out_01' and 'my_custom_out_02' components.

NOTE: Only I/O blocks on the top level of the model are allowed as External model interface.

NOTE - KNOWN ISSUE: If the Global setting for parameter optimization is set to **Inlined**, the I/O interface is not properly configured in the generated code. If optimized code is needed, a workaround can be implemented as follows:

- generated code
- modify the *_IO.h file replacing the empty I/O channels name with the proper VI-CarRealTime channels name (this should be easy because the correct name is available in the header file in the comment of each I/O connection)
- rebuild the shared library using the auto-generated script available together with the source files.

Simulink Model Parameters

When the VI-CarRealTime solver plug-in target is generated for a Simulink model, an [Auxiliary subsystem](#) is generated.

Such subsystem can be added to any VI-CarRealTime system model to embed the generated plugin. The parameters available in the Simulink Coder generated code are also exported to the [Auxiliary subsystem](#).

The VI-CarRealTime system target supports the export of **Tunable** Global Parameters, as well as the model blocks parameters if they are not **Inlined**. Global Parameters will appear in the [Auxiliary subsystem](#) with the user specified name, on the root level of the system parameter tree, while all the other parameters will be stored in a hierarchy that will reflect the same structure of the original Simulink model. **"Struct" type parameters are currently not supported.**

To learn more on how to make a Parameter available in the Simulink generated code refer to MATLAB documentation.

MaxPerformance Simulink Interface

To run a co-simulation using a MaxPerformance event, the following steps are needed:

1. In the VI-CarRealTime GUI, create a MaxPerformance event in "files only" mode.
2. Open MATLAB in the current working directory.
3. Generate the files mpx_initialize.m, mpx_runmaneuver.m, mpx_terminate.m in the working directory.
4. Run the event co-simulation using the vimaxperf_mtl command.

The vimaxperf_mtl command take as input the event file name (either the **event_name_mxp_send_svm.xml** or **event_name_vdr.mxp**) and optionally the result files output prefix and the partial history file name :

```
vimaxperf_mtl('event_name_mxp_send_svm.xml', ['event_name_work_history.xsh'])

vimaxperf_mtl({'event_name_mxp_send_svm.xml','output_files_prefix'}, ['event_name_work_history.xsh'])
```

or

```
vimaxperf_mtl('event_name_mxp.mxp', ['event_name_work_history.xsh'])

vimaxperf_mtl({event_name_mxp.mxp,'output_files_prefix'}, ['event_name_work_history.xsh'])
```

In order to use the MaxPerformance, the Simulink model must be instrumented using an interface block. The "MaxPerformance interface" block can be inserted anyway inside the model and is part of the "VI-CarRealTime library".



Run MaxPerformance from matlab

PressManeuvers Simulink Interface

To run a co-simulation using a PressManeuvers event, the following steps are needed:

1. In the VI-CarRealTime GUI, create a PressManeuvers event in "files only" mode.
2. Open MATLAB in the current working directory.
3. Generate the files isolc_initialize.m, isolc_runmaneuver.m, isolc_terminate.m in the working directory.
4. Run the event co-simulation using the vipressmaneuvers_crt command.

The vipressmaneuvers_crt command take as input the event file name (**event_name_pm_send_svm.xml**) and optionally the result files output prefix:

```
vipressmaneuvers_crt('event_name_pm_send_svm.xml')
vipressmaneuvers_crt({'event_name_pm_send_svm.xml','output_files_prefix'})
```

In order to use the PressManeuvers, the Simulink model must be instrumented using an interface block. The "PressManeuvers interface" block can be inserted anyway inside the model and is part of the "VI-CarRealTime library".



Run PressManeuvers from matlab

Matlab API Toolkit

This chapter covers the functionalities included in the Matlab API Toolbox. The final purpose of this library is to provide the user with the possibility to create and automate his own Design Of Experiment in order to find the best vehicle and suspension configuration.

This toolbox is composed by a list of Matlab API which allows the user to interact (modify, set and get data) with VI-CarRealTime files.

Using these Matlab API it is possible to manage the whole modeling and simulation processes for a VI-CarRealTime vehicle and for a VI-SuspensionGen suspension. These functions will interact directly with characteristics input files (*xml* and *sgs*) in order to modify their data directly in Matlab environment without using the specific product GUI.

Matlab library is available under the folder `$VI-CarRealTime$Installationdir\acarrt\matlab\opt_lib\lib`. For each matlab function there are two different files:

- **.p** files: contains the function working code in binary version;

- **.m** files: contains the help of compiled functions stored in the .p files.

A set of examples is stored under `$VI-CarRealTimeInstallationdir\acarrt\matlab\opt_lib\example` folder.

The following chapters describes all the API functions available:

- [VI-CarRealTime Utilities](#)
- [VI-SuspensionGen Utilities](#)

In order to fully understand how to use this set of API, a set of [tutorials](#) is provided.

VI-CarRealTime Utilities

A complete Matlab functions package can be used to interact with VI-CarRealTime models from Matlab environment. The package is divided into three main groups:

- [functions to read and modify an existing .xml file \(system or subsystem\)](#)
=> [modeling example](#)
- [functions to publish or update VI-CarRealTime databases](#)
=> [database example](#)
- [functions to run analysis on the modified model](#)
=> [analysis example](#)

The following utility functions have been developed in order to be suitable for optimization routines. In the following topics the functions will be described and some examples for the correct usage will be discussed as well.

The [API Examples](#) topic illustrates some examples about how to use the main functions of the API Toolkit.

Modeling Utilities

The functions described in this topic allow the management of VI-CarRealTime data variables collected in model and subsystem files. All the information collected in the VI-CarRealTime vehicle model (system.xml) are stored in a specific matlab structure (**systemStruct**) which will be shared between all other function utilities.

Fields that belong to the generic `systemStruct` variable are:

- **systemFile**
string variable to collect full path of new auto-generated system model file.
- **workingDir**
temporary directory where modified xml will be stored.
- **cfg**
cell array of cells with dimensions (n,2) containing the string pairs alias-path, required to solve database references.
- **body**
matlab structure variable with the following fields:
 - **file**
string to collect the full path name of body subsystem xml file.
 - **isUpdated**
logical flag to identify if subsystem file reference on system file structure has been already modified.

- **stringId**
unique string to identify the subsystem on system tree.

- **frontSuspension**

matlab structure variable with the following fields:

- **file**
string to collect the full path name of front suspension subsystem xml file.
- **isUpdated**
logical flag to identify if subsystem file reference on system file structure has been already modified
- **stringId**
unique string to identify the subsystem on system tree.
- **springs**
matlab structure variable with the following fields:
 - **file**
matlab cell (2,2) in which are collected the full path for spring property files [left left-aux; right right-aux].
 - **isUpdated**
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
- **dampers**
matlab structure variable with the following fields:
 - **file**
matlab cell (2,2) in which are collected the full path for damper property files [left left-aux; right right-aux].
 - **isUpdated**
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
- **bumpstops**
matlab structure variable with the following fields:
 - **file**
matlab cell (2,2) in which are collected the full path for bumpstop property files [left left-aux; right right-aux].
 - **isUpdated**
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
- **reboundstops**
matlab structure variable with the following fields:
 - **file**
matlab cell (2,2) in which are collected the full path for reboundstop property files [left left-aux; right right-aux].
 - **isUpdated**
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
- **arb**
matlab structure variable with the following fields:
 - **file**
matlab cell (1) in which is collected the full path for antiroll bar property file
 - **isUpdated**
matlab logical value to identify if property file reference on subsystem file has been already modified.

- **rearSuspension**

matlab structure variable with the following fields:

- **file**
string to collect the full path name of rear suspension subsystem xml file.

- *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
 - *stringId*
unique string to identify the subsystem on system tree.
 - *springs*
matlab structure variable with the following fields:
 - *file*
matlab cell (2,2) in which are collected the full path for spring property files [left left-aux; right right-aux].
 - *isUpdated*
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
 - *dampers*
matlab structure variable with the following fields:
 - *file*
matlab cell (2,2) in which are collected the full path for damper property files [left left-aux; right right-aux].
 - *isUpdated*
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
 - *bumpstops*
matlab structure variable with the following fields:
 - *file*
matlab cell (2,2) in which are collected the full path for bumpstop property files [left left-aux; right right-aux].
 - *isUpdated*
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
 - *reboundstops*
matlab structure variable with the following fields:
 - *file*
matlab cell (2,2) in which are collected the full path for reboundstop property files [left left-aux; right right-aux].
 - *isUpdated*
matlab logical matrix [2,2] to identify if property file reference on subsystem file has been already modified.
 - *arb*
matlab structure variable with the following fields:
 - *file*
matlab cell (1) in which the full path for antiroll bar property file is collected.
 - *isUpdated*
matlab logical value to identify if property file reference on subsystem file has been already modified.
- **steering**
matlab structure variable with the following fields:
- *file*
string to collect the full path name of steering subsystem xml file.
 - *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
 - *stringId*
unique string to identify the subsystem on system tree.
- **powertrain**
matlab structure variable with the following fields:
- *file*
string to collect the full path name of powertrain subsystem xml file.

- *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
- *stringId*
unique string to identify the subsystem on system tree.

- **auxiliary**

matlab structure variable with the following fields:

- *file*
string to collect the full path name of auxiliary subsystem xml file.
- *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
- *stringId*
unique string to identify the subsystem on system tree.

- **frontTire**

matlab structure variable with the following fields:

- *file*
string to collect the full path name of front tire subsystem xml file.
- *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
- *stringId*
unique string to identify the subsystem on system tree.

- **rearTire**

matlab structure variable with the following fields:

- *file*
string to collect the full path name of rear tire subsystem xml file.
- *isUpdated*
logical flag to identify if subsystem file reference on system file structure has been already modified
- *stringId*
unique string to identify the subsystem on system tree.

- **output**

matlab structure variable with the following fields:

- *output_map*
string to collect the full of VI-CarRealtime output map.
- *matFlag*
logical flag to enable results on .mat format
- *xmlFlag*
logical flag to enable results on .res format

Matlab variable **systemStruct** is directly generated starting from a VI-CarRealTime system file by using the following function:

```
systemStruct = generateSystemStruct( systemFile, cfg, workingDir )
```

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

systemFile:string variable to collect the full path name of vehicle system input file (.xml).

cfg:cell array of cells with dimensions (n,2) containing the string pairs alias-path.

workingDir:temporary directory where modified xml will be stored.

In order to resolve the xml tree, the user must specify:

VI-CarRealTime

- the tag-name string pair;
- the string attribute that will be replaced;
- the value that should be set.

Keys	Attribute
<CRITBodySetupData name="bodySetupData" active="true" userDefined="false">	
<CRITFuelConfiguration name="fuelConfiguration" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="fa	
<CRITDriverConfiguration name="driverConfiguration" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="fa	
<CRITBallast name="ballast1" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
<CRITBallast name="ballast2" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
<CRITBallast name="ballast3" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
<CRITBallast name="ballast4" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
<CRITBallast name="ballast5" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
<CRITBallast name="ballast6" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="(0 0	
</CRITBodySetupData>	
<CRITCrossWeightAdjustment name="cross_weight_adjustment" active="true" userDefined="false" crossWeightActiveFlag="false" crossWe	
<CRITAerodynamicForces name="aero_forces" active="false" userDefined="false" activeFlag="true" frontDownForceLocation="(0 0 0)" r	
</Subsystem>	
</afc>	

Such information must be included in Matlab cell variables as shown in the code example below:

```
▼ bodySetTriad
    %% set location/orientation
    keys=[];
    values=[];
    keys=cell(2,2);
    keys{1,1}='CRITBodySetupData';
    keys{1,2}='CRITBallast';

    keys{2,1}='bodySetupData';
    keys{2,2}='ballast3';

    attributes= cell(1,1);
    attributes{1}='location';

    values=cell(1,1);
    values{1} = [100, 0.0, 400];

    systemStruct = bodySetTriad(systemStruct,keys,attributes,values);
```

The following functions allow to access to subsystem files in order to modify values. For each **<subsystemName>** of the following list:

- auxiliary
- body
- brakes
- frontSuspension
- frontTire
- powertrain
- rearSuspension
- rearTire
- steering

the following methods are available (*):

```
➤ systemStruct = <subsystemName>SetBoolValue(systemStruct,keys,attributes,values)
```

function to set Boolean value to a specific field of the selected subsystem.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`attributes`:cell variable {1} containing the attribute that will be set.

`values`:cell variable {1} containing 'true' or 'false'.

```
> systemStruct = <subsystemName>SetPropertyFile(systemStruct, keys, attributes, values)
```

function to set property file to a specific field of the selected subsystem.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`attributes`:cell variable {1} containing the attribute that will be set.

`values`:cell variable {1} containing the new property file string (database search is supported).

```
> systemStruct = <subsystemName>SetSpline2D(systemStruct, keys, values)
```

function to set spline 2D to a specific field of the selected subsystem.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`values`:cell {1} variable containing a matlab structure with the following fields:

x – real array containing independent variable values.

y – real array containing spline values.

```
> systemStruct = <subsystemName>SetSpline3D(systemStruct, keys, values)
```

function to set spline 3D to a specific field of the selected subsystem.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`values`:cell {1} variable containing a matlab structure with the following fields:

x – real array containing first independent variable values (N elements).

y – real array containing second independent variable values (M elements).

f_xy – real matrix matrix containing surface data (N,M).

```
> systemStruct = <subsystemName>SetTriad (systemStruct, keys, attributes, values)
```

function to set location/ orientation to a specific field of the selected subsystem.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

VI-CarRealTime

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
attributes:cell variable {1} containing the attribute that will be set.
values:cell variable {1} containing a matlab real array of three elements.

```
> systemStruct = <subsystemName>SetValue (systemStruct,keys,attributes,values)
```

function to set location/ orientation to a specific field of the selected subsystem.

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.
keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
attributes:cell variable {1} containing the attribute that will be set.
values:cell variable {1} containing a value that will be set.

```
> value = <subsystemName>GetValue (systemStruct,keys,attribute)
```

function to get the value from a specific field of the selected subsystem.

Output:

value:matlab structure containing the value of the attribute.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.
keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
attribute:cell variable {1} containing the attribute whose value is to retrieve.

```
> splineData = <subsystemName>GetSpline2D(systemStruct,keys)
```

function to get a spline 2D from a specific field of the selected subsystem.

Output:

splineData:matlab struct with the following fields:
x - double array containing independent variable values
y - double array containing spline values

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.
keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

```
> splineData = <subsystemName>GetSpline3D(systemStruct,keys)
```

function to get a spline 3D from a specific field of the selected subsystem.

Output:

splineData:matlab struct with the following fields:
x - double array containing first independent variable values (N elements)
y - double array containing second independent variable values (M elements)
f_xy - double matrix containing surface data (N x M elements)

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle.
keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

Additional function utilities are included in MATLAB library in order to access to elastic elements (spring, damper,....) data values stored in external property files.

These methods are available for front and rear suspensions and only for <Element> that belongs to the following list:

- *MainBumpstop*
- *MainDamper*
- *MainReboundstop*
- *MainSpring*

Functions are (**):

```
> value = (front/rear)Suspension<Element>FileGetRealValue(systemStruct,keys,attributes,side,extraElement)
```

function to get real value from front subsystem elastic element property file.

Output:

value:matlab real value.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

attributes:cell variable {1} containing the attribute that will be retrieved.

side:matlab string used to identify suspension side that will be processed 'left - right'.

extraElement:matlab variable (0/1) to identify if an extra element will be processed.

```
> splineData =  
frontSuspension<Element>FileGetSpline2D(systemStruct,keys,side,extraElement)
```

function to get spline 2D from front subsystem elastic element property file

Output:

splineData:matlab structure with the following fields:

x – real array containing independent variable values.

y – real array containing spline values.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

side:matlab string used to identify suspension side that will be processed 'left - right'.

extraElement:matlab variable (0/1) to identify if an extra element will be processed.

```
> systemStruct = frontSuspension<Element>FileSetSpline2D(systemStruct,keys,values,side,extraElement)
```

function to set spline 2D to a front subsystem elastic element property file.

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

VI-CarRealTime

values:cell {1} variable containing a matlab structure with the following fields:

x – real array containing independent variable values.

y – real array containing spline values.

side:matlab string used to identify suspension side that will be processed 'left - right'.

extraElementmatlab variable (0/1) to identify if an extra element will be processed.

```
> systemStruct = frontSuspension<Element>FileSetValue(systemStruct,keys,attributes,values,side,extraElement)
```

function to set real value to a front subsystem elastic element property file.

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

attributes:cell {1} variable containing the attribute that will be set.

values:cell {1} variable containing a the value that will be set.

side:matlab string used to identify suspension side that will be processed 'left - right'.

extraElementmatlab variable (0/1) to identify if an extra element will be processed.

The same concept applies for *single* elastic elements (spring, damper,...) data values stored in external property files.

These methods are available for front and rear suspensions and only for <**Element**> that belongs to the following list:

- *SingleSpring*
- *SingleSpringBumpstop*
- *SingleSpringReboundStop*
- *ThirdSpring*
- *SingleDamper*
- *SingleDamperBumpstop*
- *SingleDamperReboundstop*
- *ThirdDamper*
- *RollDamper*
- *RollDamperBumpstop*
- *RollDamperReboundstop*
- *JDamper*
- *JDamperBumpstop*
- *JDamperReboundstop*

Functions are:

```
> value = (front/rear)Suspension<Element>FileGetRealValue(systemStruct,keys,attributes)
```

function to get real value from front subsystem elastic element property file.

Output:

value:matlab real value.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..
`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
`attributes`:cell variable {1} containing the attribute that will be retrieved.

```
> splineData = frontSuspension<Element>FileGetSpline2D(systemStruct, keys)
```

function to get spline 2D from front subsystem elastic element property file

Output:

`splineData`:matlab structure with the following fields:
`x` – real array containing independent variable values.
`y` – real array containing spline values.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..
`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

```
> systemStruct = frontSuspension<Element>FileSetSpline2D(systemStruct, keys, values)
```

function to set spline 2D to a front subsystem elastic element property file.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`values`:cell {1} variable containing a matlab structure with the following fields:

`x` – real array containing independent variable values.

`y` – real array containing spline values.

```
> systemStruct = frontSuspension<Element>FileSetValue(systemStruct, keys, attributes, values)
```

function to set real value to a front subsystem elastic element property file.

Output:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

`systemStruct`:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

`keys`:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

`attributes`:cell {1} variable containing the attribute that will be set.

`values`:cell {1} variable containing a the value that will be set.

Here the functions to modify antiroll bar property file:

```
> value = (front/rear)SuspensionArbFileGetRealValue(systemStruct, keys, attributes)
```

function to get real value from front subsystem main antiroll bar property file.

Output:

value:matlab real variable.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

attributes:cell {1} variable containing the attribute that will be retrieved.

```
> splineData = (front/rear)SuspensionArbFileGetSpline2D(systemStruct,keys)
```

function to get spline 2D from front subsystem main antiroll bar property file

Output:

splineData:matlab structure with the following fields:

x – real array containing independent variable values.

y – real array containing spline values.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

```
> value = (front/rear)SuspensionArbFileGetStringValue(systemStruct,keys,attributes)
```

function to get string value from front subsystem main antiroll bar property file.

Output:

value:matlab string value.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

attributes:cell {1} variable containing the attribute that will be retrieved.

```
> systemStruct = (front/rear)SuspensionArbFileSetSpline2D(systemStruct,keys,values)
```

function to set spline 2D to a front subsystem main antiroll bar property file.

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.

values:cell {1} variable containing a matlab structure with the following fields:

x – real array containing independent variable values.

y – real array containing spline values.

```
> systemStruct = (front/rear)SuspensionArbFileSetValue(systemStruct,keys,attributes,values)
```

function to set real value to a front subsystem main antiroll bar property file.

Output:

systemStruct:matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

Input:

`systemStruct`: matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..
`keys`: cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
`attributes`: cell {1} variable containing the attribute that will be set.
`values`: cell {1} variable containing a the value that will be set.

```
> value = frontSuspensionArbFileGetStringValue(systemStruct, keys, attributes, side, extraElement)
```

function to get string value from front subsystem elastic element property file.

Output:

`value`: matlab string value.

Input:

`systemStruct`: matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model..
`keys`: cell matrix {2,n} containing the list of (tag; name) strings required to resolve xml tree.
`attributes`: cell variable {1} containing the attribute that will be retrieved.
`side`: matlab string used to identify suspension side that will be processed 'left - right'.
`extraElement`: matlab variable (0/1) to identify if an extra element will be processed.

A function utility is included in MATLAB library in order to automatically update kinematic steering axis data properties stored in steering subsystem:

```
systemStruct = steeringUpdateSteeringAxis(systemStruct).
```

(*) Example: for the `<subsystemName>GetValue` function, the following flavors are available:

- `auxiliary GetValue`
- `body GetValue`
- `brakes GetValue`
- `frontSuspension GetValue`
- `frontTire GetValue`
- `powertrain GetValue`
- `rearSuspension GetValue`
- `rearTire GetValue`
- `steering GetValue`

(**) Example: for the `(front/rear)Suspension<Element>FileGetRealValue` function, the following flavors are available:

- `frontSuspensionMainSpring FileGetRealValue`
- `rearSuspensionMainSpring FileGetRealValue`
- `frontSuspensionMainDamper FileGetRealValue`
- `rearSuspensionMainDamper FileGetRealValue`
- `frontSuspensionMainBumpstop FileGetRealValue`
- `rearSuspensionMainBumpstop FileGetRealValue`
- `frontSuspensionMainReboundstop FileGetRealValue`
- `rearSuspensionMainReboundstop FileGetRealValue`

VI-CarRealTime

This is an example about how to manage VI-CarRealTime modeling functionalities using Matlab Optimization Tool.

The source m file is stored in `$VI-CarRealTimeInstallationdir\acarrt\matlab\opt_lib\example\testModeling` folder, together with `vicrt_cdb.cfg` file.

The main inputs for this example are:

- VI-CarRealTime system.xml file: '`mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml`'.
- `vicrt_cdb.cfg` database configuration file: this file collects the information required to solve database path.

This is an example of `vicrt_cdb.cfg` file:

```
!-----!
! *****      VI-CarRealTime Database Configuration File      *****
!-----!
!          Database name      Path of Database
!-----!
DATABASE  carrealtime_shared    C:/PROGRA~1/VI-grade/VI-CAR~1/17/acarrt/carrealtime_shar
```

As a first step, after selecting a base system.xml file (`systemFile`) and a database configuration file (`cfgFile`), the matlab structure `systemStruct` is created from input file and locally saved in the temporary directory named `modelDirFiles` through the [generateSystemStruct](#) function.

Then powertrain map will be modified by using the function [powertrainSetSpline3D](#).

Finally using [rearSuspensionMainDamperFileGetSpline2D](#) and [rearSuspensionMainDamperFilesetSpline2D](#), original damper characteristic file will be scaled using a factor 2.

All new modified files (system, rear suspension, powertrain and damper property file) will be saved in `modelDirFiles` folder.

▼ testModeling.m

```
cfgFile='vicrt_cdb.cfg';
systemFile = 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml';

%create cfg array variable
cfg = createCfg(cfgFile);

%% resolve system reference and generate a temporary system file
modelDirFiles='crtWork';
systemStruct = generateSystemStruct(systemFile,cfg,modelDirFiles);

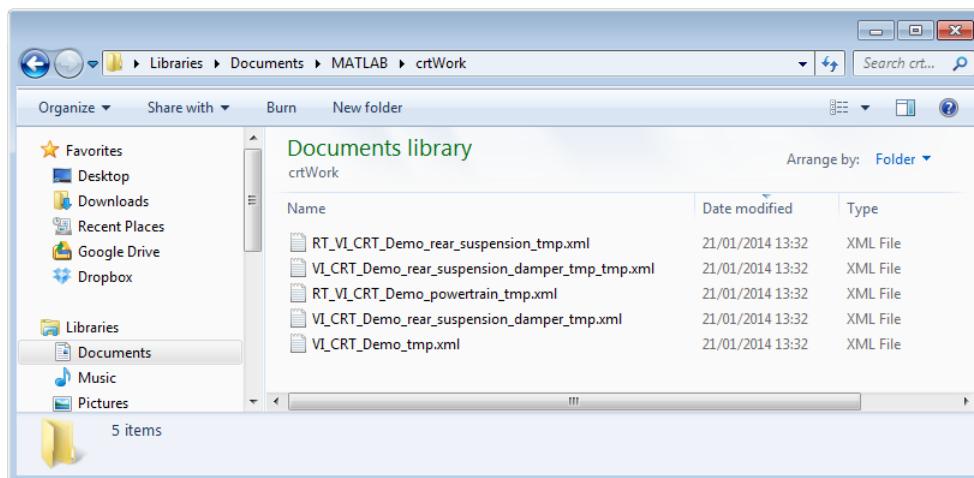
%% Powertrain map
%% modify spline 3d kinematic
keys=[];
values=[];
keys=cell(2,2);
keys{1,1}='CRTPowertrain';
keys{1,2}='Spline3d';

keys{2,1}='powertrain';
keys{2,2}='powertrainMap';

values=cell(1);

splineData.x = [0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000 5500 6000 6250 6500 6
splineData.y = [0 1];
splineData.f_xy =[           0         0
                  -39262   310000
                  -39262   310000
                  -39262   310000
                  -39262   310000
```

The following snapshot shows the files stored in the output folder.



Database Utilities

Functions collected here allow the resolution and the processing of VI-CarRealTime databases:

➤ `aliasPath = createCfg(cfgFile)`

function to generate a matlab cell variable containing info required to database resolution.

Output:

`aliasPath`:cell array of cells with dimensions (n,2) containing the string pairs alias-path .

Input:

`cfgFile`:fullname of `vicrt_cdb.cfg` file. When an empty string is passed, a default cfg file is generated in the working directory and its content is returned

➤ `createCfgFilefromCfgStruct(cfgCellArray,outFilename)`

function to create cfg file from cfg cell array.

Output:

- :-

Input:

`cfgCellArray`:cell array of cells with dimensions (n,2) containing the string pairs alias-path.

`outFilename`:output file name.

➤ `dbExists = checkDbAlias(cfg,alias)`

function to check if a database is registered on cfg cell array.

Output:

`dbExists`:matlab int variable (0- database is not registered / 1 - database already exists).

Input:

`cfg`:cell array of cells with dimensions (n,2) containing the string pairs alias-path.

`alias`:matlab string with the database alias that will be checked.

➤ `createDatabase(alias,path)`

function to create a new database

Output:

- :-

Input:

`alias`:matlab string with the database alias that will be added.

`path`:matlab string with the database path that will be added.

➤ `file = encodeDb(fullPath, cfg)`

function to generate a valid full path for a file using database reference

Output:

`file`:string variable containing a database file reference using alias (mdids://)

Input:

`fullpath`:string variable containing the full path of a file referenced using database reference.

`cfg`:cell array of cells with dimensions (n,2) containing the string pair alias-path.

➤ `cfg = addDbToCfgStruct(cfg,alias,path)`

function to add a new database to an existing matlab cfg cell array variable

Output:

`cfg`: cell array of cells with dimensions (n,2) containing the string pairs alias-path

Input:

`cfg`: cell array of cells with dimensions (n,2) containing the string pair alias-path.

`alias`: matlab string with the database alias that will be added.

`path`: matlab string with the database path that will be added.

```
> [systemFile cfg] = publishSystemFileToDb(systemFileBase, cfg, targetSystemName ,targetDbAlias, targ
```

function to publish an existing system.xml into a database (if database does not exist, it will be automatically generated)

Output:

`systemFile`: matlab string with full path of published system .xml file.

`cfg`: cell array of cells with dimensions (n,2) containing the string pairs alias-path. With respect to input value, it will include the row relative to the new database.

Input:

`systemFileBase`: matlab string with full path of system .xml file that will be published.

`cfg`: cell array of cells with dimensions (n,2) containing the string pairs alias-path.

`targetSystemName`: matlab string name of new system xml file (if empty string the new system file will have the same name as the base file).

`targetDbAlias`: matlab string with the database alias that will be added.

`targetDbPath`: matlab string with the database path that will be added.

```
> [systemFile cfg] = publishSystemStructToDb (systemFileBase, targetSystemName ,targetDbAlias, targ
```

function to publish an existing system.xml into a database (if database does not exist, it will be automatically generated)

Output:

`systemFile`: matlab string with full path of published system .xml file.

`cfg`: cell array of cells with dimensions (n,2) containing the string pairs alias-path. It will include the row relative to the new database.

Input:

`systemStruct`: matlab structure containing the list of available subsystems in VI-CarRealTime vehicle. This structure represents the model that will be published.

`targetSystemName`: matlab string name of new system xml file (if empty string the new system file will have the same name as the base file).

`targetDbAlias`: matlab string with the database alias that will be added.

`targetDbPath`: matlab string with the database path that will be added.

```
> fullPath= resolveDb (file, cfg)
```

function to generate a valid full path for a file collected using database reference.

Output:

`fullPath`: matlab string with the file full path.

Input:

`file`: matlab string with a database file reference using alias (mdids://").

`cfg`: cell array of cells with dimensions (n,2) containing the string pairs alias-path. It will include the row relative to the new database.

```
> subsystemFullName = resolveSubsystemFromSystem (systemFullName,subsystemString ,cfg)
```

VI-CarRealTime

function to generate a valid full path for a subsystem file collected on a system file.

Output:

`subsystemFullNa`matlab string with the file full path.

Input:

`systemFullNam`string variable containing the full path of vehicle model system file (.xml).

`subsystemStr`string variable containing the unique string representing the role (auxiliary, body, ...) of subsystem inside system file tree.

`cfg`:cell array of cells with dimensions (n,2) containing the string pairs alias-path. It will include the row relative to the new database.

This is an example about how to use Matlab Optimization Tool for managing VI-CarRealTime database functionalities.

The source m file is stored in `$VI-CarRealTimeInstallationdir\acarrt\matlab\opt_lib\example\testDb` folder.

The main inputs for this example are:

- VI-CarRealTime system.xml file: 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml'

As a first step, after selecting a base system.xml file (`systemFileOri`), a fresh copy of the whole vehicle model is created into a new local database named `viCrtDemo` . The source model is loaded from the default VI-CarRealTime shared database.

Next steps are the creation of a `systemStruct` variable and the modification of the chassis center of gravity. Finally the matlab structure variable will be used to update VI-CarRealTime model in the `viCrtDemo` database.

▼ testDb.m

```
%%%%%%
% replacing multiple entities on crt model and run a collection of events %
%%%%%
systemFileOri = 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml';

%create cfg array variable
cfg = createCfg('');

dbAlias='viCrtDemo';
dbPath='';

%% publish a system file in a new database
[systemFile cfg] =publishSystemFileToDb(systemFileOri, cfg ,'',dbAlias, dbPath);

%% load new system file
systemFile='mdids://viCrtDemo/systems.tbl/VI_CRT_Demo.xml';

%% resolve system reference and generate a temporary system file
modelDirFiles='crtWork';
systemStruct = generateSystemStruct(systemFile,cfg,modelDirFiles);

%% Body Subsystem Modifications %%
keys = cell(2,1);
keys{1,1}='CRTSprungMass';
keys{2,1}='body';

attributes= cell(3,1);
attributes{1}='cgHeight';
attributes{2}='cgX';
attributes{3}='cgY';

values=cell(3,1);
values{1}=600;
```

```

values{2}=400;
values{3}=0.5;

systemStruct = bodySetValue(systemStruct,keys,attributes,values);

%% update model in viCrtDemo Database
[systemFile cfg] =publishSystemStructToDb(systemStruct,systemFile,dbAlias, dbPath );

```

Analysis Utilities

The main function for analysis purposes is given by **runViCrt** which runs a VI-CarRealTime batch session using as input a `fingerprint.xml` file (containing the event specifications) and the vehicle model (collected as `systemStruct` matlab variable).

```
> [successFlag                               outputNameList]
=runViCrt(baseFingerprint,systemStruct,outSuffix,workingDir,runWindowON)
```

Output:

`successFlag`: flag to monitor solver status (ok=1).
`outputNameList` cell {n} containing the list of outputs file that will be available after VI-CarRealTime simulation.

Input:

`baseFingerprint`: `fingerprint.xml` template that will be used to collect the events that will be simulated using vehicle system struct

`systemStruct`: matlab structure containing the list of available subsystems in VI-CarRealTime vehicle model.

`outSuffix`: base name used for analysis results.

`workingDir`: name of temporary folder in which simulations files will be saved.

`runWindowON`: flag to enable python window console (enabled=1).

The output files of the simulation will be stored in the `workingDir`. For a full description of all the solver output files look [here](#).

Furthermore a file called **vicrt.log** will be written in the current working directory: such file is the VI-CarRealTime simulation log file and it can be useful for debugging process in case of issues during the batch simulation.

This is an example about how to run VI-CarRealTime solver in batch mode using Matlab Optimization Tool.

The source m file is stored in `$VI-CarRealTimeInstallationdir\acarrt\matlab\opt_lib\example\testRun` folder, together with `vicrt_cdb.cfg` and `sin.xml` file.

The main inputs for this example are:

- VI-CarRealTime system.xml file: '`mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml`'
- A `fingerprint.xml` file named `sin.xml`

This file contains all the events specification (driver file, road file, integration time step,...)

▼ sin.xml

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<afc xmlns="http://www.mscofware.com/:vfc" xmlns:afc="http://www.mscofware.co
<afc:Bibliography>
    <File schema="afc" version="2.0.0.0" URI="file:///D:/K/Customer/HMC/Optimizati
    <Revision version="2" />
    <Corporation author="MSC.Software" URI="http://www.mscofware.com/" />
    <Author user="Michele" name="-unknown name-" />
    <Environment hostName="DUKKO" operatingSystem="MS Windows NT 6.1" />
    <Application name="ADAMS/Chassis" version="2005" />
</afc:Bibliography>
<afc:Units>

```

```

<afc:UnitSetting name="angle" current="degree" />
<afc:UnitSetting name="current" current="ampere" />
<afc:UnitSetting name="force" current="newton" />
<afc:UnitSetting name="length" current="mm" />
<afc:UnitSetting name="luminosity" current="candela" />
<afc:UnitSetting name="mass" current="kilogram" />
<afc:UnitSetting name="quantity" current="mole" />
<afc:UnitSetting name="solidAngle" current="steradian" />
<afc:UnitSetting name="temperature" current="kelvin" />
<afc:UnitSetting name="time" current="second" />
<afc:UnitSetting name="vehicle_velocity" current="kph" />
<afc:UnitSetting name="course_length" current="meter" />
</afc:Units>
<chassis:SVMFingerprint name="sin" active="true" userDefined="false">
  <afc:FileDriven name="sin_steer" active="true" userDefined="false">
    systemFile="mdids://bus/systems.tbl/hmc_bus_dual.xml" overlay=
    integrationTimeStep="0.001" outputTimeStep="0.01" integrator=
    roadFile="mdids://carrealtime_shared/roads.tbl/flat.rdf" fLro=
    fRroadFile="mdids://carrealtime_shared/roads.tbl/flat.rdf" rL=
    individualRoad="false" useRoadGraphicsFile="false" useEventLa=
    enableIndividualRoad="true" tireLimits="false" tLFZMin="1e-00" tLLongitudinalSlipMax="0.3" tLLongitudinalSlipStep="0.5" tLLa=
    showSolverSettingsWidgets="true" showTireLimitsWidgets="true" plotConfigFile="mdids://carrealtime_shared/plot_configs.tbl/v=
    smartDriverFile="mdids://carrealtime_shared/driver_controls.tl
  </chassis:SVMFingerprint>
</afc>
```

As a first step the Matlab structure `systemStruct` is created from the existing `system.xml` file, using `cfgFile` for database resolution.

Next steps are the enabling of res output file format and the definition of a name for the output folder.

Finally using the event file collected in the variable `baseFingerprint`, VI-CarRealTime solver will perform the dynamic simulation. When simulation will be terminated, the results file in `mat` format will be imported and the dynamic behavior of vehicle Yaw Rate will be shown by Matlab plot tool.

▼ testRun.m

```

%%%%%%%%%%%%%
% replacing multiple entities on crt model and run a collection of events %
% -sine steer %
%%%%%%%%%%%%%

systemFile = 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml';
baseFingerprint='sin.xml';

%% resolve system reference and generate a temporary system file
modelDirFiles='crtWork';
systemStruct = generateSystemStruct(systemFile,createCfg(''),modelDirFiles);

%%%%%%%%%%%%%
%% update fingerprint and run %%
%%%%%%%%%%%%%
systemStruct.output.xmlFlag=1; %% set it to 1 in order to enable output file in res form
workingDir = 'crtWork';
[successFlag outputNameList]=runViCrt(baseFingerprint,systemStruct,'work',workingDir);

Legend=cell(1,1);
Legend{1}='Yaw Rate Sin';

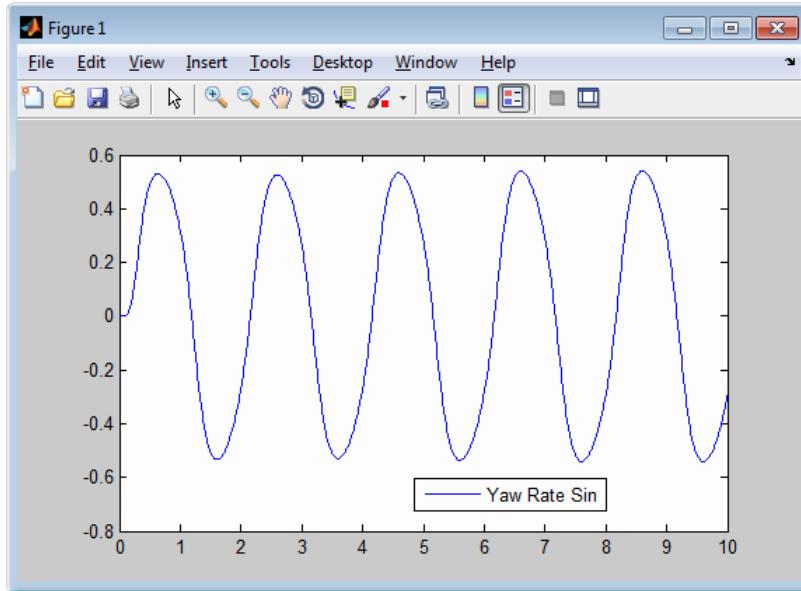
% load and plot yaw_rate
for i=1:length(outputNameList)
```

```

output=sprintf('%s.mat',outputNameList{i});
out=load(output);
figure;
plot(out.time,out.chassis_velocities_yaw);
legend(Legend{i});
clear out;
end

```

The following plot shows the computed yaw rate signal:



API Examples

The aim of this topic is to provide the user with a couple of flash examples which show some basic functionalities of Matlab API tool.

The Matlab code for each example will be reported.

Note: for reproducing each example copy and paste the related code together with the following lines in the working directory used in Matlab to run the script. The shared examples generates a default vicrt_cdb.cfg file on the fly.

```

addpath_vicrt_20;

systemFile = 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml';

modelDirFiles = 'tmpFolder';
systemStruct = generateSystemStruct(systemFile,createCfg(''),modelDirFiles);

```

Example 1: Retrieving Vehicle Body Cg Height

<pre>key(1,1)</pre>	<pre>key(2,1)</pre>	<pre>attr(1) cgHeight</pre>
<pre><Subsystem name="RT_VI_CRT_Demo_body" active="true" userDefined="false"> <CRTSprungMass name='body' active="true" userDefined="false" cgHeight="413.404" <SensorPoint name="sensor_point" active="true" userDefined="false" location="(-1 <cgReferencePoint name="cg_reference_point" active="true" userDefined="false" lo <Spline name="front_ride_height_map" id="0" active="true" userDefined="false" in</pre>		

VI-CarRealTime

```
% (tag ; name)
key(1,1) = {'CRTSprungMass'};
key(2,1) = {'body'};
% attribute to retrieve: mass
attr(1) = {'cgHeight'};

% get mass of body subsystem
cgHeight = bodyGetValue(systemStruct,key,attr)
```

Example 2: Set Vehicle Body Mass

```
% (tag ; name)
key(1,1) = {'CRTSprungMass'};
key(2,1) = {'body'};
% attribute to retrieve: mass
attr(1) = {'mass'};
mass_val = {500};

% get mass of body subsystem
bodySub = bodySetValue(systemStruct,key,attr,mass_val)
```

Example 3: Retrieving Left Kingpin Location Arbitrary Point on Steering subsystem

```
<Subsystem
  .active="true"
  .userDefined="false"
>                               key(1,1)
<CRTSteeringSystem>           key(2,1)
  .name="steering_gear"
  .active="true"
  .userDefined="false"
  .nominalSteeringGearRatio="1.0"
  .kingpinLeftFront="2.0527334317"    kplf
  .kingpinRightFront="2.0527334317"    kpfr
  attr(1)[arbitraryLeftPoint="(5.0 -90.0 -145.0)"]
  arbitraryRightPoint="(5.0 90.0 -145.0)"
  .casterAngleLeftFront="2.0527334838"
  .casterAngleRightFront="2.0527334838"
```

```
% (tag ; name)
key(1,1) = {'CRTSteeringSystem'};
key(2,1) = {'steering_gear'};
attr(1) = {'arbitraryLeftPoint'};

% get left kingpin point location
kplf = steeringGetValue(systemStruct,key,attr)
```

Example 4: Retrieving Vehicle Body Ballast 1 Mass

```
key(1,1)          key(2,1)          attr(1)
<CRIBodySetupData name="bodySetupData" active="true" userDefined="false">
  <CRITruckConfiguration name="truckConfiguration" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0">
    <CRIDriverConfiguration name="driverConfiguration" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0">
      <CRTBallast name="ballast1" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
      <CRTBallast name="ballast2" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
      <CRTBallast name="ballast3" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
      <CRTBallast name="ballast4" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
      <CRTBallast name="ballast5" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
      <CRTBallast name="ballast6" active="true" userDefined="false" setupActiveFlag="false" runningActiveFlag="false" location="0 0 0" mass="0"/>
```

key(1,2) key(2,2) mass1

```
% (tag ; name) first level
```

```

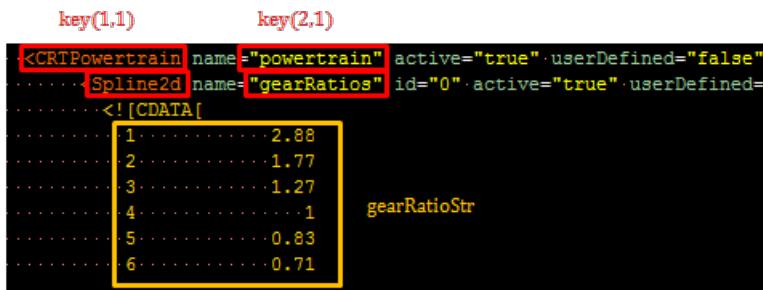
keys(1,1) = {'CRTBodySetupData'};
keys(2,1) = {'bodySetupData'};
% (tag ; name) second level
keys(1,2) = {'CRTBallast'};
keys(2,2) = {'ballast1'};
attr(1) = {'mass'};

% get mass of ballast 1
mass1 = bodyGetValue(systemStruct,keys,attr)

```

Example 5: Retrieving Powertrain Gear Ratio 2D Spline

key(1,1) key(2,1)



```

<CRTPowertrain name="powertrain" active="true" userDefined="false"
    ><Splined2d name="gearRatios" id="0" active="true" userDefined="false">
        <![CDATA[
            1.....2.88
            2.....1.77
            3.....1.27
            4.....1
            5.....0.83
            6.....0.71
        ]>
    </Splined2d>
</CRTPowertrain>

```

key(1,2) key(2,2)

```

% (tag ; name) first level
keys(1,1) = {'CRTPowertrain'};
keys(2,1) = {'powertrain'};
% (tag ; name) second level
keys(1,2) = {'Splined2d'};
keys(2,2) = {'gearRatios'};

gearRatioStr = powertrainGetSpline2D(systemStruct,keys)

```

Example 6: Set Powertrain Gear Ratio

```

% (tag ; name) first level
keys(1,1) = {'CRTPowertrain'};
keys(2,1) = {'powertrain'};
% (tag ; name) second level
keys(1,2) = {'Splined2d'};
keys(2,2) = {'gearRatios'};
val = {gearRatioStr}; % gearRatioStr is the structure defined in the Example 5

gearRatios = powertrainSetSpline2D(systemStruct,keys,val)

```

VI-SuspensionGen Utilities

A complete Matlab functions package can be used to interact with VI-SuspensionGen models from Matlab environment.

The package is divided into three groups:

- [functions to read and modify an existing .sgs file.](#)
- [functions to manipulate database.](#)
- [functions to run analysis on the modified model.](#)

The utilities have been developed in order to be suitable for optimization routines. In the following topics the functions will be described.

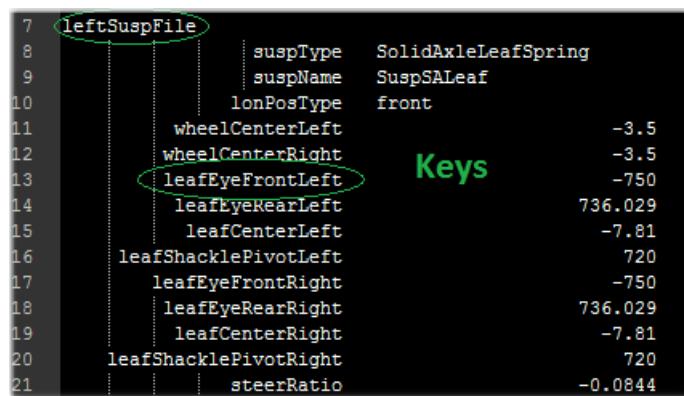
Modeling Utilities

Functions collected here allow the management of VI-SuspensionGen data variables collected in suspension .sgs input file. All the information collected in the suspension input file (suspension.sgs) are stored in a specific matlab structure which will be shared between all other function utilities.

Fields that belong to the generic suspensionGenStruct variable are:

- **s**
string variable collecting the whole content of the .sgs file.
- **role**
string variable to collect the suspension role (front/rear).
- **output**
matlab struct variable with the following fields:
 - **suspensionFile**
string to collect the full name of generated suspension xml file.
 - **steeringFile**
string to collect the full name of generated steering xml file.
 - **mat**
string to collect the full name of output results on .mat format.
- **originalFileName**
basic .sgs file name.
- **workingDir**
temporary directory where modified.
- **cfg**
cell array of cells with dimensions (n,2) containing the string pair alias-path.

The functions below use a keys variable which is a cell array containing the path of the parameter to be found in the suspension file structure.



7	leftSuspFile		
8		suspType	SolidAxeLeafSpring
9		suspName	SuspSAleaf
10		lonPosType	front
11		wheelCenterLeft	-3.5
12		wheelCenterRight	-3.5
13	leafEyeFrontLeft		-750
14		leafEyeRearLeft	736.029
15		leafCenterLeft	-7.81
16		leafShacklePivotLeft	720
17		leafEyeFrontRight	-750
18		leafEyeRearRight	736.029
19		leafCenterRight	-7.81
20		leafShacklePivotRight	720
21		steerRatio	-0.0844

```
▼ suspensionGenGetValue
keys = cell(2,1);
keys{1} = 'leftSuspFile';
keys{2} = 'leafEyeFrontLeft';
leftFrontEyeLoc = suspensionGenGetValue(suspensionGenStruct,keys);
```

Functions are:

```
> suspensionGenStruct = generateSuspensionGenStruct(suspensionGenFile, cfg, workingDir)
```

function to generate a Matlab struct variable *suspensionGenStruct*' to collect data about VI-SuspensionGen input file.

Output:

suspensionGen:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenFile:string variable to collect the full name of .sgs input file.

cfg:cell array of cells with dimensions (n,2) containing the string pair alias-path.

workingDir:temporary directory where modified xml will be stored.

```
> value = suspensionGenGetString(suspensionGenStruct, keys)
```

function to obtain a string from suspension stored as sgs struct. It is useful to get information like suspension type, suspension name and front/rear attributes.

Output:

value:matlab string variable containing the researched attribute

Input:

suspensionGen:matlab structure containing data storage for suspension model input file (.sgs).

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve .sgs tree.

```
> value = suspensionGenGetValue(suspensionGenStruct, keys)
```

function to obtain a variable from suspension stored as sgs struct. Use it for accessing to scalar parameters such as steerRatio, toeAngleLeft, camberAngleLeft and so on. It can return also a vector of size 3 when applied to get a point location.

Output:

value:matlab variable containing the researched attribute

Input:

suspensionGen:matlab structure containing data storage for suspension model input file (.sgs).

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve .sgs tree.

```
> suspensionGenStruct = suspensionGenSetLocation (suspensionGenStruct, keys, value)
```

function to set a point in a suspension stored as sgs struct. It is needed to set the location of the points in the suspension: wheelCenterLeft, tierodOuterLeft, kingpinPointLeft and so on.

Output:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve .sgs tree.

values:matlab cell {1} containing a matlab array of 3 elements

```
> suspensionGenStruct = suspensionGenSetRole (suspensionGenStruct, side)
```

function to set a specific role in a suspension stored as sgs struct. The string used as argument can be 'front' or 'rear'.

Output:

VI-CarRealTime

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

 side:matlab string variable containing the role (front/rear).

 values:matlab cell {1} containing a matlab array of 3 elements

```
> suspensionGenStruct = suspensionGenSetValue (suspensionGenStruct, keys, value)
```

function to set a variable in a suspension stored as sgs struct. There are several parameters in the .sgs file that are represented by scalar values: steerRatio, toeAngleLeft, camberAngleLeft as examples.

Output:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

 keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve .sgs tree.

 value:matlab cell {1} containing the value that will be set.

```
> leafForceData = suspensionGetLeafVerticalStiffness (suspensionGenStruct, keys, side)
```

function to get left/right leaf spring force curves from a suspension stored as sgs struct. A matrix [n,2] containing x-values (jounce) and y-values (force) as first and second column for leaf curve is returned.

Output:

leafForceData:matrix [n,2] containing x-values (jounce) and y-values (force) as first and second column for leaf curve.

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

 keys:cell matrix {2,n} containing the list of (tag; name) strings required to resolve .sgs tree.

 side:matlab integer variable 0:left, 1:right.

```
> suspensionGenStruct = suspensionSetDatabase (suspensionGenStruct, alias)
```

function to set output database alias. It changes the parameter which is named defaultDatabase. If a registered database is specified then the output files will be save in the database folders.

Output:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

 alias:matlab string variable with database alias.

```
> suspensionGenStruct = suspensionSetLeafVerticalStiffness (suspensionGenStruct, keys, leafForceData ,
```

function to set left/right leaf spring force curves to a suspension stored as sgs struct.

Output:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

Input:

suspensionGenStruct:matlab structure containing data storage for suspension model input file (.sgs).

 keys:cell matrix {2,n} containing the list of (tag ; name) strings required to resolve .sgs tree.

 leafForcedata:matrix [n,2] containing x-values (jounce) and y-values (force) as first and second column for leaf curve.

 side:matlab integer variable 0:left, 1:right.

Database Utilities

Function utility collected here interacts with VI-CarRealTime database, in order to publish a suspension.sgs file into a new or existing database.

```
> [suspgenFile , cfg] = publishSgsFileToDb (suspgenFileBase, cfg,targetSgsName ,targetDbAlias, target
```

function to publish an sgs file into a database (if database does not exists it will be automatically generated).

Output:

suspgenFile: matlab string with full path of published system .xml file.

cfg:cell array of cells with dimensions (n,2) containing the string pairs alias-path. With respect to input value, it will include the row relative to the new database.

Input:

suspgenFileBase:matlab string with full path of suspension.sgs file that will be published.

cfg:cell array of cells with dimensions (n,2) containing the string pairs alias-path.

targetSgsname:matlab string name of new suspension.sgs file name (if empty string the new system file will have the same name as the base file).

targetDbAlias:matlab string with the database alias that will be added.

targetDbPath:matlab string with the database path that will be added.

Analysis Utilities

The main function for analysis purposes is given by **runViSg** which runs a VI-SuspensionGen batch session using as input a .sgs file collected as *suspensionGenStruct* .

```
> [successFlag suspensionGenStruct] = runViSg(suspensionGenStruct, output,workingDir)
```

Output:

successFlag:flag to monitor solver status (1 = ok).

suspensionGenStruct:matlab struct variable containing vehicle model system file (.sgs).

Input:

suspensionGenStruct:matlab struct variable containing vehicle model system file (.sgs).

output:base name used for analysis results.

workingDir:name of temporary folder in which will be saved simulations files.

ResReader Tool

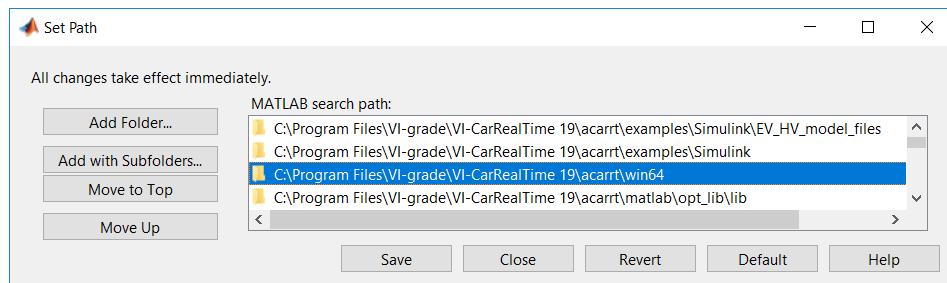
The ResReader tool is a mex function which allows to load the result (*.res) file of a VI-CarRealTime simulation into Matlab.

A structure embedding all the result set components defined in the result file is created in Matlab.

Note: all the result set component loaded in Matlab structure are automatically converted in SI (International System of Units).

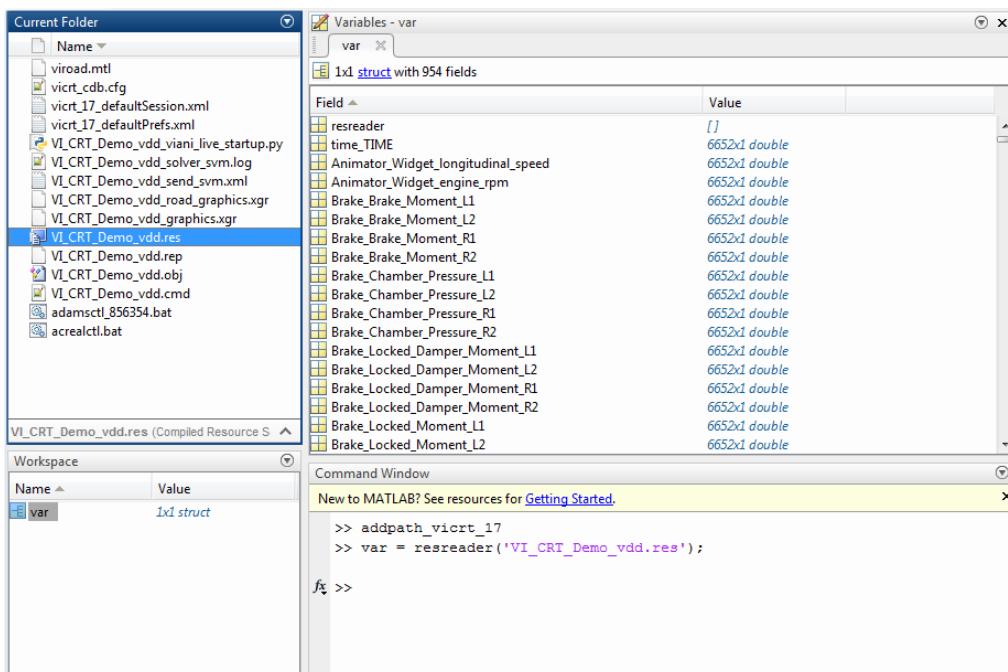
In order to use the ResReader in Matlab it is sufficient to type in Matlab Command Window the command `addpath_vicrt_20` which loads the VI-CarRealTime folders into Matlab session Paths:

VI-CarRealTime



In particular, the `<vicrt_installdir>\acarrt\win64` folder embeds the `resreader.mexw64` mex function. Once the paths have been set in MATLAB, it is possible to load a result file stored in any folder by issuing the command:

```
var = resreader ('resfilename.res');
```

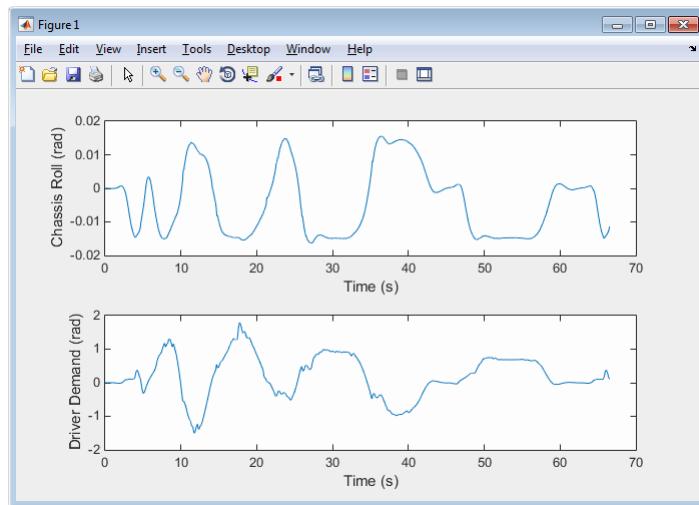


The structure of a result file is composed by a list of result sets and each result set has one or more result set components as children: for example the chassis roll angle is stored as `chassis_displacement.roll` where `chassis_displacement` is the result set and `roll` is the result set component child of the `chassis_displacement`.

Each result set component becomes a child of the res structure in MATLAB and its name is composed as: `<result_set_name>_<result_set_component_name>`.

To avoid problems with data management in MATLAB, all the dots (.) are replaced with underscores (_).

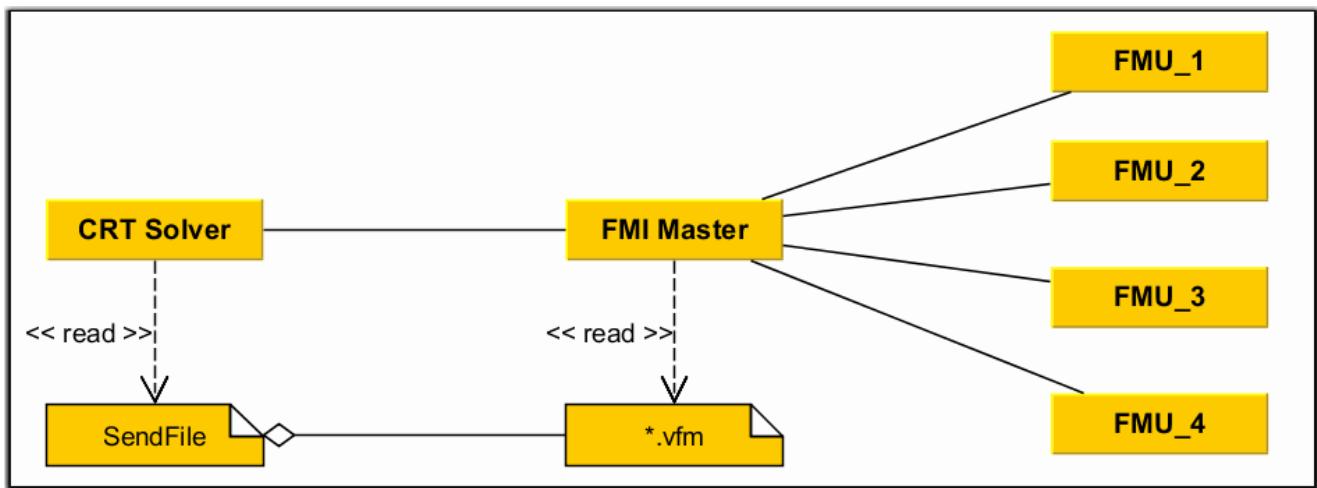
Once the res is loaded in the session, it is possible to manage, use and plot all the result set components in MATLAB environment.



1.3.4 Functional Mock-up Interface (FMI)

The [Functional Mock-up Interface](#) (FMI) is an open standard to support simulation models exchange Co-Simulation between different tools. Currently VI-CarRealTime includes an FMI Master that provides the Co-Simulation support for FMI 1.0 and 2.0.

Through the FMI Co-Simulation Master, VI-CarRealTime could import external models compliant to the FMI 1.0 and FMI 2.0 Co-Simulation standard. The model is also called *Functional Mock-up Unit* (FMU). VI-CarRealTime supports FMI Master via a specific auxiliary subsystem, through which the user could specify the configuration file and the enable/disable the FMI Master.

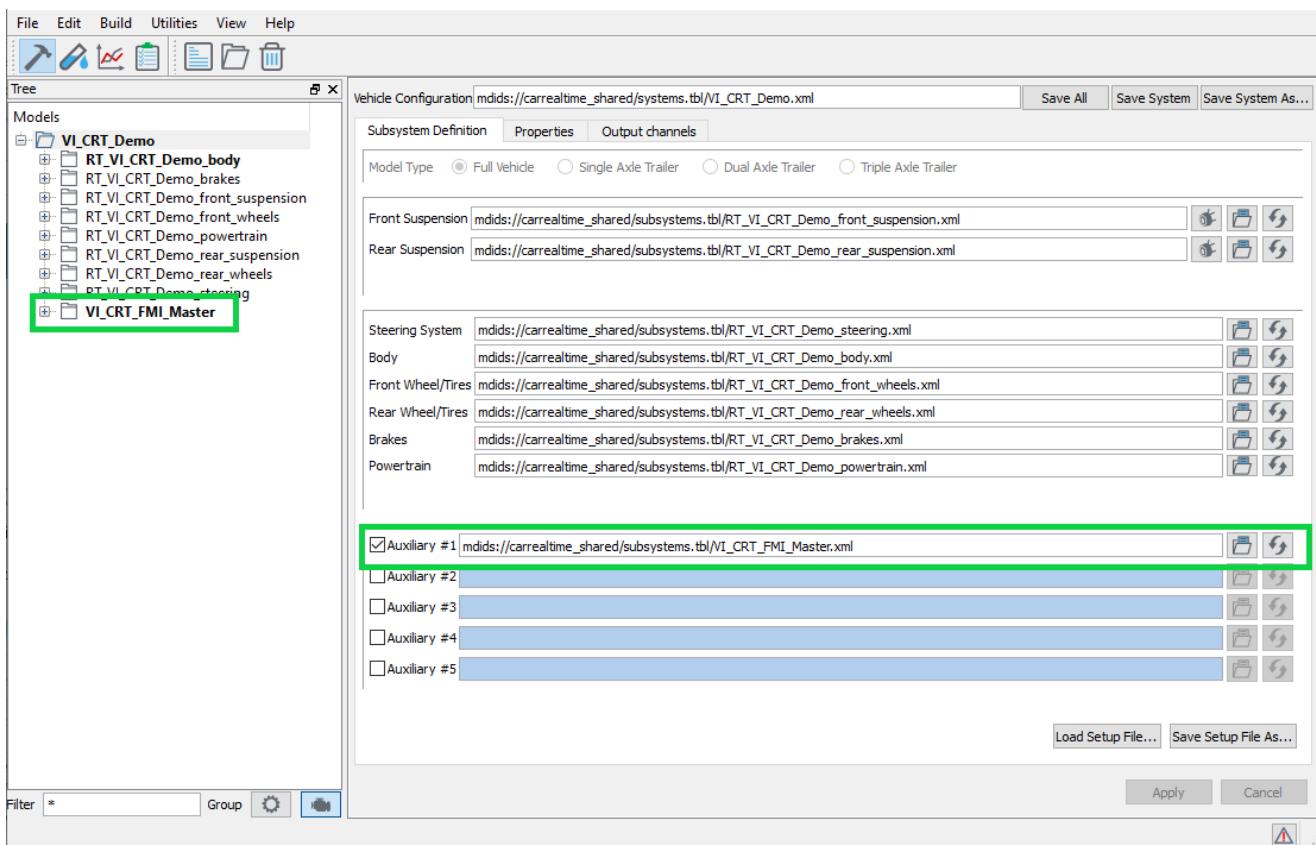


At every integration step, the FMI Master collects the outputs from VI-CarRealTime and integrates the FMUs according to their execution order.

Note: Currently the FMI Master can only manage compiled FMU and only the connection between VI-CarRealTime and the FMUs. FMU to FMU connection isn't allowed.

FMI Master Subsystem

The FMI Master can be used in VI-CarRealTime by including specific auxiliary subsystem file in the vehicle system file editor. The property file can be found in VI-CarRealTime distribution under `carrealtime_shared` database.



FMI Master requires a configuration file including the information related to FMU number, how they are connected to VI-CarRealTime and other FMU configurations (see [FMI Master configuration file](#)).

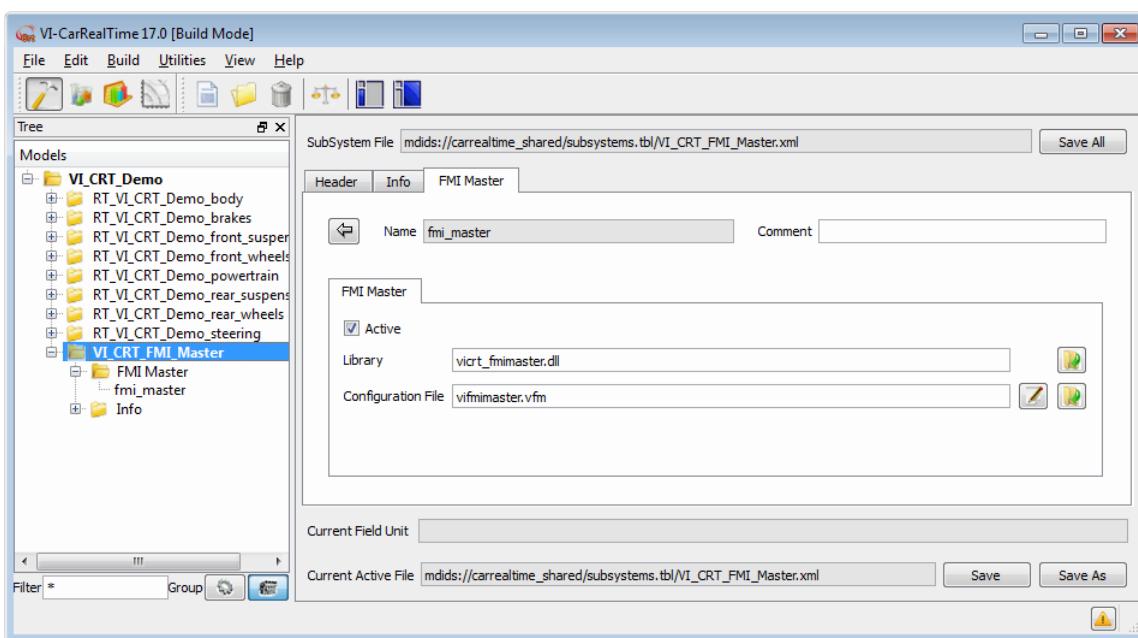
As shows the image below, FMI master subsystem has the following parameters:

Active

Activation flag that allows to enable and disable the FMI module without disabling the subsystem from the main property tree.

Configuration File

Full path of the FMI Master configuration file (vfm)



FMI Master Configuration File

The VI-grade FMI Master for VI-CarRealTime should be configured using a specific ASCII file in teim orbit format. The following pages describe the blocks defining how the FMUs are managed and connected to VI-CarRealTime.

- [VIGRADE_HEADER Block](#)
- [UNITS Block](#)
- [FMI_MASTER_PARAMETERS Block](#)
- [FMU Block](#)

VIGRADE_HEADER Block

The header block contains the general information about the teim orbit file like the version and the format. No user settings are currently supported in this block

```
[VIGRADE_HEADER]
FILE_TYPE      = 'vfm'
FILE_VERSION   = 9
FILE_FORMAT    = 'ASCII'
```

UNITS Block

The information included in each VFM file are expressed in the units system declared in the units block.

```
[UNITS]
LENGTH = 'meter'
ANGLE   = 'radians'
FORCE   = 'newton'
MASS    = 'kg'
TIME    = 'second'
```

FMI_MASTER_PARAMETERS Block

The FMI MASTER PARAMETERS block contains FMI Master general configuration parameters

FMU_TABLE

List of FMU blocks representing the FMUs managed by FMI_Master.

Example FMI MASTER PARAMETERS data block:

```
[FMI_MASTER_PARAMETERS]
(FMU_TABLE)
{ fmu_block }
'FMU_1'
'FMU_2'
'FMU_3'
```

FMU Block

The FMU block contains for each FMU managed by the FMI master the following information:

FMU_PATH = './sample.fmu'

FMU archive full path.

INSTANCE_NAME = 'fmu_example'

FMU instance name used by FMI Master to identify the FMU instance runtime.

EXECUTION_ID = '1'

Fmu execution order during the integration loop.

COMMUNICATION_STEP = '0.01'

Time interval between two communications (data exchange) with the current FMU during the integration loop.

NOTE: The COMMUNICATION_STEP must be multiple of the VI-CarRealTime integration step.

LOG_LEVEL = 'WARNING'

Filter on log messages severity level for the current FMU. The severity levels are:

'OFF': no message;

'ERROR': only error messages;

'WARNING': only warnings and error messages,

'VERBOSE': all the log messages.

INPUT_CONNECTIONS_MAP

Connection between VI-CarRealTime output channels and current FMU input channels.

The channels are identified by their names, retrievable from [VI-CarRealTime output channels list](#) and from the FMU specific documentation.

OUTPUT_CONNECTIONS_MAP

Connection between current FMU output channels and VI-CarRealTime input channels.

The channels are identified by their names, retrievable from [VI-CarRealTime input channels list](#) and from the FMU specific documentation.

INITIAL_CONDITIONS_MAP

Connection between current FMU Initial condition state and VI-CarRealTime initial condition channels available after statics.

The available Initial conditions on VI-CarRealTime side are:

- 'IC_CHASSIS_POS_X': X-coordinate of the vehicle chassis
- 'IC_CHASSIS_POS_Y': Y-coordinate of the vehicle chassis
- 'IC_CHASSIS_POS_Z': Z-coordinate of the vehicle chassis
- 'IC_CHASSIS_ANGLE_ROLL': chassis roll angle
- 'IC_CHASSIS_ANGLE_PITCH': chassis pitch angle
- 'IC_CHASSIS_ANGLE_YAW': chassis yaw angle
- 'IC_CHASSIS_VEL_X': chassis longitudinal speed
- 'IC_CHASSIS_VEL_Y': chassis lateral speed
- 'IC_CHASSIS_VEL_Z': chassis vertical speed
- 'IC_WHEEL_JOUNCE_L1': front left wheel jounce
- 'IC_WHEEL_JOUNCE_R1': front right wheel jounce
- 'IC_WHEEL_JOUNCE_L2': rear left wheel jounce
- 'IC_WHEEL_JOUNCE_R2': rear right wheel jounce
- 'IC_WHEEL_OMEGA_L1': front left wheel angular speed
- 'IC_WHEEL_OMEGA_R1': front right wheel angular speed
- 'IC_WHEEL_OMEGA_L2': rear left wheel angular speed
- 'IC_WHEEL_OMEGA_R2': rear right wheel angular speed
- 'IC_PWRTRN_ENGINE_TRQ': rear right wheel angular speed
- 'IC_PWRTRN_ENGINE_OMEGA': engine angular speed
- 'IC_PWRTRN_CLUTCH_OMEGA': clutch angular speed
- 'IC_PWRTRN_DIFFER_OMEGA': differential angular speed
- 'IC_DRV_GEAR': initial gear

PARAMETERS_MAP

Table FMU parameter name - value, used to change the FMU parameters. Parameter names are retrievable from the FMU documentation.

Example FMU data block:

```
[FMU_1]
FMU_PATH='.\Driveline2WD.fmu'
FMU_UNZIP_PATH=''
INSTANCE_NAME='FMU_Driveline'
EXECUTION_ID=1
COMMUNICATION_STEP=0.1e-3
LOG_LEVEL='WARNING'

(INPUT_CONNECTIONS_MAP)
{vicrt_output_chl fmu_input_chl}
'Wheel.Omega.L2'          'WL_omega'
'Wheel.Omega.R2'          'WR_omega'
'driver_demands.gear'     'GearNumber'
'driver_demands.clutch'   'Clutch'
'engine.torque'           'EngineTorque'

(OUTPUT_CONNECTIONS_MAP)
{fmu_output_chl vicrt_input_chl}
'WL_tau'                  'Wheel_driving.moment.L2'
'WR_tau'                  'Wheel_driving.moment.R2'
'CurrentGearRatio'        'GearBox.transmission_ratio'
'EngineOmega'              'Engine.engine_omega'

(PARAMETERS_MAP)
{fmu_parameter_name fmu_parameter_value}
>Main.Driveline.r1'      3.5
>Main.Driveline.r2'      1.94
>Main.Driveline.r3'      1.26
>Main.Driveline.r4'      0.93
>Main.Driveline.r5'      0.76
>Main.Driveline.r6'      0.675
>Main.Driveline.rD'      4.11
>Main.Driveline.J_e'     0.07

(INITIAL_CONDITIONS_MAP)
{fmu_initial_condition_name          vicrt_initial_condition_name}
>Main.Driveline.MD1.S1.w(t)'        'IC_WHEEL_OMEGA_L2'
>Main.Driveline.MD2.S1.w(t)'        'IC_WHEEL_OMEGA_R2'
>Main.Driveline.Engine1.T1.tau(t)'  'IC_PWRTRN_ENGINE_TRQ'
```

1.3.5 Customization

VI-CarRealTime vehicle model can be customized by means of the following tools:

- [Custom Events](#)
- [Custom Code](#)

Custom Events

VI-CarRealTime offers the capability of implementing custom events.

The event can be derived from basic VI-CarRealTime event specializing, though a specific python class, the simulation tasks and associating a graphical user interface to it.

Optionally the event can support a preliminary maneuver suitable for retrieving parameters needed for the dynamic event definition.

Both pre-event (optional) and main event are based on VI-Driver technology and rely on vdf file input.

VI-Driver input files are created from prototypes, by replacing placeholders keys using a specific utility.

The following steps are needed to add new custom events to the VI-CarRealTime environment.

- [Setting Up The Environment](#)
- [Event Class Methods](#)
- [Event GUI](#)

A [tutorial](#) can be used to follow step by step the procedure which allows to implement a new event and use it in VI-CarRealTime Test Mode.

Setting Up The Environment

The implementation of new events in VI-CarRealTime requires a specific folder including all the code needed for the customization

The environment customization can be stored on the computer in a separate folder from VI-CarRealTime installation.

The root path will include the following sub-folders:

- **events**

The folder contains the classes (.py files) needed to define new user events. Each file will implement a class and the methods needed to build up and run the custom event. In this sub-folder the user can add new events. Class source files can be organized in a user defined sub-folder structure that will reflect in the Test mode event tree.

- **vdf_prototypes**

The folder contains the Vi-Driver File Prototype (.vdp) used by the event class for generating the vdf files for the analyses. These file contains placeholders which are replaced with numeric/string values by the user event class.

- **visedit**

The folder contains the files needed to associate a graphical user interface to the event class each user event:

In order to register the custom root folder in the VI-CarRealTime environment, the user must set the following environment variable:

VICRT_CUSTOM = <root_folder path>

For example:

```
set VICRT_CUSTOM = %cd%/vicrt_custom
```

where *vicrt_custom* is the name given to the *root_folder* stored in current directory.

Event Class Methods

The *events* sub-folder stores a set of python files; each file implements a class which defines a custom event. Each class has methods that can be modified by the user to customize the event.

Each *user_event* class embeds the following methods:

- [__init__](#)
- [register](#)

VI-CarRealTime

- [getSolverSystem](#)
- [runPreSolver](#)
- [createPreVDF](#)
- [createVDF](#)
- [simResultsCompute](#)
- [createOutputChannelsFile](#)

In order to create a new event the fundamental methods to implement are:

```
def __init__(self, parent): ...
```

The method, automatically called when the class is instantiated, sets the default values for the main class members. The user can modify the default values for the class members declared by the register method.

```
def register(self): ...
```

The method is used to register member to the class, specifying for each member the name, the type and the units.

```
def getSolverSystem(self): ...
...
return top_class
```

The method retrieves the current active system parameters, set the name for the vdf files needed for the analyses, and call the createPreVDF and createVDF methods.

In case of need of a preliminary event the method runPreSolver should be also implemented:

```
def runPreSolver(self, solverExe): ...
...
return results
```

The method runs the pre-analysis event, retrieves the user-defined result components and generate the vdf file, updated with the pre-analysis outputs, for the main analysis.

It calls the following methods:

- createOutputChannelsFile
- eventToPreSendFile
- simResultsCompute
- createVDF

Note: Such method must return *True* or *False*: *True* means that everything was ok with pre-event, so the main event will be run, *False* stops the simulation preventing the solver to run the main event.

Other auxiliary methods can be implemented to support the ones listed above.

```
def createPreVDF(self, vdfFileName): ...
```

The method creates the vdf file for the pre-analysis maneuver: it replaces the placeholders stored in the vdp file, whose string name is stored in the vdpFile variable, with numeric values through the preVdpData dictionary.

```
def createVDF(self, vdfFileName, data = None): ...
```

The method creates the vdf file for the analysis: it replaces the placeholders stored in the vdp file, whose string name is stored in the vdpFile variable, with numeric values through the data dictionary.

```
def simResultsCompute(self, resultFile): ...
...
return resultValue
```

The method computes the simulation results: the user can manipulate this method in order to retrieve the desired results which will be fed to the runPreSolver method.

```
def createOutputChannelsFile(self, fileName): ...
...
return Boolean (True=success, False=fail)
```

The method creates a custom channels map file; it generates an output map controlling the number and names of results set components.

Additional functions or modules can be eventually invoked in the custom event implementation. VI-CarRealTime distribution includes the module `vicrt_event_utils` that may be useful.

The module can be loaded using the command:

```
from vigrade import vicrt_event_utils
```

and it is including the following functions:

- [vdpToVdf](#)
- [eventToSendFile](#)
- [createCustomOutputChannelsFile](#)
- [resultFileToComponent](#)
- [componentNameTold](#)
- [sortByComponent](#)

```
def vdfToVdf(vdp, vdf, data): ...
...
return status (True=success, False=fail)
```

The function reads the vdp (vi-driver file prototype) file, replacing the placeholders with the key value read in the data dictionary and writing out a new vdf file.

```
def eventToSendFile(eventObject, outputChannelsFile = None, vdf = None, outputPrefix =
'default'): ...
...
return Boolean (True=success, False=fail)
```

The function generates a send file from event instance by specifying vdf file and output map.

```
def createCustomOutputChannelsFile(fileName, chlLst, resSetLst, cmpLst ): ...
...
return Boolean (True=success, False=fail)
```

The function creates a custom channels map file.

It generates an output map setting the number and names of result set components.

```
def resultFileToComponent (infile, fullCmpNamLst): ...
...
```

VI-CarRealTime

```
return outCmp
```

The function extracts the components from results file and returns the output component array (outCmp).

```
def componentNameToInt (lines, fullCmpNam): ...
...
return int
```

The function returns the component id (int) in a result file; it returns False if the component ID is not found.

```
def sortByComponent (x,y,maneuverID=None, id=1, method = 'ascending' ): ...
...
return [outX,outY]
```

The function returns x and y vectors sorted in x ascending or descending order depending on *method* variable; the optional input parameters *maneuverID* (list) and *id* are used to filter the results.

VI-Driver File Prototypes

VI-Driver event prototypes can be used to implement specific events avoiding the burden of creating the input file from scratch.

Prototype file has the same format of VI-Driver input file (it can be created modifying an existing vdf) having the key

```
param(...)
```

in place of numerical or string values that are to be set in the resulting vdf.

The argument of the *param* key is used to identify the value to replace by the function `vdpToVdf` from [vicrt_event_utils](#) module.

Example of VI-Driver file prototypes can be found in VI-CarRealTime installation under acarrt/examples/Custom_Events/vdf_prototypes.

A snapshot of a section of a vdp file is shown below:

```

[CLUTCH_STANDARD]
MAX_VALUE = -1
MIN_VALUE = 0
SCALING_FACTOR = 1
MIN_RPM = 500
ACTIVATION = 'TRUE'
MODEL = 'STANDARD'

[STARTUP]
STARTING_TIME = 0
INITIAL_SPEED = param(initial_speed)
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRaking = 0
INITIAL_BRaking2 = 0
INITIAL_GEAR = param(initial_gear)
INITIAL_CLUTCH = 0
INITIAL_SETUP = 'OFF'
```

Event GUI

This section describes how to set up, define and connect a custom GUI to the related user event class.

The user is allowed to customize his own GUI, deriving a new class from a default layout defined in the `UserEventLayout` class; in particular the user can do the following:

- register the desired of custom GUI ([Custom Register](#))
- connect the GUI to the event ([Event Editor](#))

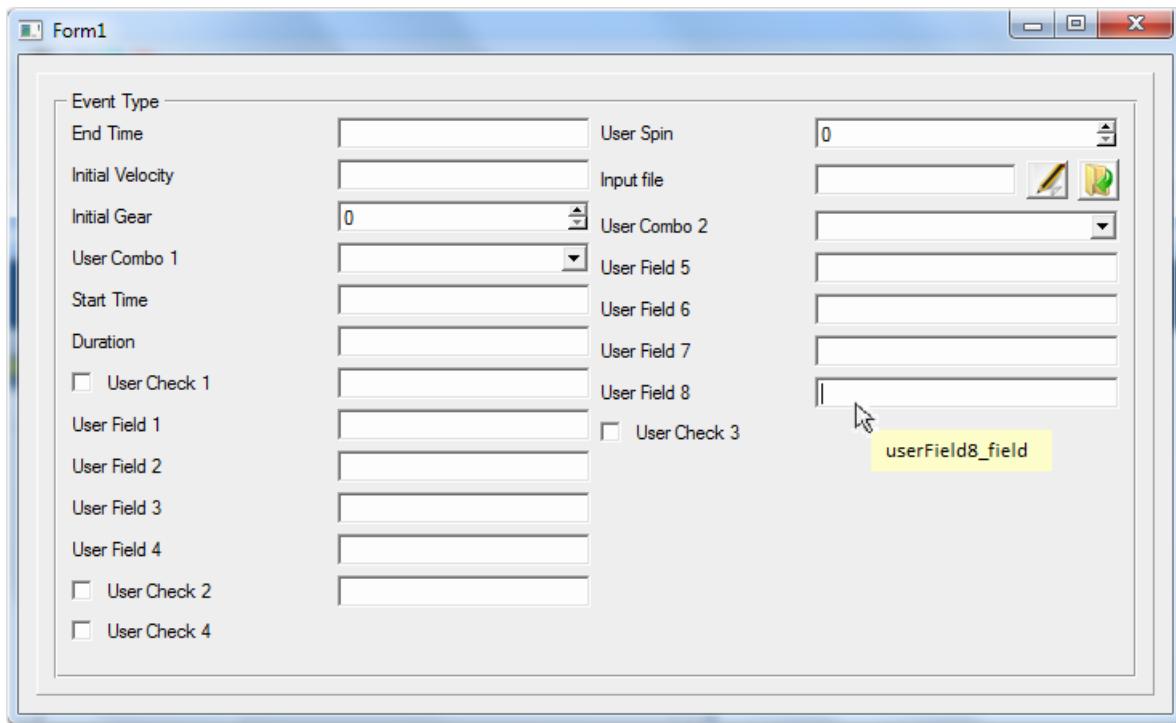
Event Layout

The `userEventLayout` class is embedded in the file `user_event_layout.pyc` located under `<inst_directory>/python/win32/Lib/site-packages/mdi/visedit`.

This class defines the layout of the prototype GUI; it is used as the base class for the definition of the [user event editor](#) class.

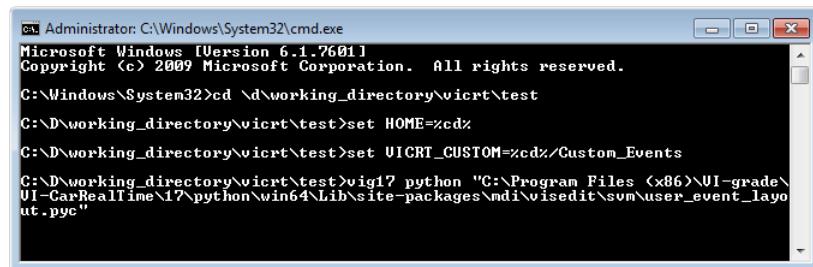
In the specific custom implementation all the entities not needed by the specific event can be hidden.

The following picture shows the GUI:



The user can visualize this GUI by typing the following command in a shell dos ([VICRT_CUSTOM](#) environment variable must be defined):

```
vig20 python <event_layout_file_name>
```



Tip: On windows systems it is possible to drag and drop the `user_event_layout.pyc` into the shell dos instead of manually typing all the path.

By loading the `user_event_layout.pyc` as a standalone widget it is possible to visualize the name of all GUI entities as object help tips.

Custom Register

The `customEditorsRegister` function is responsible for retrieving both the user event class and the related event editor class.

The function must be implemented in file `custom_editors_register.py` to be located in custom events folder under `/visedit`.

All the calls to the user event editor have to be added inside this function. An example is reported below:

```

if hasattr (svm_paths, "UserEvent") :
    g = EventGuiInfo ()
    g.editorClass   ("UserEventDialog")
    g.editorFile    ("user_event_editor")

```

UserEvent: class name of the user event;

UserEventDialog: class name of the user event editor to be associated to the UserEvent;

user_event_editor: file name of the user event editor.

Event Editor

The user event editor class enables the communication between user defined event class and the custom event GUI.

The purpose of this class is to display the fields related to the custom event and to enable the connection between the user event class members and the fields of the GUI.

Typically event editor class is stored in custom event folder under `/visedit`

Each user event editor class embeds the following methods:

- [__init__](#)
- [afcConnect](#)
- [load](#)
- [renameItems](#)
- [hideAll](#)

- [showItems](#)

```
def __init__(self, parent = None, name = None, modal = 0, fl = 0): ...
```

The method, automatically called when the class is instantiated, performs the following tasks:

- call the parent class [__init__](#) method;
- call the [afcConnect](#) method;
- call the [hideAll](#) method;
- call the [renameItems](#) method;
- call the [showItems](#) method;
- call the parent class show method to display itself.

```
def afcConnect(self): ...
```

The method is used to connect the objects (fields, combo boxes, spin boxes, ecc..) of the generic custom GUI with the corresponding members of the user event class.

```
def load(self, object): ...
```

The method is automatically called and it loads all the items instantiated in the [afcConnect](#) method.

```
def renameItems(self): ...
```

The method renames the labels of the generic custom GUI through the setText method. It can be used to show consistent label names to the current custom event.

```
def hideAll(self): ...
```

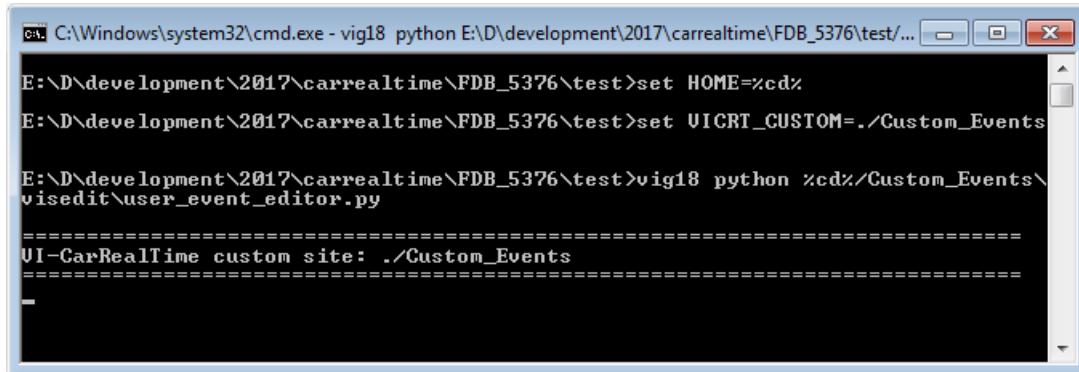
The method hides all the objects (labels, fields, check-boxes, ecc..) which are children of the event GUI.

```
def showItems(self): ...
```

The method shows the custom GUI objects requested for the particular event.

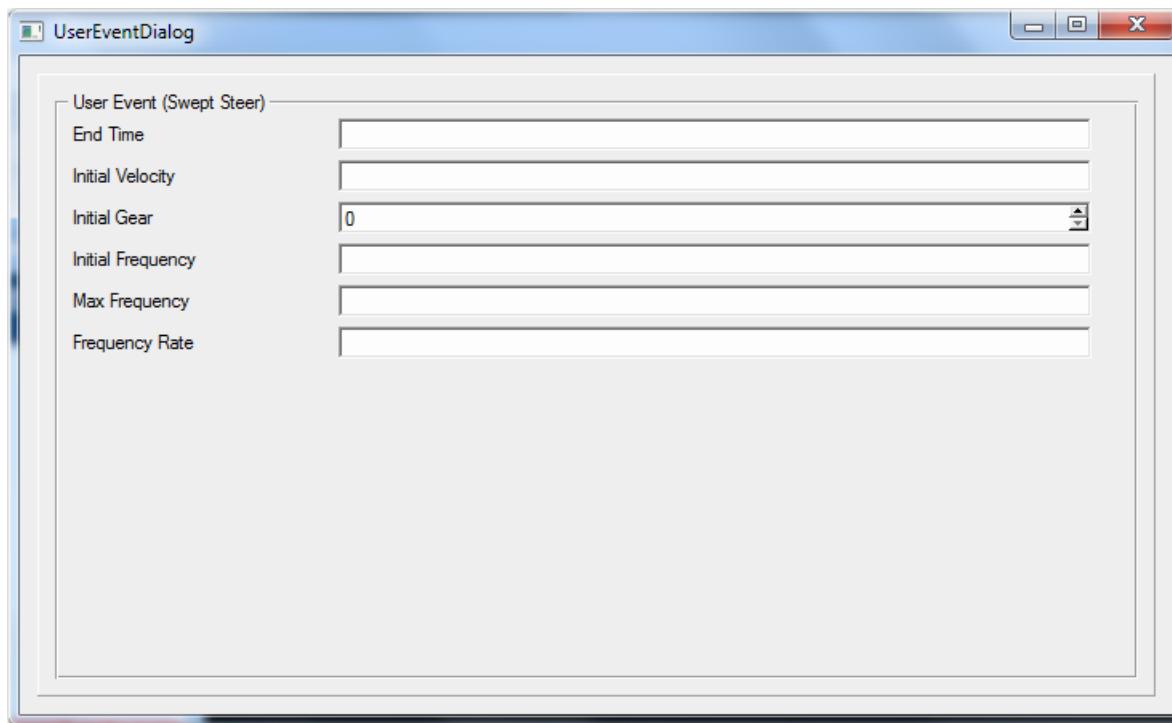
While the user is building up the event editor, he might want to verify how the custom GUI looks. The custom GUI can be shown though the following command in a shell dos (the [VICRT_CUSTOM](#) environment variable has to be defined):

```
vig20 python <event_editor_file_name>
```



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe - vig18'. The command entered is 'python E:\D\development\2017\carrealtime\FDB_5376\test>vig18 python ./Custom_Events\visedit\user_event_editor.py'. The output shows the command being run and the message 'UI-CarRealTime custom site: ./Custom_Events' displayed at the bottom of the window.

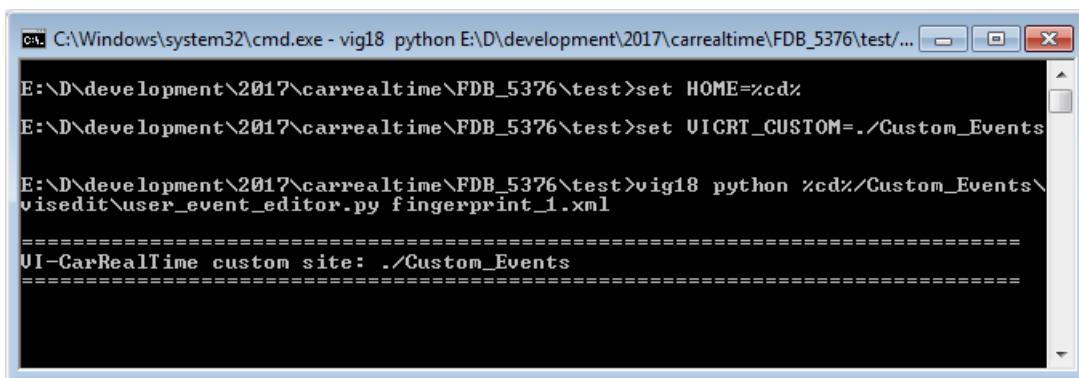
The current state of the dialog box will be shown:



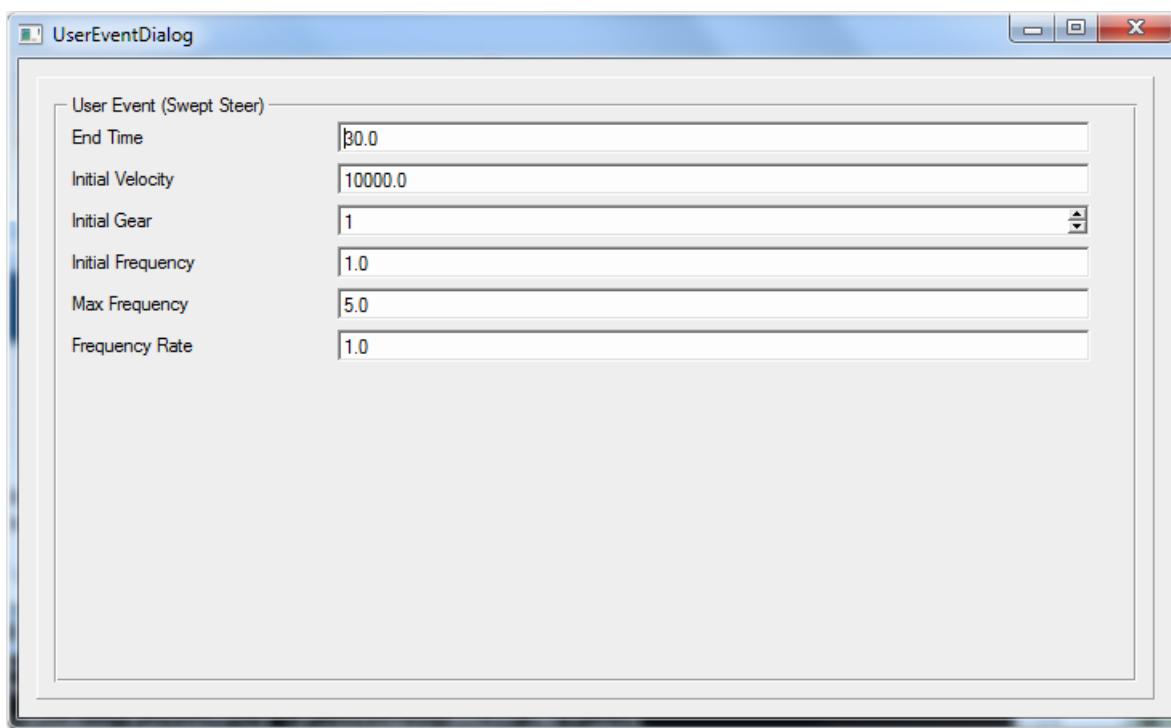
There is also the possibility to show up the custom GUI with its fields filled with the values read in a previously saved fingerprint.

The command to be written is the following:

```
vig20 python <event_editor_file_name> <fingerprint_xml_file>
```



The values are read in the xml file and are assigned to the related field of the GUI.



Error messages are typically shown in launch command window and may be investigated to debug the editor code during implementation.

Custom Code

VI-CarRealTime model can be customized by means of an external plug-in, modeled as a compiled dll, which is referenced in the model [system](#) or a generic [auxiliary subsystem](#).

The dll must embed all the functions needed for communicating with VI-CarRealTime solver in order to exchange information and data at different simulation stages; in general all the VI-CarRealTime [inputs](#) and [outputs](#) channels can be managed with a plug-in dll.

The source files which are used to compile the dll will be written in C or C++.

In VI-CarRealTime installation folder, under `\acarrt\examples\SolverPlugin`, the `solverPlugin_template.cpp` source file is stored: this file guides the user about what functions are to be used to communicate with VI-CarRealTime solver and where (inside which function) to write the custom code.

Note: Please refer to [VI-CarRealTime Customization](#) tutorials to see complete examples about how to compile dll starting from source code and then use dll in a VI-CarRealTime simulation event.

The `solverPlugin_template.cpp` defines the following functions:

- `extcall_load`
- `extcall_set_user_input`
- `extcall_set_user_output`
- `extcall_set_input`
- `extcall_set_static_input`
- `extcall_sync`
- `extcall_get_output`
- `extcall_get_static_output`
- `extcall_init_model`
- `extcall_init_static`
- `extcall_save`
- `extcall_restore`
- `extcall_terminate`

```
//  
// This function is called only ONCE, at the beginning of the simulation,  
// immediately after reading the model data from the input xml file.  
// If the plugin needs to read input from file or to perform some preliminary  
// operation, this is the right place where the code should be added.  
//  
PUBLIC EXPORT int extcall_load( void *crtptr )  
//  
// INPUT  
// void* crtptr: pointer to the VI-CarRealTime internal data structure. Needed to  
// initialize the API interface of the plugin.  
// OUTPUT  
// int status: if 0 error else success  
{  
    // IMPORTANT note: The following call is mandatory to access  
    // the VI-CarRealTime API.  
    int status = (crt_set_instance(crtptr));  
  
    return status;  
}  
  
//  
// This function is called only ONCE, at the beginning of the simulation,  
// immediately after plugin dll load call.  
// If the plugin needs to declare additional input channels this is the right place where the cod  
//  
PUBLIC EXPORT int extcall_set_user_input()  
{  
    // IMPORTANT note: The following calls are mandatory to create additional input channels.  
    std::vector< ExtCallInpVars > userInput;  
    userInput.clear();  
    // create input  
    ExtCallInpVars tmpInput;  
    sprintf(tmpInput.name,"test_input_1");  
    sprintf(tmpInput.type,"velocity");  
    tmpInput.guess=0.0;  
    userInput.push_back(tmpInput);  
  
    sprintf(tmpInput.name,"test_input_2");  
    sprintf(tmpInput.type,"displacement");  
    tmpInput.guess=0.0;  
    userInput.push_back(tmpInput);  
  
    sprintf(tmpInput.name,"test_input_3");  
    sprintf(tmpInput.type,"force");  
    tmpInput.guess=0.0;  
    userInput.push_back(tmpInput);  
  
    sprintf(tmpInput.name,"test_input_4");  
    sprintf(tmpInput.type,"torque");  
    tmpInput.guess=0.0;  
  
    userInput.push_back(tmpInput);  
  
    return crt_set_solver_additional_input(userInput);  
}  
  
//  
// This function is called only ONCE, at the beginning of the simulation,  
// immediately after plugin dll load call.  
// If the plugin needs to declare additional output channels this is the right place where the co  
//
```

```

PUBLIC EXPORT int extcall_set_user_output()
{
    // IMPORTANT note: The following calls are mandatory to create additional input channels.
    std::vector< ExtCallOutvars > userOutput;
    userOutput.clear();
    // create output
    ExtCallOutvars tmpOutput;
    sprintf(tmpOutput.name,"out_1");
    sprintf(tmpOutput.result_set,"test_out");
    sprintf(tmpOutput.units,"meter/second");
    tmpOutput.enabled = true;
    tmpOutput.isLive=0;
    userOutput.push_back(tmpOutput);

    sprintf(tmpOutput.name,"out_2");
    sprintf(tmpOutput.result_set,"test_out");
    sprintf(tmpOutput.units,"meter");
    tmpOutput.enabled = true;
    tmpOutput.isLive=0;
    userOutput.push_back(tmpOutput);

    sprintf(tmpOutput.name,"out_3");
    sprintf(tmpOutput.result_set,"test_out");
    sprintf(tmpOutput.units,"Newton");
    tmpOutput.enabled = true;
    tmpOutput.isLive=0;
    userOutput.push_back(tmpOutput);

    sprintf(tmpOutput.name,"out_4");
    sprintf(tmpOutput.result_set,"test_out");
    sprintf(tmpOutput.units,"Newton*meter");
    tmpOutput.enabled = true;
    tmpOutput.isLive=0;
    userOutput.push_back(tmpOutput);

    return crt_set_solver_additional_output(userOutput);
}

// This function is called BEFORE each integration step.
// If the plugin need to set input to VI-CarRealtime, it should be done here
//
PUBLIC EXPORT int extcall_set_input( double *inputvec, int inputvecSize )
//
// INPUT
//   double* inputvec: pointer to the VI-CarRealTime input array, as described
//                     in svm_ii_FV_io_map.h and
//   int inputvecSize: size of the input array
// OUTPUT
//   int status: if 0 error else success
{
    //
    // place your code here
    //
    // IMPORTANT note: The following calls is mandatory to access additional input channels.
    double* aux_input = crt_get_auxplugin_input();
    return 1;
}

// This function is called BEFORE each static integration step.
// If the plugin need to set input to VI-CarRealtime, it should be done here
//
PUBLIC EXPORT int extcall_set_static_input( double *inputvec, int inputvecSize )

```

VI-CarRealTime

```
//  
// INPUT  
//   double* inputvec: pointer to the VI-CarRealTime input array, as described  
//   in svm_ii_FV_io_map.h and  
//   int inputvecSize: size of the input array  
// OUTPUT  
//   int status: if 0 error else success  
{  
    // IMPORTANT note: The following calls is mandatory to access additional input channels.  
    double* aux_input = crt_get_auxplugin_input();  
    //  
    // place your code here  
    //  
    return 1;  
}  
  
//  
// This function is called BEFORE each integration step, AFTER extcall_set_input  
// If the plugin need to run in realtime os it must be in sync with an external  
// timing source, the synchronization calls should be done here  
//  
PUBLIC EXPORT int extcall_sync()  
// OUTPUT  
//   int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    return 1;  
}  
  
//  
// This function is called AFTER each integration step.  
// If the plugin need to retrieve output from VI-CarRealtime, it should be  
// done here  
//  
PUBLIC EXPORT int extcall_get_output(double currtime, double *outputvec, int outputvecSize)  
// OUTPUT  
//   int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    // IMPORTANT note: The following calls is mandatory to access additional output channels.  
    double* aux_output = crt_get_auxplugin_output();  
    return 1;  
}  
  
// This function is called AFTER each static integration step.  
// If the plugin need to retrieve output from VI-CarRealtime, it should be  
// done here  
//  
PUBLIC EXPORT int extcall_get_static_output(double currtime, double *outputvec, int outputvecSize)  
// OUTPUT  
//   int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    // IMPORTANT note: The following calls is mandatory to access additional output channels.  
    double* aux_output = crt_get_auxplugin_output();  
    return 1;  
}
```

```
//  
// This function is called only ONCE, at the beginning of the simulation.  
// This is the last call before the start of the integration.  
// If the plugin needs to initialize a timer, or to perform any other operation  
// to enable the execution, it should be done here.  
//  
PUBLIC EXPORT int extcall_init_model( )  
// OUTPUT  
//    int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    return 1;  
}  
  
//  
// This function is called only ONCE, at the beginning of the static simulation.  
// This is the last call before the start of the simulation static integration.  
// If the plugin needs to initialize a timer, or to perform any other operation  
// to enable the execution, it should be done here.  
//  
PUBLIC EXPORT int extcall_init_static( )  
// OUTPUT  
//    int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    return 1;  
}  
  
//  
// This function is called AFTER each solver restore call ( for press maneuvers & mxp events ).  
// If the plugin need to restore internal states it should be  
// done here  
// Key input represent the unique id for the restoring configuration  
PUBLIC EXPORT int extcall_restore( double key )  
// OUTPUT  
//    int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    return 1;  
}  
  
//  
// This function is called AFTER each solver save call ( for press maneuvers & mxp events ).  
// If the plugin need to save internal states it should be  
// done here  
// Key input represent the unique id for the saving configuration  
PUBLIC EXPORT int extcall_save( double key )  
// OUTPUT  
//    int status: if 0 error else success  
{  
    //  
    // place your code here  
    //  
    return 1;  
}
```

```

// This function is called AFTER each solver save call ( for press maneuvers & mxp events ).  

// If the plugin need to save internal states it should be  

// done here  

// Key input represent the unique id for the saving configuration  

PUBLIC EXPORT int extcall_save( double key )  

    // OUTPUT  

    //     int status: if 0 error else success  

{  

    //  

    // place your code here  

    //  

    return 1;  

}  
  

// This function is called AFTER each solver restore call ( for press maneuvers & mxp events ).  

// If the plugin need to restore internal states it should be  

// done here  

// Key input represent the unique id for the restoring configuration  

PUBLIC EXPORT int extcall_restore( double key )  

    // OUTPUT  

    //     int status: if 0 error else success  

{  

    //  

    // place your code here  

    //  

    return 1;  

}  

//  

// This function is called only ONCE, at the end of the simulation, immediately  

// before the plugin is unloaded.  

// If the plugin needs to cleanup memory, release a license, write buffered data  

// or perform any other termination operation, it should be done here.  

//  

PUBLIC EXPORT int extcall_terminate()  

{
    //  

    // place your code here  

    //  

    return 1;  

}

```

VI-CarRealTime also provides an interface both for tires and aerodynamics as described in the following topics:

- [Interfacing Custom Tire Model](#)
- [Interfacing Custom Aero Forces](#)

Interfacing Custom Tire Model

The interface between VI-CarRealTime and the tire model consists on a function call, whose name is specified by the user and is stored in the tire property file as explained in the [Creating Custom Tire Library tutorial](#).

The interface function is called once per each tire during the different phase of the simulation (initialization, computation, termination), with different computation and simulation modes. Tire Forces and Torques at hub are returned in the `wheelForces` and `wheelTorques` variables. All the other quantities related to tire calculation are returned in the `outputValuesArray` array. The indexes of the values to be set in the `outputValuesArray` array are defined in the file `tutl_outvarptr.h` stored in the `acart/user_api` folder of the VI-CarRealTime installation directory.

For further details about the tire interfaces available look here.

Interfacing Custom Aero Forces

The interface between VI-CarRealTime and the aerodynamic model consists of the following function call (function name is specified by the user, and is stored in the `aero` property file as explained in the [Creating Custom Aerodynamics Force Library tutorial](#)):

```
/***
 *  STANDARD (user) function interface for VI-grade aerodynamics forces
 *
 * @param aerodynamicsID  the aerodynamics force ID
 * @param simulationMode  the simulation mode (static/dynamic)
 * @param computationMode the computation mode (init/standard)
 * @param propertiesFile  the properties file name
 * @param inputsArray     the runtime input parameters array [!--SI UNITS--!]
 * @param modifiersArray  the modifiers parameters array [!--SI UNITS--!]
 *
 * @return outputArray    the output parameters array [!--SI UNITS--!]
 * @return errorID        the error identifier (0 => NO ERROR)
 * @return workArray      the work parameters array (I/O storage) -!not used!-
 *
 * @see input array map: aero_inputpar.h
 * @see input array map: aero_modspar.h
 * @see simulation/computation mode: comp_commoncdef.h
 * @see output array map: aero_outputpar.h
 */
PUBLIC void aero_cpp_usersample(int      aerodynamicsID,
                                int      simulationMode,
                                int      computationMode,
                                char*   propertiesFile,
                                double* inputArray,
                                double* modifiersArray,
                                double* outputArray,
                                int*    errorID,
                                double* workArray      )
```

This function is called once per each simulation step (initialization and computation), with different computation and simulation modes. At the moment the simulation mode (`simulationMode`) is unused, while the computation mode (`computationMode`) is used to distinguish between initialization and standard computation calls. The property file name can be used to retrieve the user defined parameters.

The files "aero_inputpar.h" and "aero_outputpar.h" (stored in the `acarrt/user_api` folder of the VI-CarRealTime installation directory) contain the map of the input and output array. Refer to the sample files provided for additional information.

1.3.6 VI-CarRealTime Solver API

VI-CarRealTime provides to the user a set of C (and C++) api, that can be used to properly interface the internal vehicle model with externally modeled subsystems, using the Solver Plugin approach, or to create a custom executable to drive the model initialization and simulation.

API documentation can be reached through the following link: [VI-CarRealTime Solver API](#)

1.3.7 External Models

The following models are described in order for the user to define his own external model and connect it with VI-CarRealTime model:

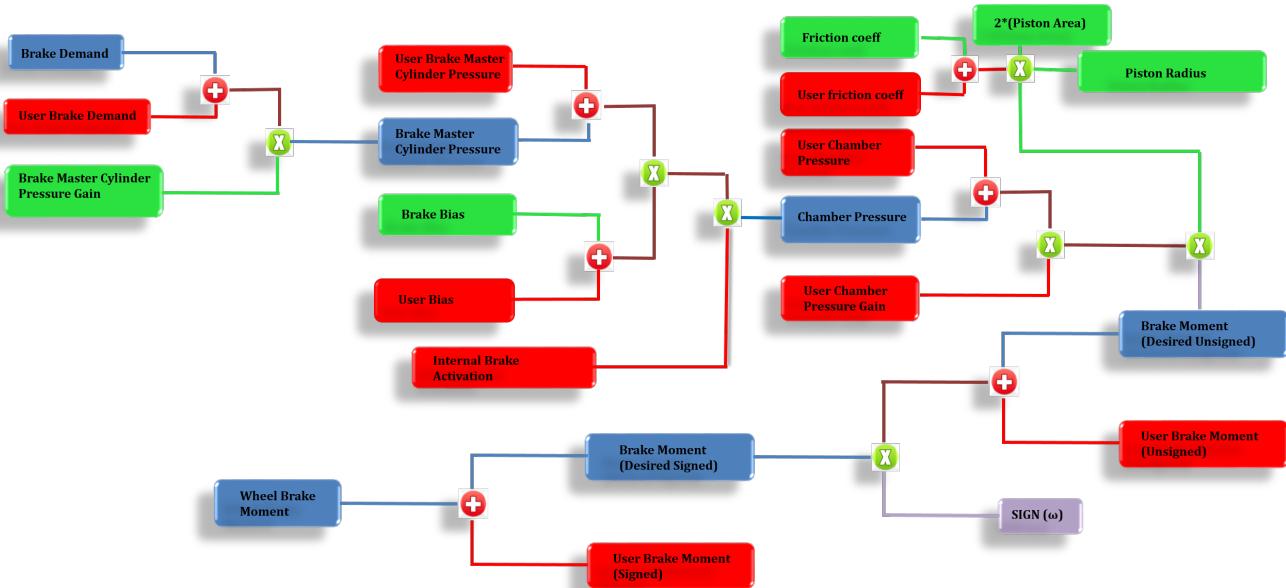
- [About External Brake Model](#)
 - [About External Powertrain Model](#)
 - [About External Suspension Model](#)

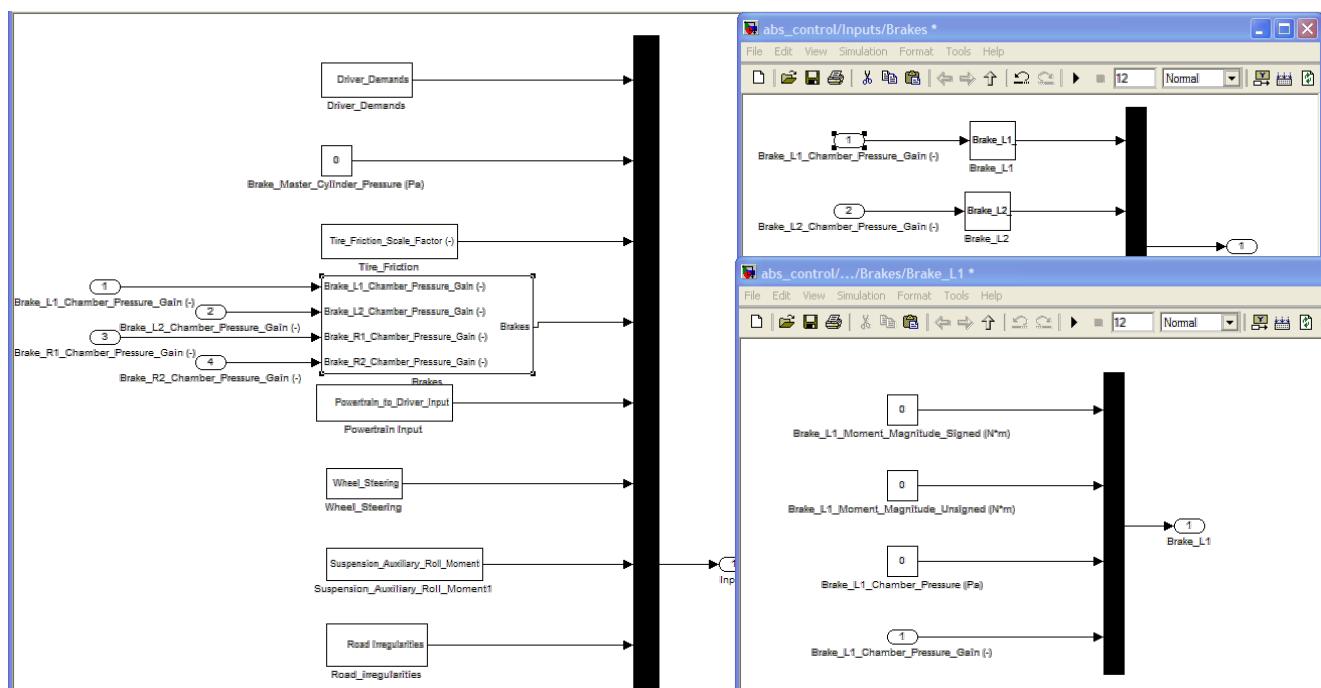
About External Brake Model

Brake system model in VI-CarRealTime allows the user to customize inputs at different levels:

- User Brake Demand;
 - User Brake Master Cylinder Pressure;
 - User Chamber Pressure;
 - User Chamber Pressure Gain;
 - User Friction Coefficient;
 - User Brake Moment (unsigned);
 - User Brake Moment (signed);
 - User Bias;
 - Internal Brake Activation.

Diagram below shows how the different user inputs (red background) are fed into the computation model:





About External Powertrain Model

Customizing powertrain model in VI-CarRealTime is possible as well. Powertrain consists of four different subsystems - engine, clutch, gearbox, differential - and each of them can be replaced by a custom model.

Built in model for these subsystems are disabled through the flags available in the [Powertrain Parameters](#) section of the model parameter tree, except for the differentials, whose activity or typology are specified in the [Differential](#) section of the Powertrain Subsystem.

In order to properly interact with the VI-CarRealTime model and with the embedded Virtual driver, the custom subsystems need to return a predefined set of values to the vehicle model:

- **Engine model**

Custom engine model must return the following values:

- Maximum Engine Torque;
- Minimum Engine Torque;
- Actual Engine Torque.

- **Clutch model**

Custom Clutch model must return the following values:

- Clutch Angle (integral of clutch slip, used output only);
- Clutch torque;
- Engine Omega.

- **Gearbox model**

Custom Gearbox model must return the following values:

- Gear Ratio

- **Differential model**

Custom Differential model(s) must return the following values:

- Differential torque (i.e. the delta torque with respect to an open differential (50% torque split)).

For example, in case only the differential should be modeled in Simulink, a user has to select *Open* as differential method and pass the amount of torque split by the differential (L/R delta torque with proper sign depending on side) to the user wheel drive torque input channel .

About External Suspension Model

Suspension kinematics description

VI-CarRealTime vehicle model consists of a sprung mass (optionally split in two halves) and 4 unsprung masses. Each unsprung mass has 2 degrees of freedom, the vertical displacement with respect to the sprung mass and the wheel spin.

The position of each of the sprung masses, with respect to its design position, is defined by the following expression:

$$KinPosition = x(z, v_1, v_2)\vec{s}_x + y(z, v_1, v_2)\vec{s}_y + z\vec{s}_z$$

where z is the wheel jounce and $\vec{s}_x, \vec{s}_y, \vec{s}_z$ are the versors of the sprung mass reference system. The x and y component are defined as a function of wheel jounce (z) and of two additional independent variables (v_1 and v_2), that, depending on the type of suspension can be either the steering (in terms of steering wheel angle or rack displacement), the jounce of the opposite side or both of them.

In the same way, the orientation of the sprung masses is defined as function by the following expression:

$$KinOrientation = f(sva, toe, camber)$$

where side view angle, toe and camber are again function of three independent variables:

$$sva = sva(z, v_1, v_2)$$

$$toe = toe(z, v_1, v_2)$$

$$camber = camber(z, v_1, v_2)$$

The same dependencies also apply to the derivatives of the kinematic position and orientation. For example if we look at the time derivative of the position, we get:

$$KinVelocity = \left(\frac{\delta x}{\delta z} \dot{z} + \frac{\delta x}{\delta v_1} \dot{v}_1 + \frac{\delta x}{\delta v_2} \dot{v}_2 \right) \vec{s}_x + \left(\frac{\delta y}{\delta z} \dot{z} + \frac{\delta y}{\delta v_1} \dot{v}_1 + \frac{\delta y}{\delta v_2} \dot{v}_2 \right) \vec{s}_y + z \dot{\vec{s}}_z$$

The jacking forces generated by the suspension kinematics when submitted to an external load (i.e. tire forces) depends on the derivatives of the x/y/sva/toe/camber function with respect to all the independent variables.

External suspension

The suspension characteristic in VI-CarRealTime are described by interpolating splines having as independent variable the suspension jounce, the steering wheel angle and the opposite suspension jounce (the two last independent variable are optional).

In order to allow the integration of an external suspension model VI-CarRealTime exposes a quite large number of inputs to allow the suspension replacement on both subsystem level or component level. To activate the additional input for external suspension interface, you need to activate the specific switches available in the system properties tree, as described in [External Suspension](#).

The entire list of inputs that are exposed when the one or more of the switches are activated is available in [Input Channels In VI-CarRealTime](#)

External suspension models can be coupled to VI-CarRealTime using one of the available interfaces: [MATLAB/Simulink](#), [Functional Mock-up Interface \(FMI\)](#), [Custom code \(Solver Plugin\)](#).

There are two different approaches to integrate an external model of suspension kinematic into VI-CarRealTime.

- **component replacement:** in case you only want to replace of one (or more) components of the the suspension kinematics, you need to provide VI-CarRealTime not only with the value of the specified component (camber for example), but also with its derivatives with respect to the independent variables, that allows the suspension code to properly compute the suspension jacking forces. This approach can be used if you want to couple an external suspension kinematics solver to the VI-CarRealTime vehicle model.
- **subsystem replacement:** in case you want to replace the entire suspension, kinematic or elastokinematic plus all the suspension elements(spring, dampers, antirollbar...) you need to provide to VI-CarRealTime:
 - (a) the suspension characteristic and their time derivatives (used to properly evaluate the hub position an velocity, needed for tire forces computation);
 - (b) the force acting on th suspension for each of the suspension element;
 - (c) the jacking force produced by the suspension as a consequence of the applied forces and moments. This approach can easily be used every time you want to couple a full suspension subsystem model to the VI-CarRealTime vehicle model. An example that demonstrate this approach, using the [Functional Mock-up Interface \(FMI\)](#) is available in the [FMI: External Suspension](#) tutorial.

1.4 Tutorials

The following VI-CarRealTime tutorial section comprises a series of step by step exercises, which users should follow in order to become more familiar with the most common tasks performed using the VI-CarRealTime program.

Major sections that are covered are:

1. [VI-CarRealTime Interface](#)
 - [Getting Started with VI-CarRealTime](#)
 - [Export Vehicle from Adams Car](#)
 - [Run Analyses in VI-CarRealTime](#)
 - [Using VI-Driver/MaxPerformance](#)
 - [Using MF-Tyre/MF-Swift tires](#)
 - [Fuel Consumption](#)
 - [Using Vehicle Wizard](#)
 - [Using K&C Wizard](#)
 - [Running a Vehicle-Trailer analysis](#)
 - [Using CarMaker Converter](#)
 - [Simulating a DRS System](#)
 - [Running Investigation](#)

- [Using 7Post Rig](#)

2. MATLAB interface

[Co-Simulation](#)

- [Implementing ABS controller in Simulink](#)
- [Replacing VI-CarRealTime standard components](#)
- [Assessing vehicle active systems](#)
- [Implementing a damper skyhook control system](#)
- [Using VI-Driver/MaxPerformance in MatLab/Simulink](#)
- [Using VI-Driver/PressManeuvers in MatLab/Simulink](#)
- [Using External Road](#)
- [Implementing an Electric Vehicle](#)
- [Implementing an Hybrid Vehicle](#)
- [Connecting External Steering Model](#)

[VI-CarRealTime Solver Plugin Export from Simulink](#)

[Run Investigation with a Solver Plugin](#)

[API Toolkit](#)

- [Manage a VI-CarRealTime Model](#)
- [Manage a VI-SuspensionGen Model](#)
- [Suspension Optimization](#)

3. FMI Master

- [FMI: External Driveline](#)
- [FMI: External Suspension](#)

4. VI-CarRealTime Customization

- [Creating Solver Plugin](#)
- [Creating Custom Tire Library](#)
- [Creating Custom Aerodynamics Force Library](#)
- [Creating Custom Event](#)

1.4.1 VI-CarRealTime Interface

The tutorials found in the links below are intended to teach a new VI-CarRealTime user how to complete a few elementary tasks, namely:

- Becoming familiar with the main VI-CarRealTime interface
- Managing the parameters of a given vehicle model
- Submitting required analyses
- Reviewing the results of analyses, using animation(s) and/or corresponding plots

The following topics will be covered:

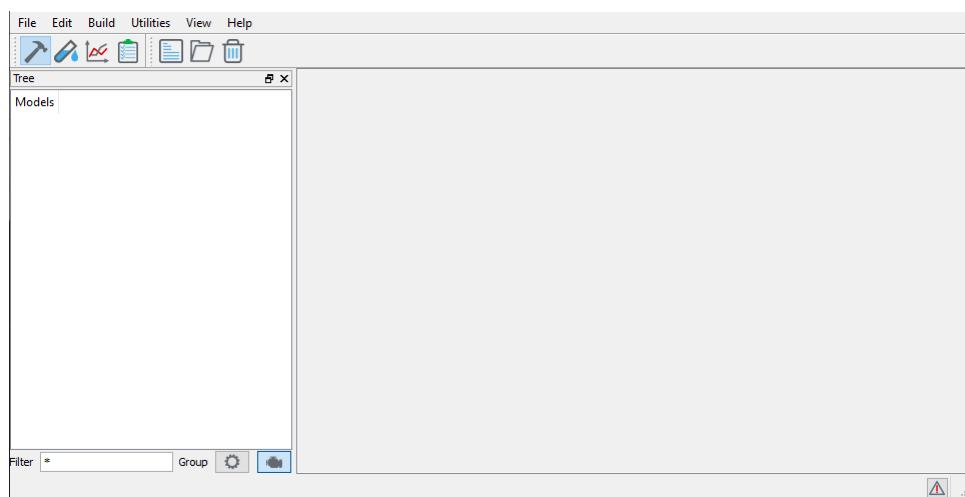
- [Getting Started with VI-CarRealTime](#)
- [Export Vehicle from Adams Car](#)
- [Run Analyses in VI-CarRealTime](#)
- [Using VI-Driver/MaxPerformance](#)
- [Using MF-Tyre/MF-Swift tires](#)
- [Fuel Consumption](#)
- [Using Vehicle Wizard](#)
- [Using K&C Wizard](#)
- [Running a Vehicle-Trailer analysis](#)
- [Using CarMaker Converter](#)
- [Simulating a DRS System](#)
- [Running Investigation](#)
- [Using 7Post Rig](#)

Getting Started

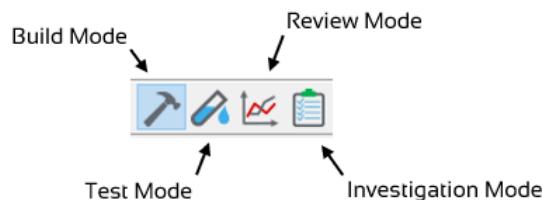
The following tutorial gives you an overview of the basic VI-CarRealTime capabilities including model management, analysis, and post-processing of results.

Start a new VI-CarRealTime session using either the Windows Start Menu, or by typing **vicrt20** in a DOS (cmd) shell.

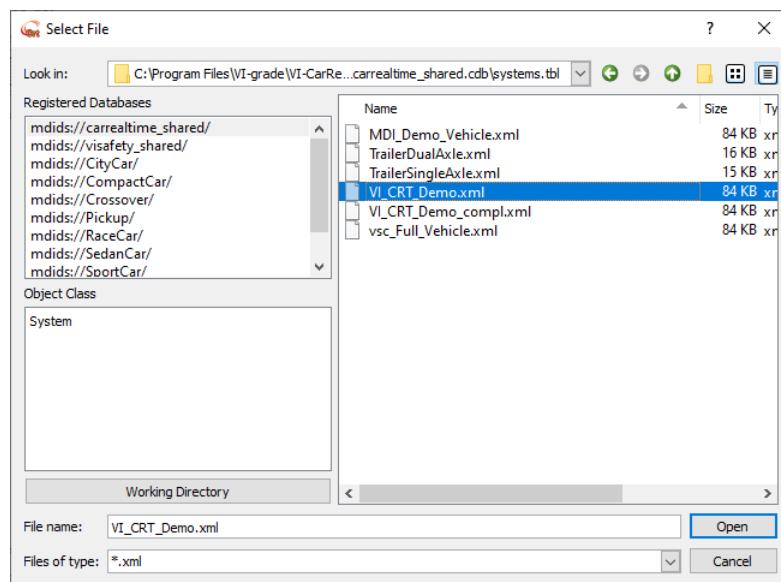
VI-CarRealTime



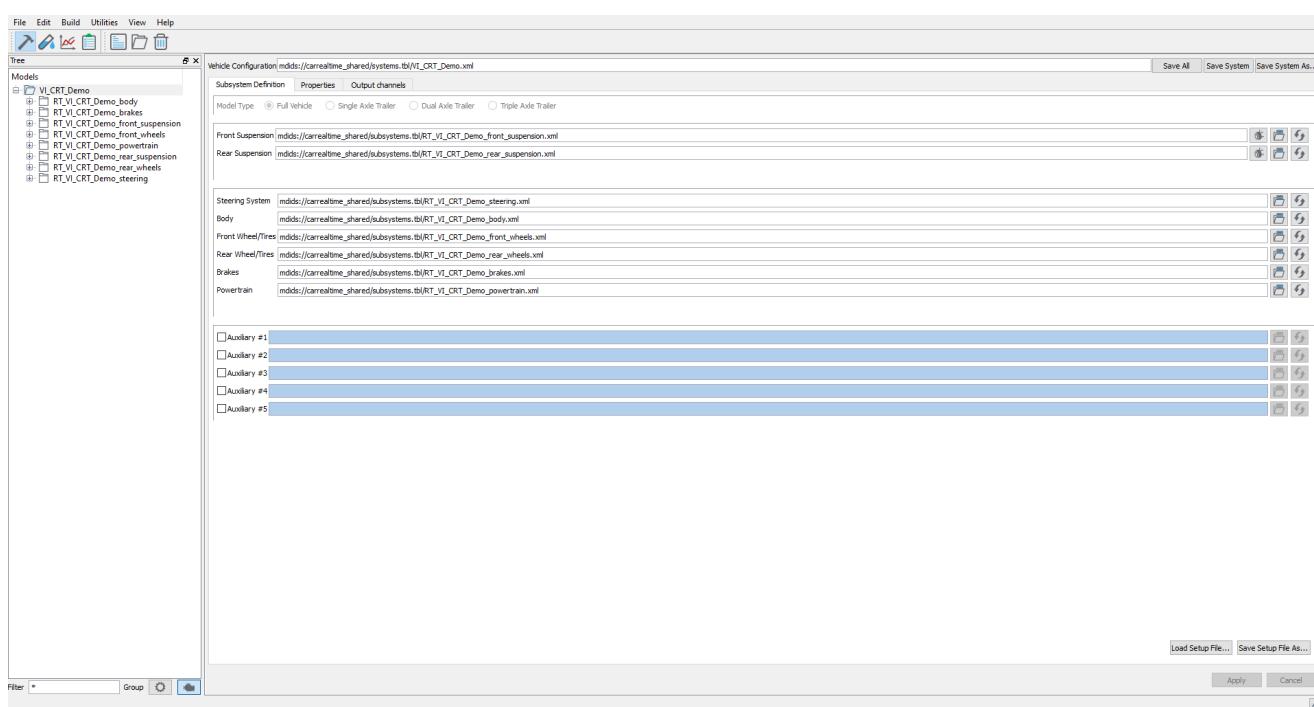
The VI-CarRealTime application is now in **Build** mode, as is shown by the highlighted hammer icon; if another mode is highlighted instead, use the simple toolbar pictured below to switch to the appropriate mode.



Open the **VI-CRT_Demo.xml** Vehicle Configuration by clicking on the icon and by locating it in the **carrealtime_shared** database.

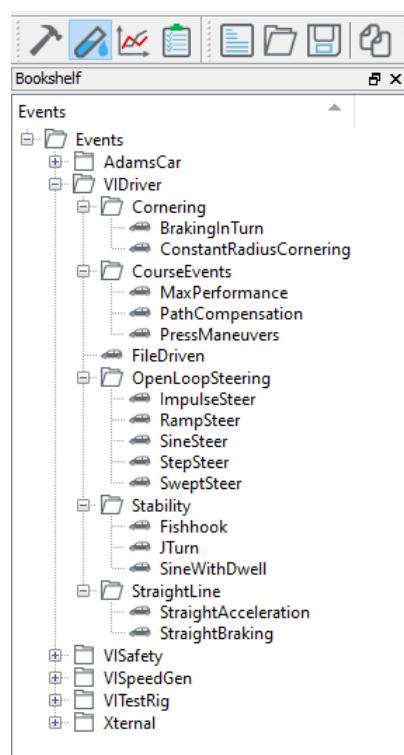


The new vehicle will now be visible in the tree view on the left side of the VI-CarRealTime application screen.

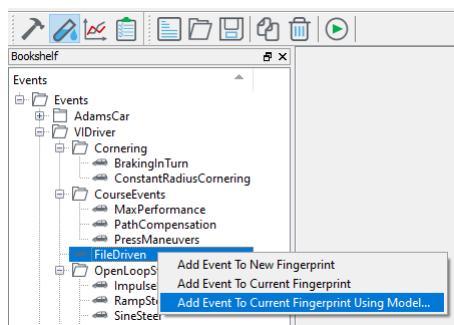


By clicking on a subsystem, it is possible to navigate and edit relevant vehicle properties.

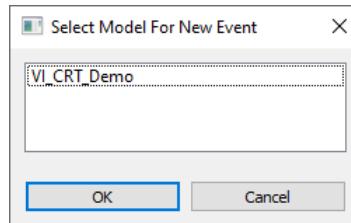
Switch to **Test Mode** using the button.



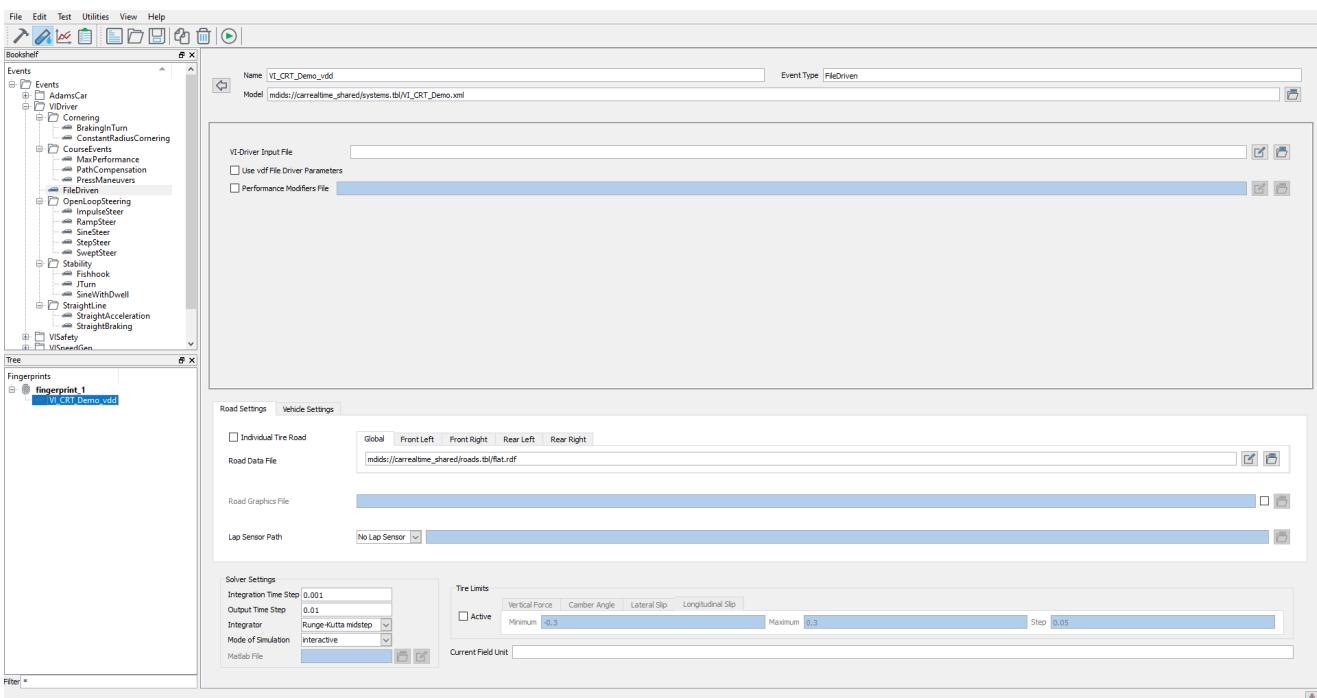
Right click on the **FileDriven** event and select: **Add Event To Current Fingerprint Using Model...**



Select the **VI_CRT_Demo** model and push the **OK** button.

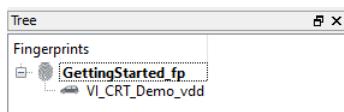


A new fingerprint, named **fingerprint_1** (by default), will be created.



A fingerprint is a placeholder file containing the reference to a particular set of events.

Right click  on the new fingerprint and rename it **GettingStarted_fp**.



You will see that the fingerprint's name is in **bold**; this means that changes made to the entity (or to any contained events) have not been saved to a file.

To save, simply click on the  button in the Test Mode Toolbar.

Now, fill the event editor with the following parameters, which are also shown in the picture below:

- **VI-Driver Input File**

`mdids://carrealtime_shared/driver_controls.tbl/chicane_R100_25.vdf` ; this file contains the definition of the relevant maneuver and can be loaded into the field by browsing the `carrealtime_shared` database (using the  button);

- **Road Data File**

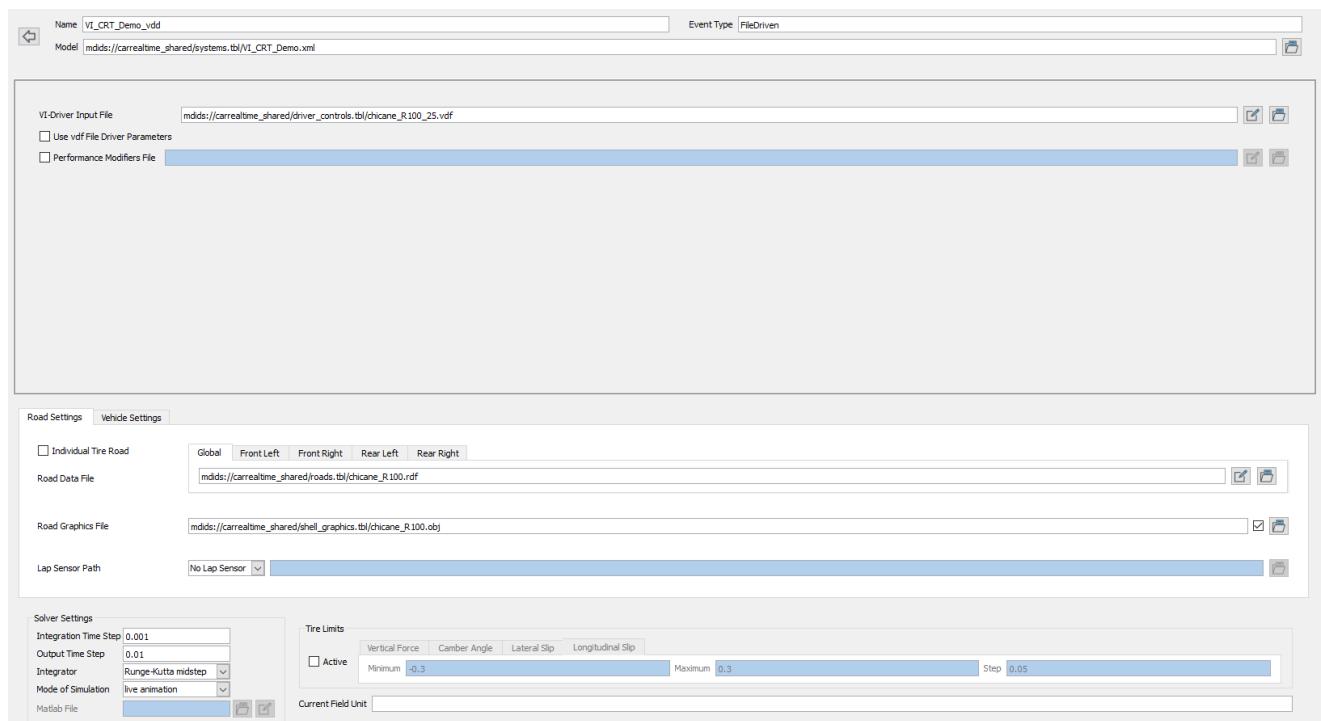
`mdids://carrealtime_shared/roads.tbl/chicane_R100.rdf`

- **Road Graphics File**

`mdids://carrealtime_shared/shell_graphics.tbl/chicane_R100.obj`

- **Mode of Simulation**

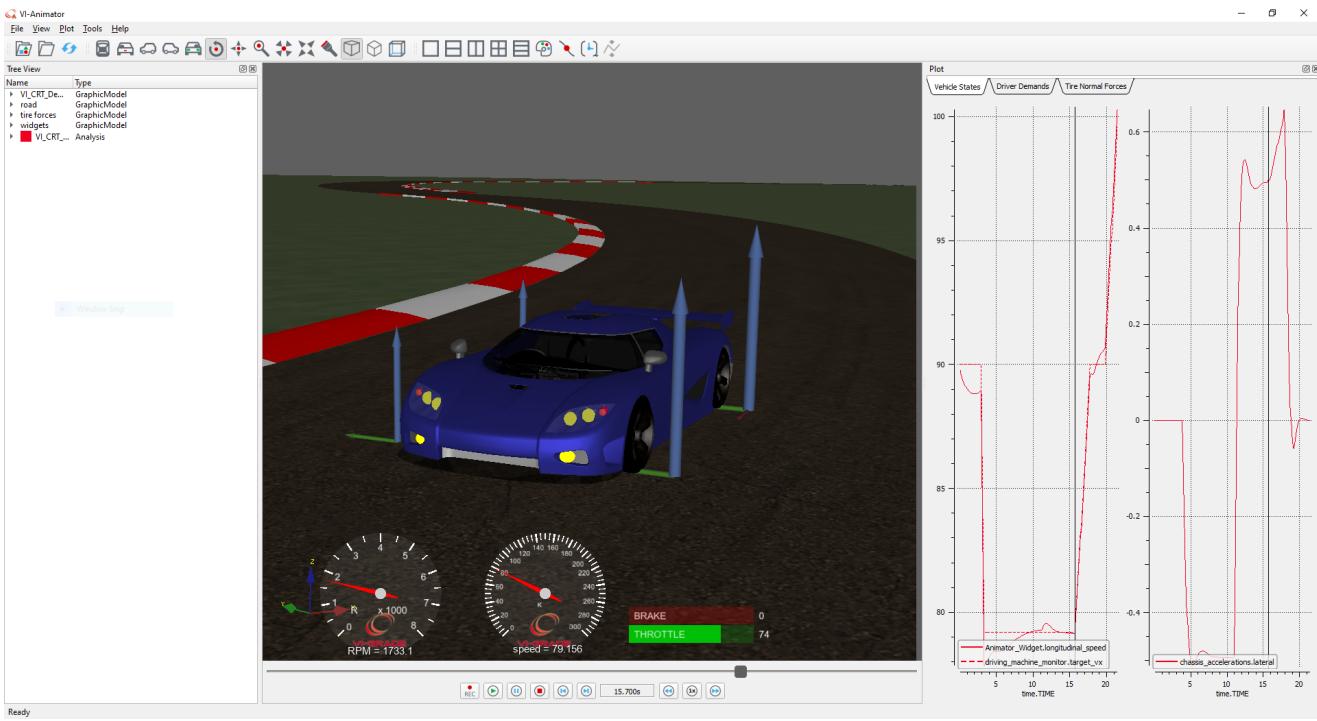
live animation ; this setting allows you to display the resulting animation while the simulation progresses.



Run the simulation by pressing the  button.

VI-Animator will start, showing the progress of our analysis.

VI-CarRealTime



At the end of simulation it is possible to replay the animation and to inspect the corresponding plots.

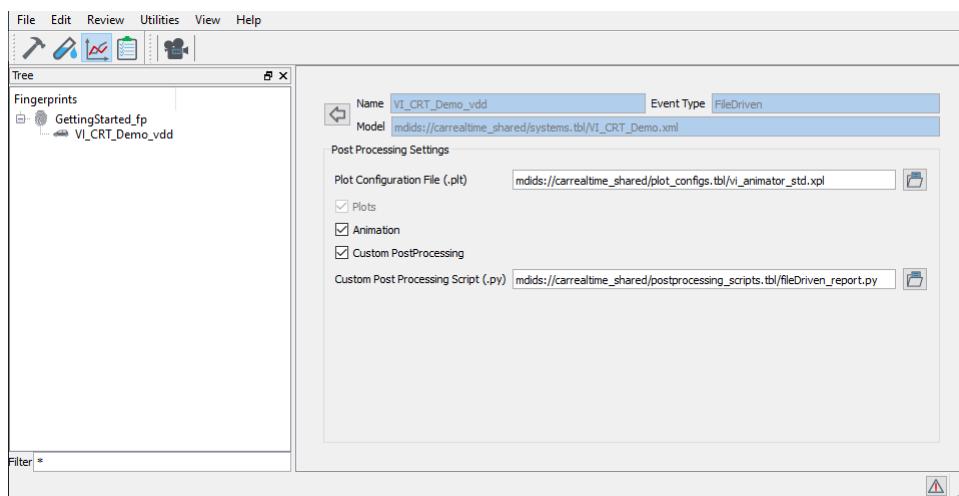
At this point, close **VI-Animator**.

Note: If the Mode of Simulation has not been set to **live animation**, post-processing of results can still be performed after the analysis is complete.

Switch to **Review Mode** using the button.

Now, select the **VI_CRT_Demo_vdd** event as shown below, click on **Custom PostProcessing** toggle button to activate the execution of custom post processing python script.

Finally, press film reel icon button:



VI-Animator will start loading the relevant results and animation.

Browsing on the default tabs within VI-Animator, it is possible to inspect:

Vehicle States:

- Target Vehicle Longitudinal Velocity, and Actual Vehicle Longitudinal Velocity, both vs. Time
- Chassis Lateral Acceleration and Side Slip Angle vs. Time

Driver Demands:

- Steering Demand vs. Time
- Brake Demand vs. Time
- Throttle Demand vs. Time
- Gear Demand vs. Time

Tire Normal Forces:

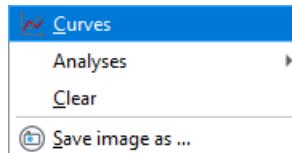
- Front Left Tire Normal Force vs. Time
- Front Right Tire Normal Force vs. Time
- Rear Left Tire Normal Force vs. Time
- Rear Right Tire Normal Force vs. Time

We will now create a new tab for the Lateral Acceleration plot.

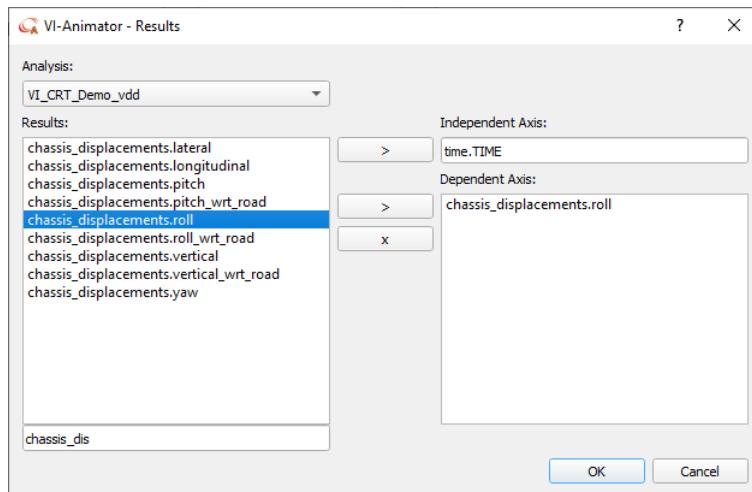
Double click on the **Tire Normal Forces** tab with the left mouse button and rename the page "**Roll Angle**" and select the layout for one plot per page.



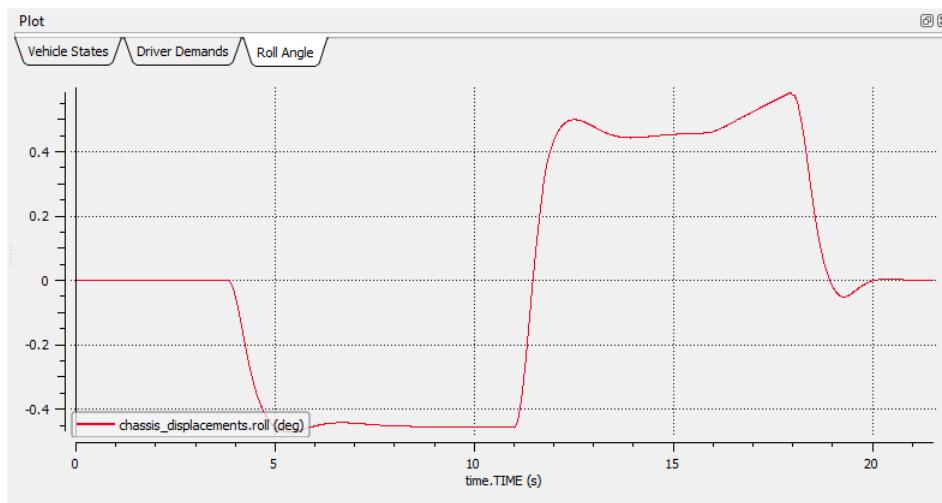
Right click anywhere on the plot area and select



A pop-up window showing the curves menu will appear; complete it as shown below and click on the **OK** button.

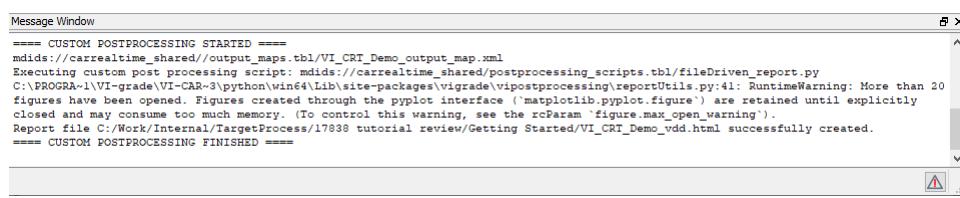


The Roll Angle of the Chassis, as a function of Time, will now be displayed, as illustrated in the following plot.



In the meantime, VI-CarRealTime executes the custom post processing python script (`mdids://carrealtime_shared/postprocessing_scripts.tbl/fileDriven_report.py`) that has been enabled in Review Mode.

A message is thrown in VI-CarRealTime Message Window instructing the user about the custom script:



In particular, the report `VI_CRT_Demo_vdd.html` file is generated in VI-CarRealTime working directory. Browse to the working directory and open it in the browser.

VI-CarRealTime FileDriven event report

Result File	C:/Work/Internal/TargetProcess/17838 tutorial review/Getting Started/VI_CRT_Demo_vdd.res
System File	mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml

Result Name	Value	Units
Laptim	21.551000000000002	sec
Longitudinal velocity maximum	100.57717369100001	Km/h
Longitudinal velocity minimum	77.83880031249997	Km/h
Steering wheel angle maximum	29.53622817700158	deg
Steering wheel angle minimum	-32.44402254422536	deg
Steering wheel torque maximum	7.0799831377	N-m
Steering wheel torque minimum	-9.28912684832	N-m
Roll angle maximum	0.5833786753167349	deg
Roll angle minimum	-0.4704062327201901	deg
Pitch angle maximum	0.5471967220607284	deg
Pitch angle minimum	-0.3648262500072214	deg
Yaw velocity maximum	0.2332561874690004	deg/dec
Yaw velocity minimum	-0.270488931740001	deg/dec
Longitudinal acceleration maximum	0.167297665231	G
Longitudinal acceleration minimum	-0.80257953906	G
Lateral acceleration maximum	0.6497642949190001	G
Lateral acceleration minimum	-0.5110974356650001	G

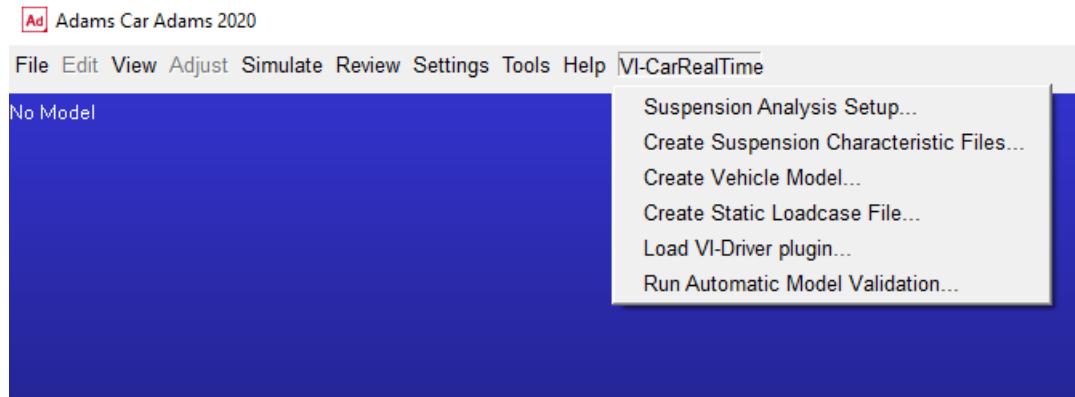
Exporting Vehicle From Adams Car

The tutorial demonstrates how to export a vehicle model to VI-CarRealTime from Adams Car.

Complete the following steps to load the Adams Car plugin:

- Start Adams Car from your working directory
- Load the VI-CarRealTime plug-in
- From the Main Menu, go to *Tools -> Plug-in Manager -> VI-CarRealTime*

Now, the Adams Car Main Menu should look like this:

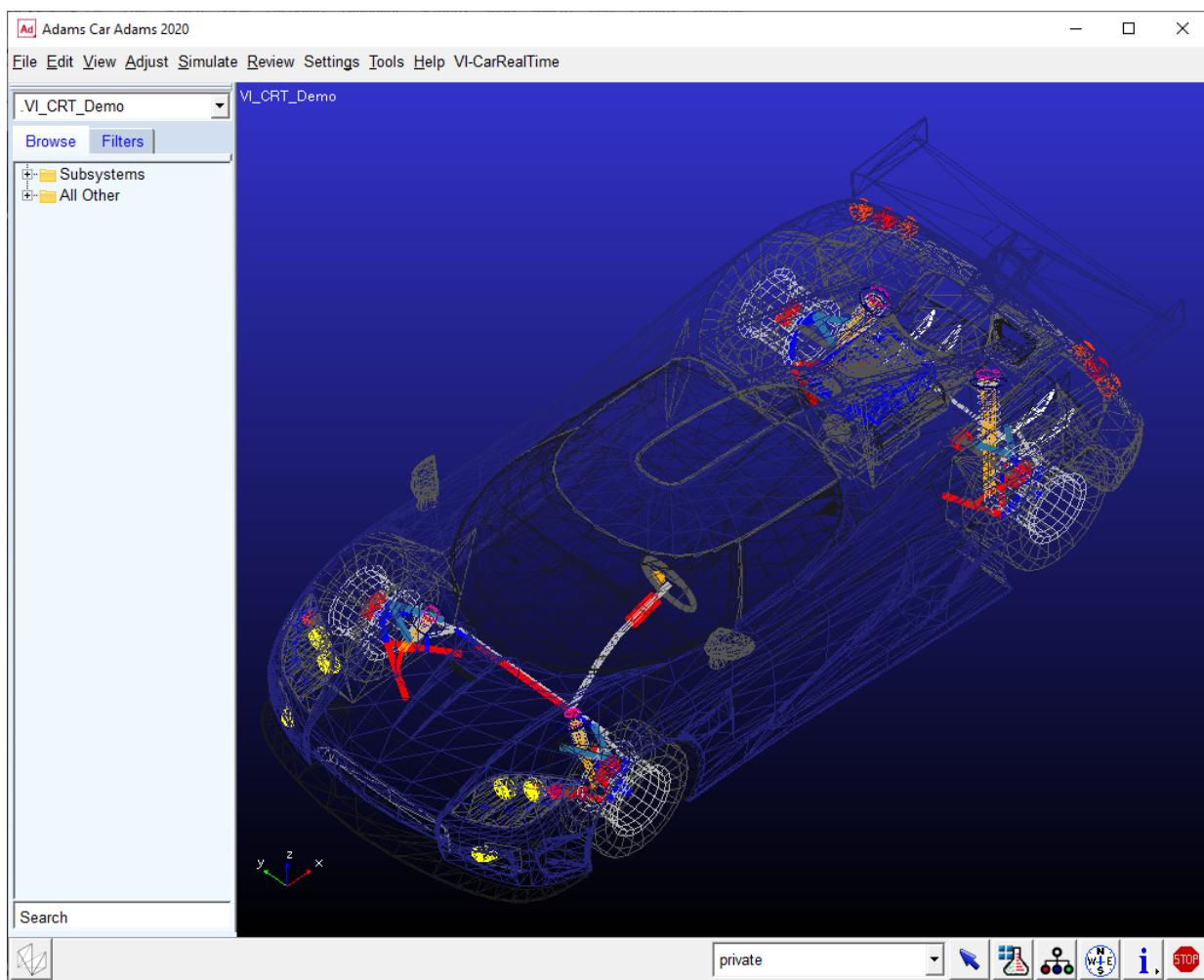


The VI-CarRealTime shared database, called *carrealtime_shared*, has been added to the Adams Car session.

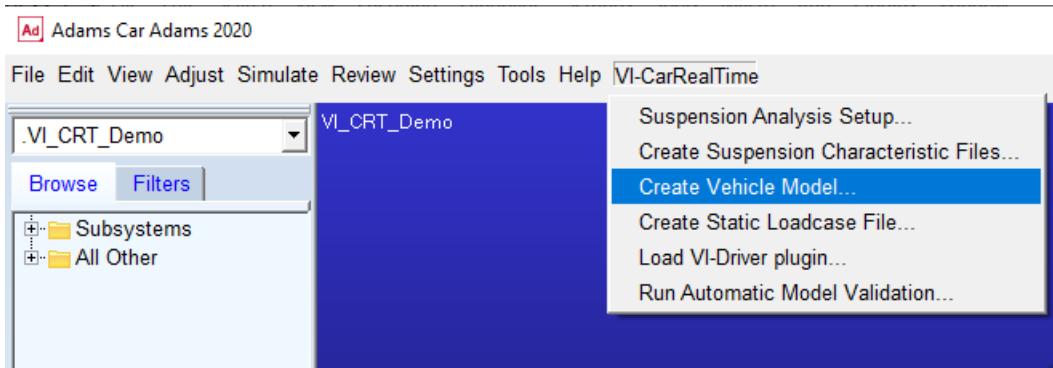
To load an assembly, follow these steps:

- click on **File**;
- click on **Open Assembly**;
- right mouse click inside the **Assembly Name** field;
- click on **Search**;
- select <**carrealtime_shared**>/assemblies.tbl;
- the **Select File** Dialog Box will appear;
- double click on the file **VI_CRT_Demo.asy**.

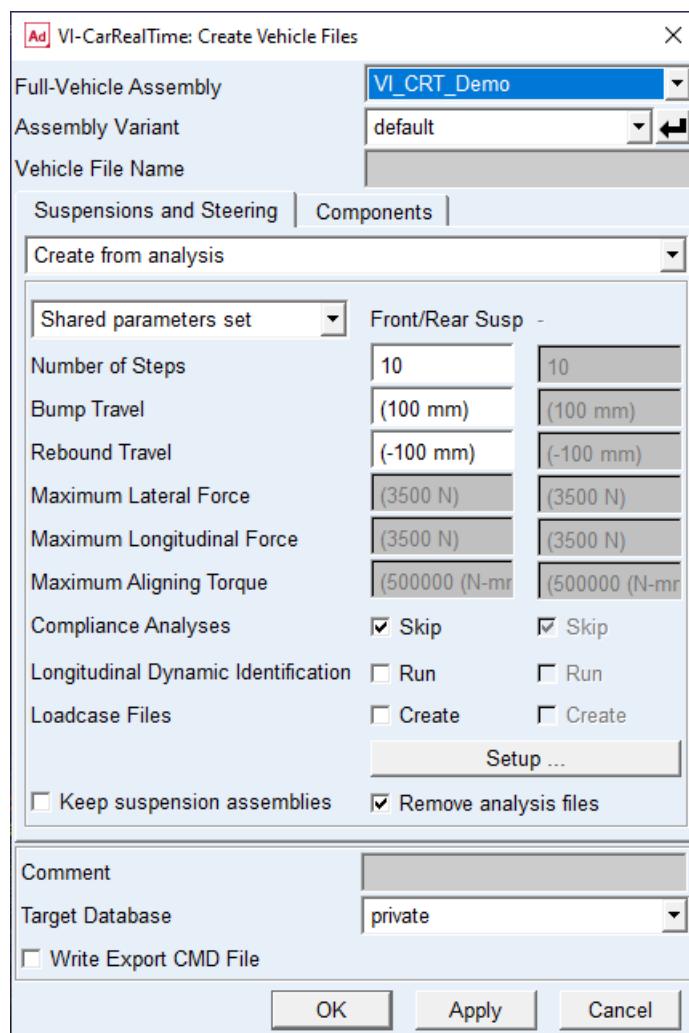
After loading the assembly you can access the VI-CarRealTime menu to start the parameter extraction.



Select the **Create Vehicle Model** menu entry:



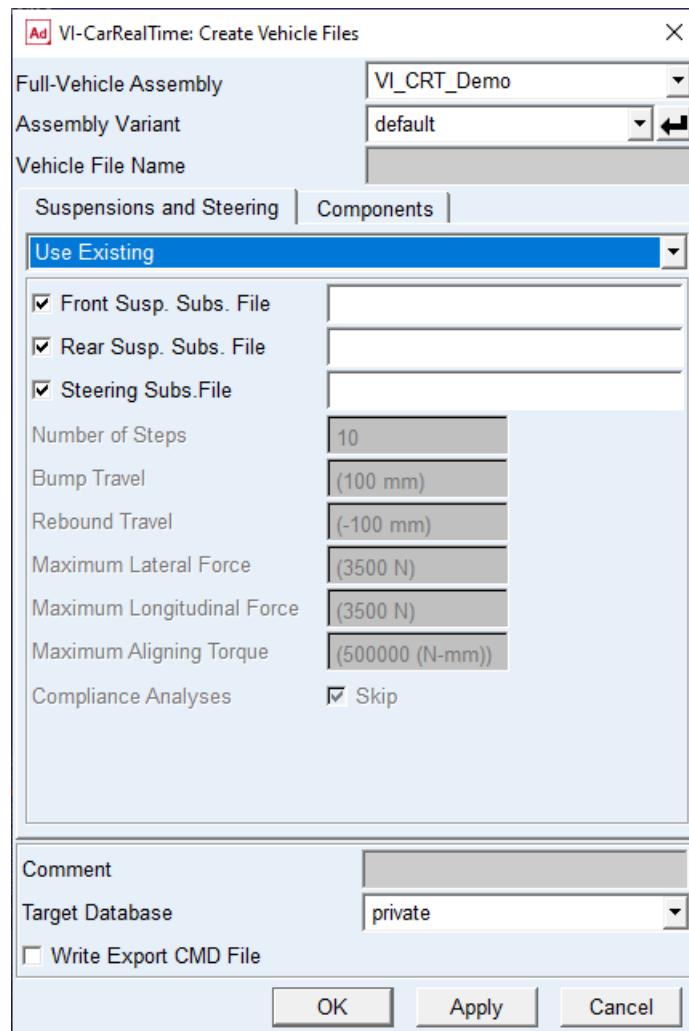
The following window will show up:



The VI-CarRealTime vehicle model is created using a procedure which gathers model data from:

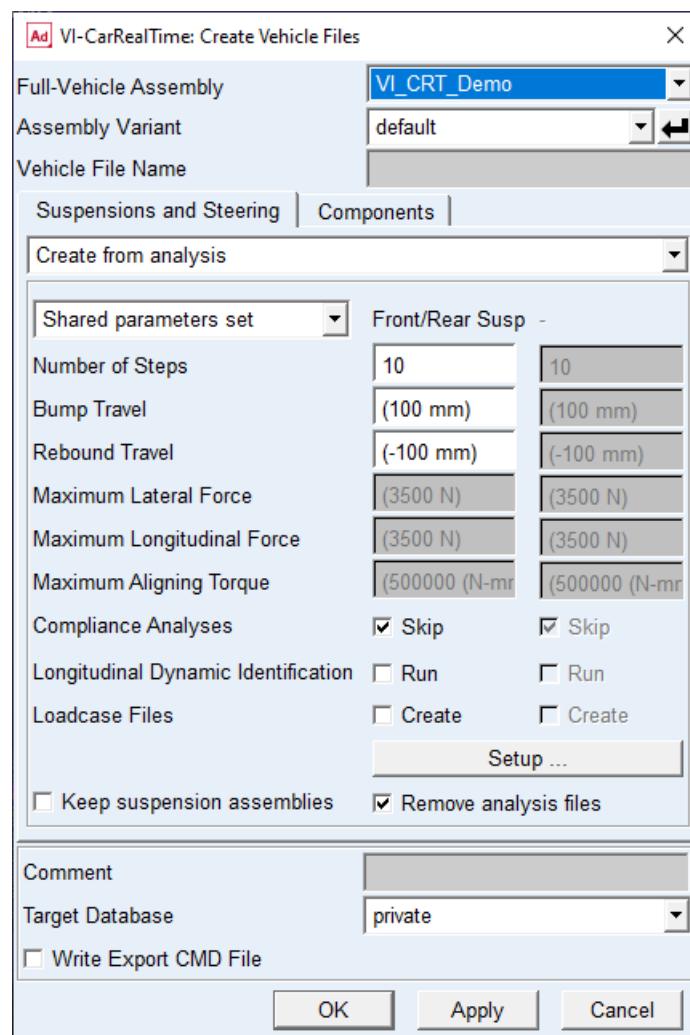
1. A set of analyses for suspensions and steering systems;
2. Direct look-up into vehicle assembly (tires, body, powertrain subsystems, etc.);
3. Full vehicle static analysis, in order to compute sprung and unsprung mass distribution.

It is also possible to skip the first aforementioned step by selecting **existing suspension and steering subsystems**, as shown below:



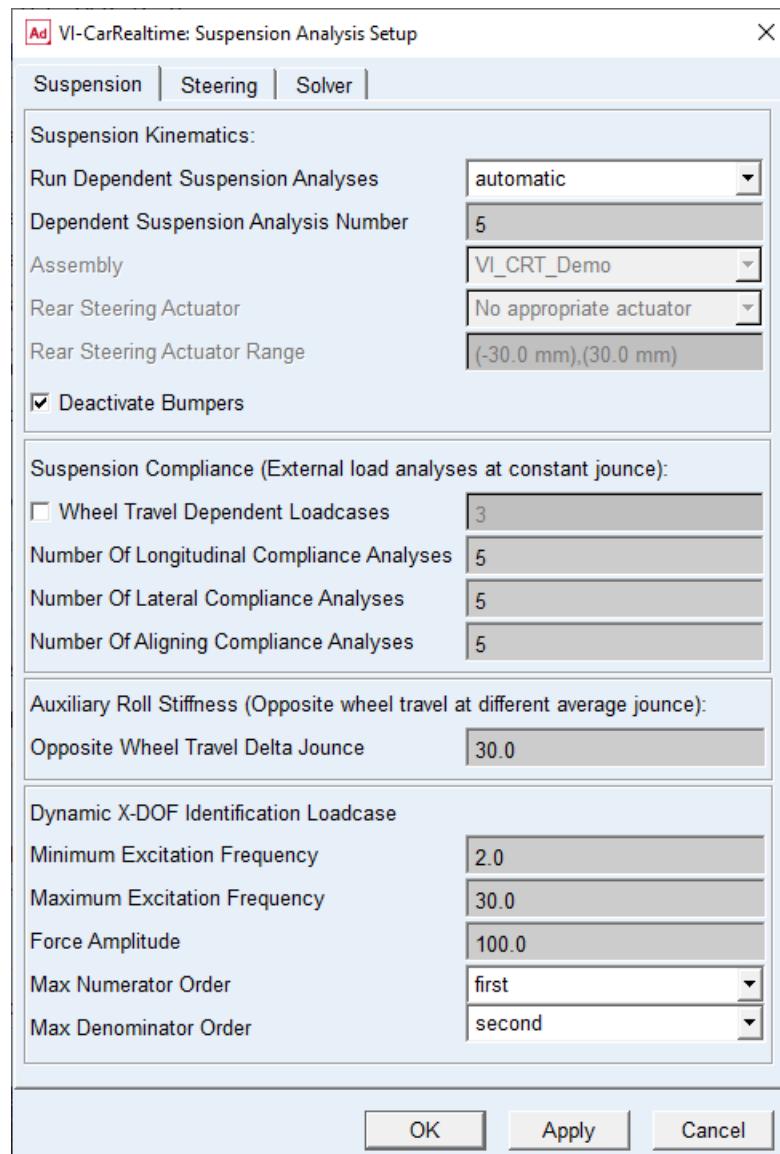
When choosing to perform **suspension analyses** it is possible to customize the detail of data to be generated by:

- Selecting a **shared** or **independent** set of analysis parameters (bounce and number of steps) for front and rear suspension.
- **Skipping compliance** evaluation analyses (for longitudinal, lateral, and aligning effects on suspension angles).
- Defining a **Target Database** used to store the generated VI-CarRealTime model.
- Creating **Loadcase Files**, containing wheel force data applied during suspension parameter extraction analyses. These can be used in VI-CarRealTime with the **Suspension Testrig** event for validation purposes.

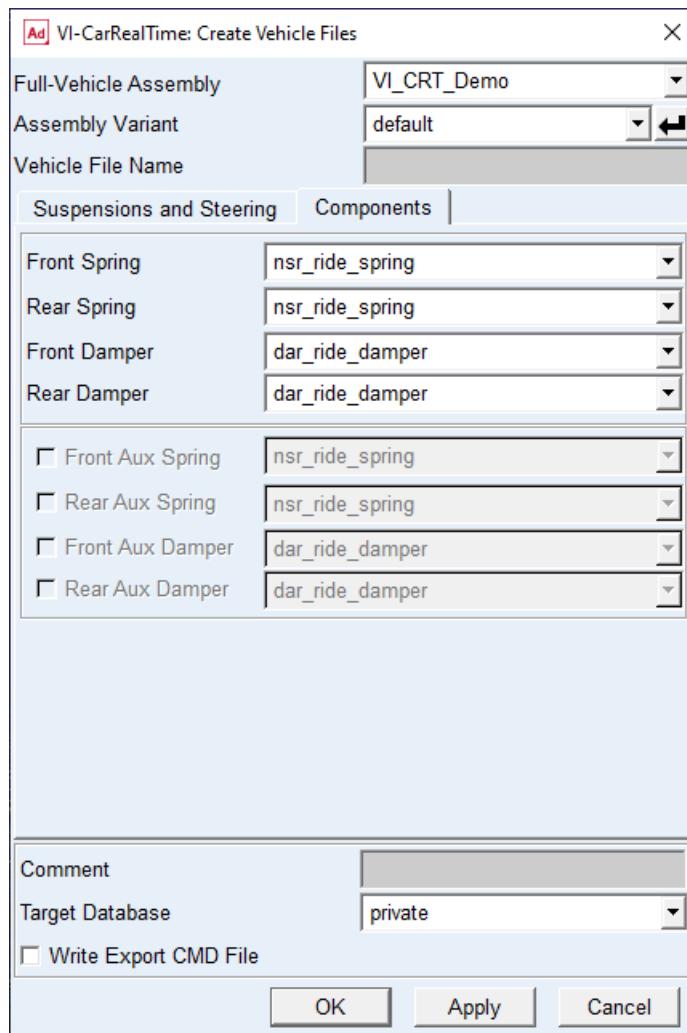


Since some of the suspension curves are defined using 3D splines, it is also possible to control the level of detail by setting the number of columns to be used for the second independent variable:

- Paired wheel jounce for dependent suspensions (rear);
- Wheel jounce definition for compliance analyses;
- Average jounce definition for auxiliary roll stiffness (ARB);
- Delta jounce for auxiliary roll stiffness (ARB);
- Jounce definition for steering analyses;
- Maximum steering wheel angle.



In a particular case where the suspension has multiple components for springs and dampers, a user can choose which set of components will be used in the VI-CarRealTime model (only one spring and damper pair for the suspension is supported). The contribution from additional spring components will be stored in the auxiliary vertical force data, whereas additional dampers will be neglected.



Parameter extraction includes the following tasks:

1. Front suspension curves extraction
2. Steering system curves extraction
3. Rear suspension curves extraction
4. Body subsystem data extraction and mass evaluation event
5. Other subsystems' (power-train, brakes, wheels) data extraction

For suspension curve extraction, the steps are:

1. Creation of suspension assembly from a full vehicle
2. Parallel wheel travel analysis for suspension kinematics (track, base, toe, camber, side view angle vs. suspension jounce, components motion ratios)
3. Set of opposite wheel travel analyses for Anti Roll Bar characteristic computation
4. Set of external load analyses under varying external force (FX, FY, TZ) at different fixed wheel travel settings, for compliance effects evaluation

For steering system curve extraction, the steps are:

1. Steering wheel sweeps at fixed wheel jounce for kinematics
2. External load analysis (TZ) to compute steering system compliance

VI-CarRealTime

During the mass evaluation event, a static analysis on the full vehicle is performed. This procedure is used to calculate sprung and unsprung mass values along with longitudinal and vertical position of sprung mass CG. The other masses of a VI-CarRealTime model are concentrated at wheel centers.

At the end of the extraction procedure, the model is stored in the selected target database.

The VI-CarRealTime GUI can be launched from the Windows Start Menu, or from a DOS (cmd) shell using the **vicrt20** command.

Running Analyses

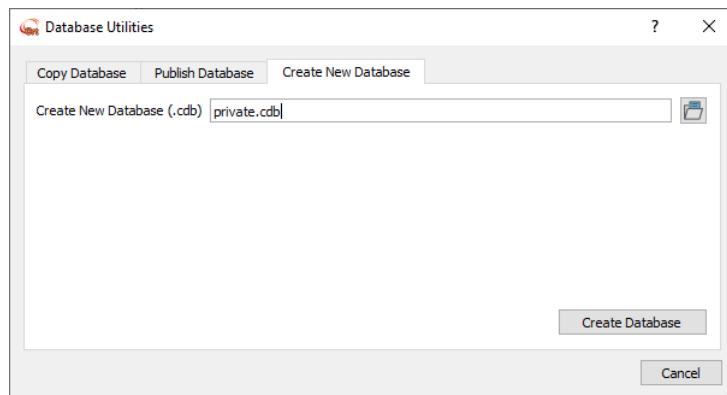
The following tutorial shows how to run analyses run in VI-CarRealTime and how to benefit from their use, in a vehicle dynamics analysis perspective.

The effect of vehicle parameters changes on performance will be investigated, using the **VI_CRT_Demo** model contained in the *carrealtime_shared* database as a reference.

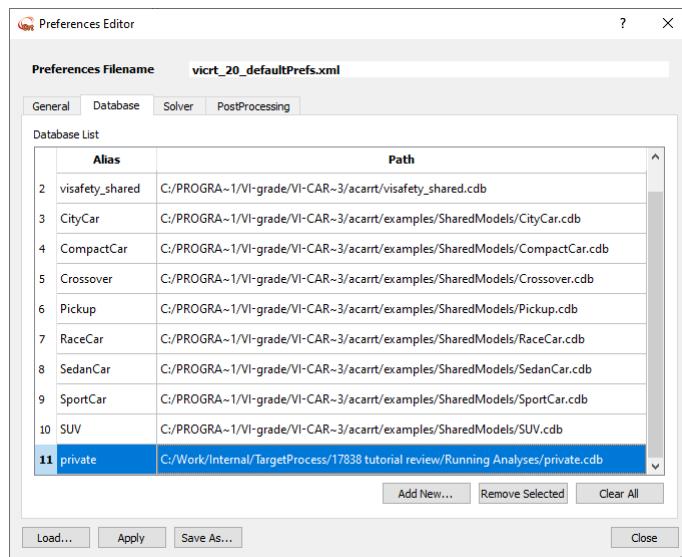
The first action will be to create a vehicle model clone, in order to safely apply the required changes to the model (without altering default parameters in the model within our shared database).

Note: If you have already completed the [Exporting Vehicle From Adams Car](#), you may skip the following steps; simply use the existing **VI_CRT_Demo_v1** model found in your pre-existing **private** database.

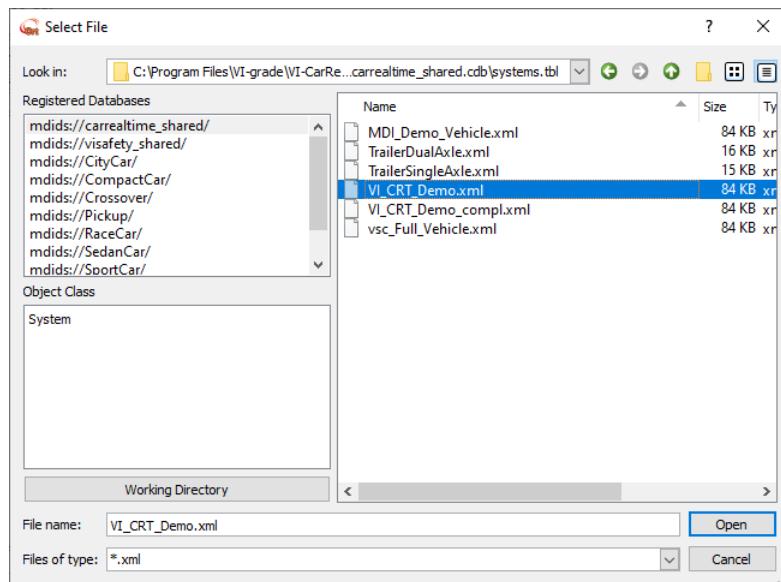
1. **Create a new database:** from the VI-CarRealTime Main Menu, select *Utilities-> Database Utilities* and create a new database in your current working directory called **private.cdb**. To do this, fill in the corresponding field as shown in the picture below and then press the **Create Database** button.



2. **Register the database:** From Main Menu, select *Edit->Preferences* and verify that the newly created database has indeed been added to the current session.

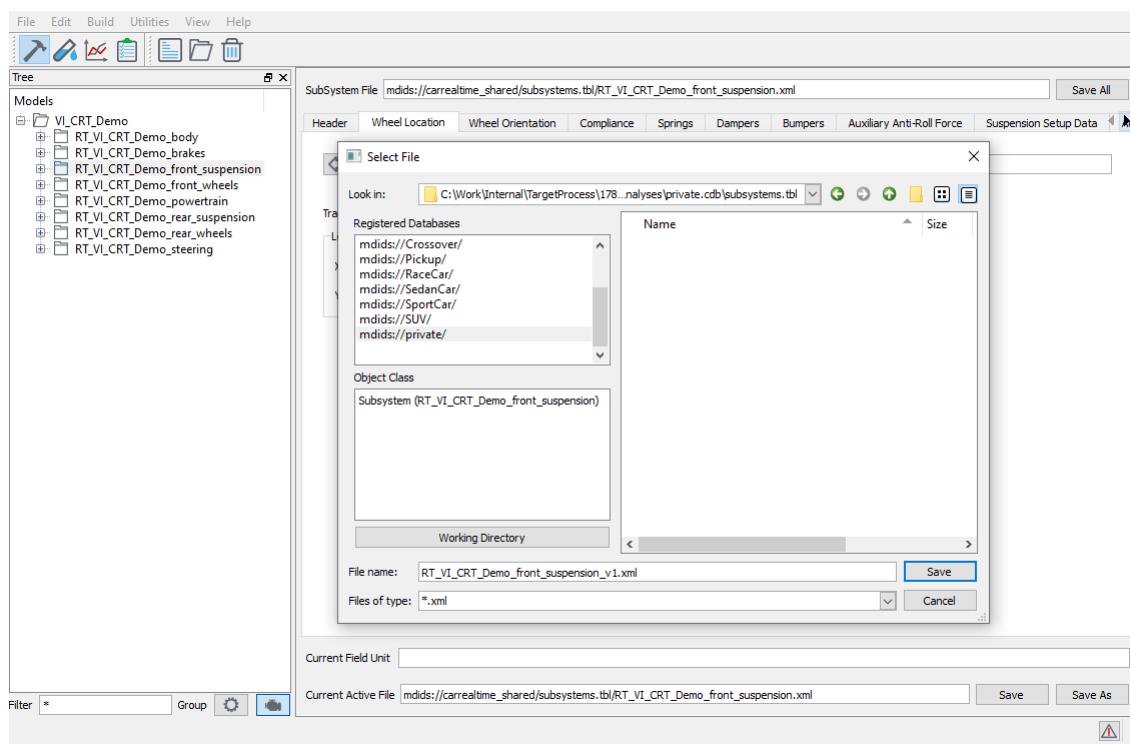


3. **Load the VI_CRT_Demo model:** In Build Mode, open **VI-CRT_Demo.xml** by clicking on the icon, and browse to the **carrealtime_shared** database.

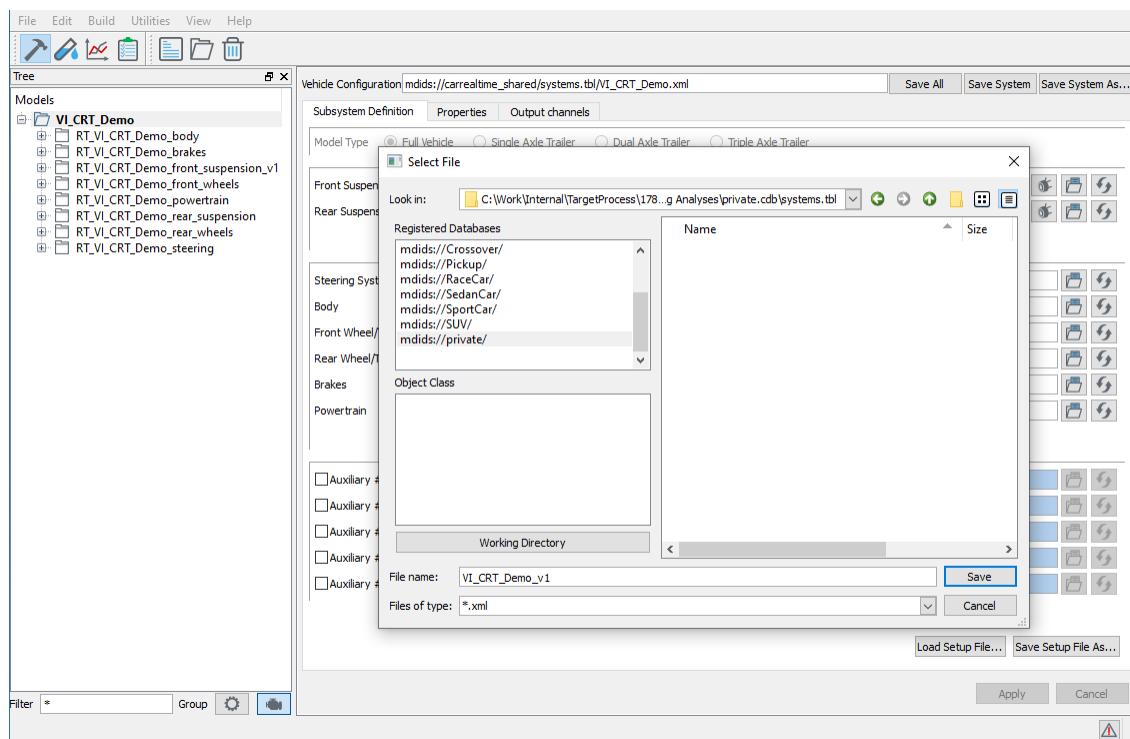


4. **Clone the Front Suspension:** In Build Mode's tree view, click on the front suspension subsystem (**RT_VI_CRT_Demo_front_suspension**) and save the file to your private database using the **Save As** button in the bottom right part of the window (save it as **RT_VI_CRT_Demo_front_suspension_v1.xml** to distinguish it from the original); click on **YES** in the resulting dialog box asking to reference the saved data in the subsystem file, so that the system model now points to the new subsystem's xml file.

VI-CarRealTime

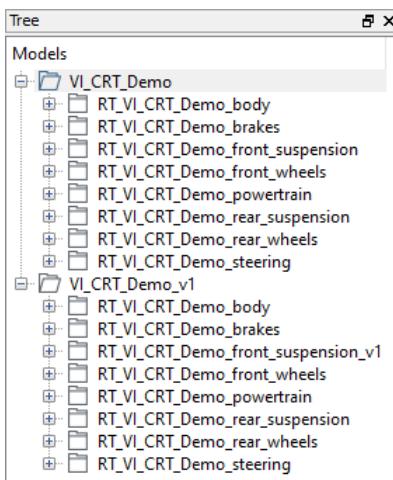


- Clone System:** In the **Build Mode** tree view, click on the **VI_CRT_Demo** model; this will show the **System Editor**, which allows you to edit the subsystem list and global vehicle properties. You will notice that the front suspension subsystem is pointing to the newly created file in **private.cdb** database, and that the system name in the tree view is in bold, since there are changes that have not yet been saved. Using the **Save System As** button, save the modified system into your private database, naming it **VI_CRT_Demo_v1**.



You will now have **VI_CRT_Demo_v1** vehicle model in your private database (effectively a copy of the **VI_CRT_Demo** model, referring to the same subsystems, except for the Front Suspension, on which some changes will be applied).

Open **VI_CRT_Demo** from the shared database once again, in order to have both models in the same session.

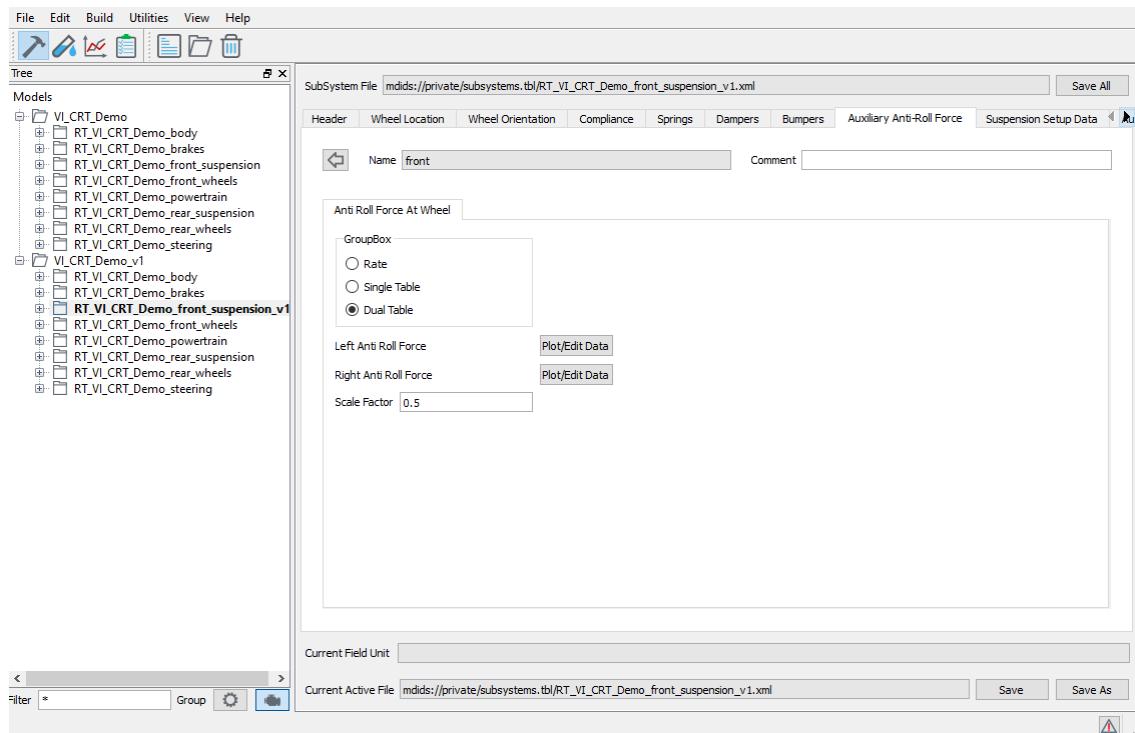


We will now investigate the change that a front anti roll bar rate exerts on vehicle handling.

In the **Build** mode tree view, click on the **RT_VI_CRT_Demo_front_suspension_v1** subsystem and then navigate to the **Auxiliary Anti Roll Force** tab.

We will now cut the ARB stiffness in half, in order to simulate a softer anti roll bar.

We can do this by easily by setting the **Scale Factor** parameter to **0.5**. Then, save the subsystem file by clicking on **Save All**.



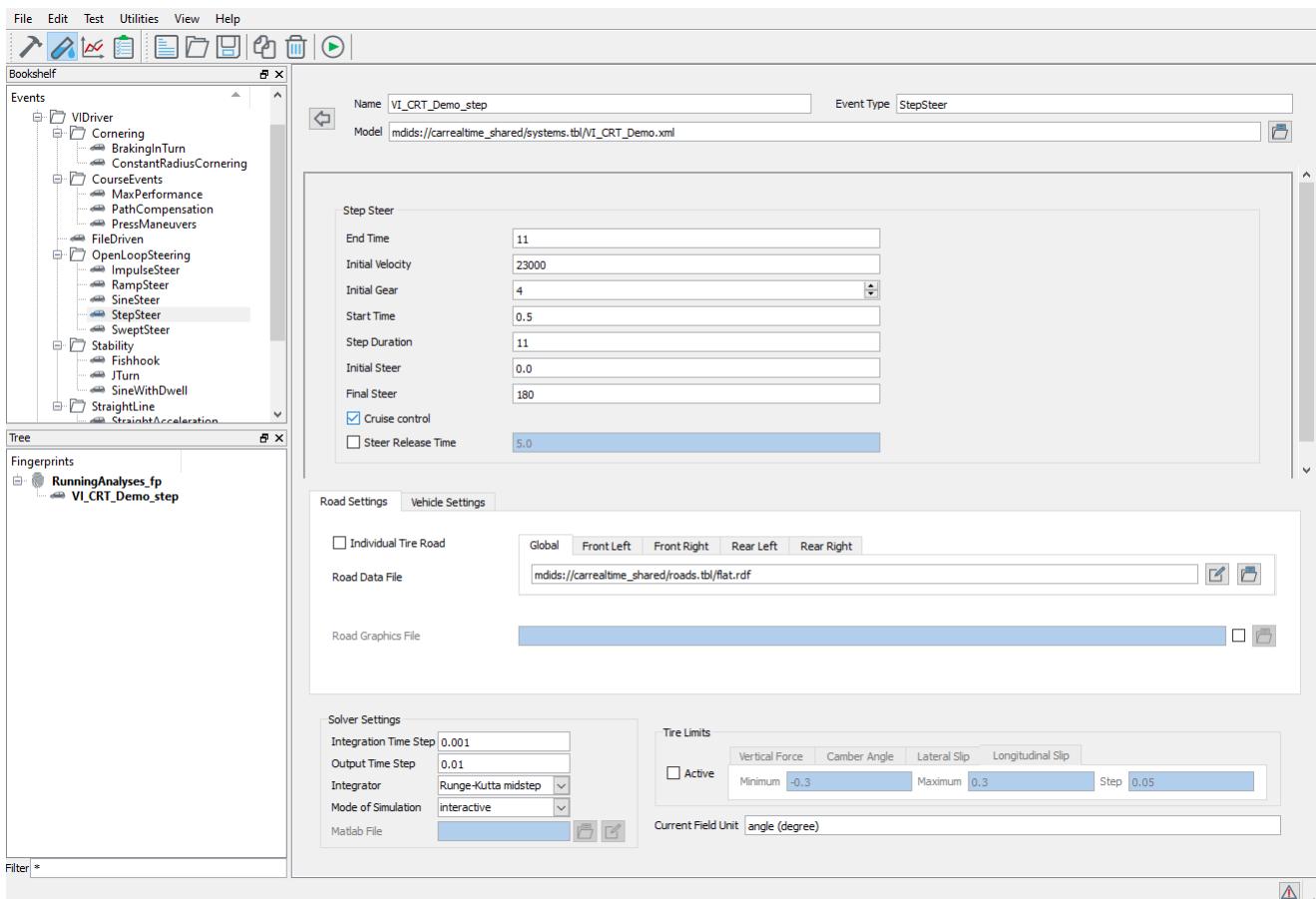
We will now analyze the effect of the change in anti roll bar stiffness, by comparing the edited ARB stiffness with that of our base model.

Switch to **Test Mode** using the  button.

Create a new fingerprint named **RunningAnalyses_fp**

By browsing *Events -> VI-Driver -> Open Loop Steering* in the tree view, add a new **StepSteer** event named **VI_CRT_Demo_step** to the fingerprint, using the **VI_CRT_Demo.xml** model.

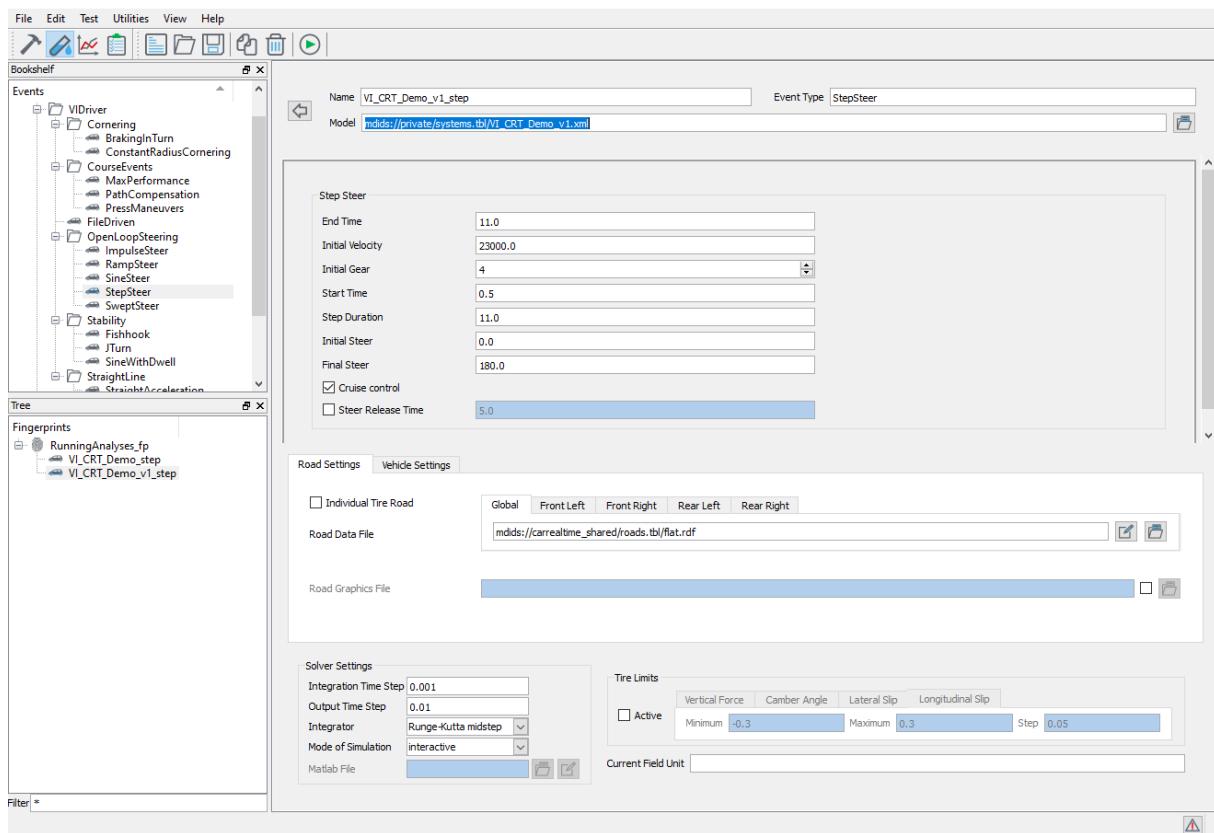
Set the event parameters as shown below.



Right click on the  **VI_CRT_Demo_step** event and select **Copy Selected Event**.

Rename the newly created event **VI_CRT_Demo_v1_step** and change the Model field in the event editor to **mdids://private/systems.tbl/VI_CRT_Demo_v1.xml**, as shown below.

Save the fingerprint.



It is now possible to run both of our created events by selecting our fingerprint in the Test mode tree view, and by clicking the button.

```
VI.PTW (VI-CarRealTime Python Task Window)
Performing dynamic simulation...
Simulation Progress (percent complete):
0 50 100
=====
= VI-CarRealTime =
= VEHICLE SIMULATION STATISTICS =
=====
--- Simulation Statistics ---
>> LAP TIME = 11.000000 sec
>> LAT ACCEL MAX = 0.000002 G
>> LAT ACCEL MIN = -1.151676 G
>> LON ACCEL MAX = 0.220417 G
>> LON ACCEL MIN = -0.057039 G
>> LON SPEED MAX = 82.799464 km/h
>> LON SPEED MIN = 30.692590 km/h
>> LON SPEED AVG = 77.621903 km/h
>> STEERING MAX = 3.122035 rad
>> STEERING MIN = 0.000000 rad

Writing output to file. Please Wait ...

Elapsed Simulation Time: 11.000 (sec).
Elapsed Clock Time: 1.635 (sec).
Computational Efficiency: 6.728 (sim. sec)/(sec).

Simulation is complete.
```

Switch to Review mode using the button once the simulation is complete.

VI-CarRealTime

Now select the **RunningAnalyses_fp** fingerprint as shown below, using the left mouse button, and then

click on the film reel button that looks like this:



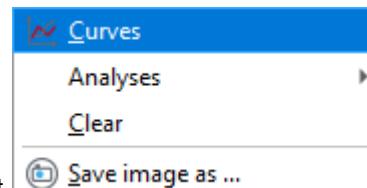
VI-Animator will start and load the desired simulation results and animations.

The events that just finished running are dynamic open loop step steering analyses, performed at a constant velocity and with low steering wheel angle variation, in order to evaluate the steady state understeering properties of the vehicle; one could expect that by having a stiffer front anti roll bar, the vehicle would exhibit a more pronounced understeering behavior.

We can assess this behavior by creating the **understeer gradient** plot (which translates to Lateral Acceleration vs Steering Wheel Angle) in VI-Animator.

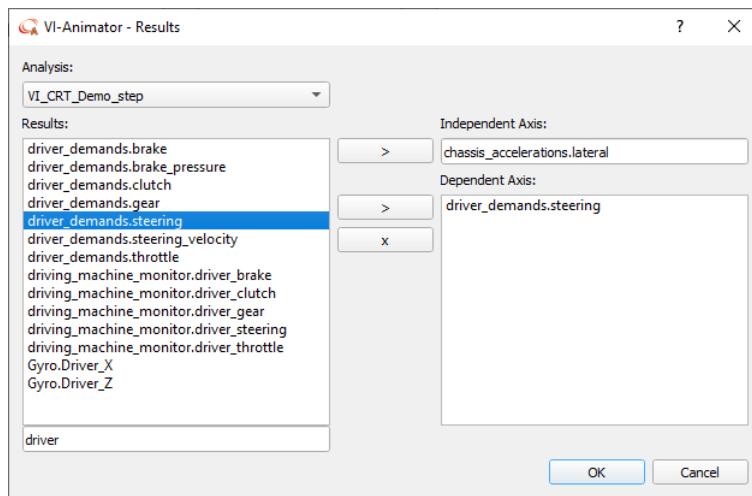


Click on **Tire Normal Forces** tab with the right mouse button and select **Insert After**; rename the page "**Understeer Gradient**" by double clicking on the page label.

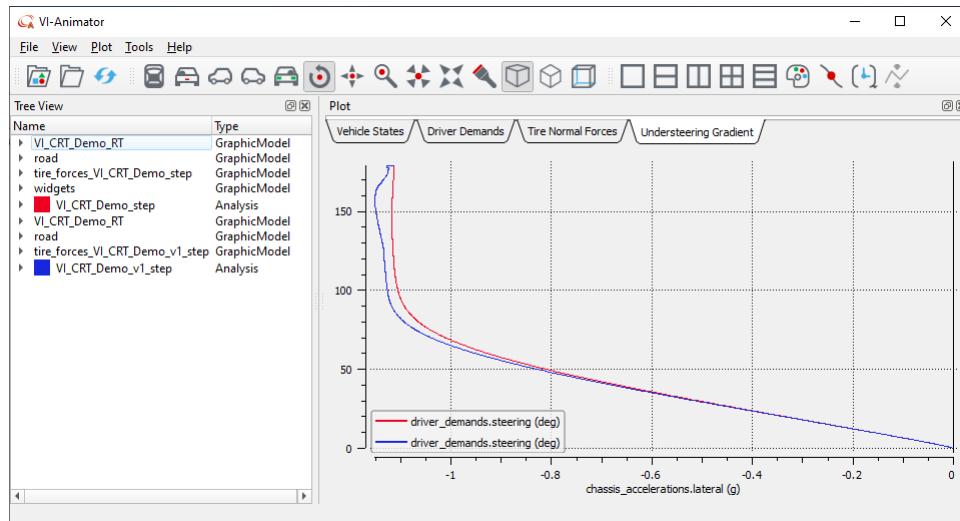


Now, right click anywhere on the plot area and select

A pop-up window will appear; fill out the axes as shown below and click OK. Please note that the leftmost text field in the curves editor window can be used to filter Results to quickly find relevant variables.



The following plot will subsequently be created:



As expected, the base vehicle shows more understeering behavior (the steering angle required to accomplish a given lateral acceleration is larger), and additionally, the steady state lateral acceleration limit is noticeably higher for the softer ARB (anti roll bar) vehicle.

VI-Animator may now be closed for the time being.

For the next exercise, the handling performance in a more generic dynamic event will be analyzed.

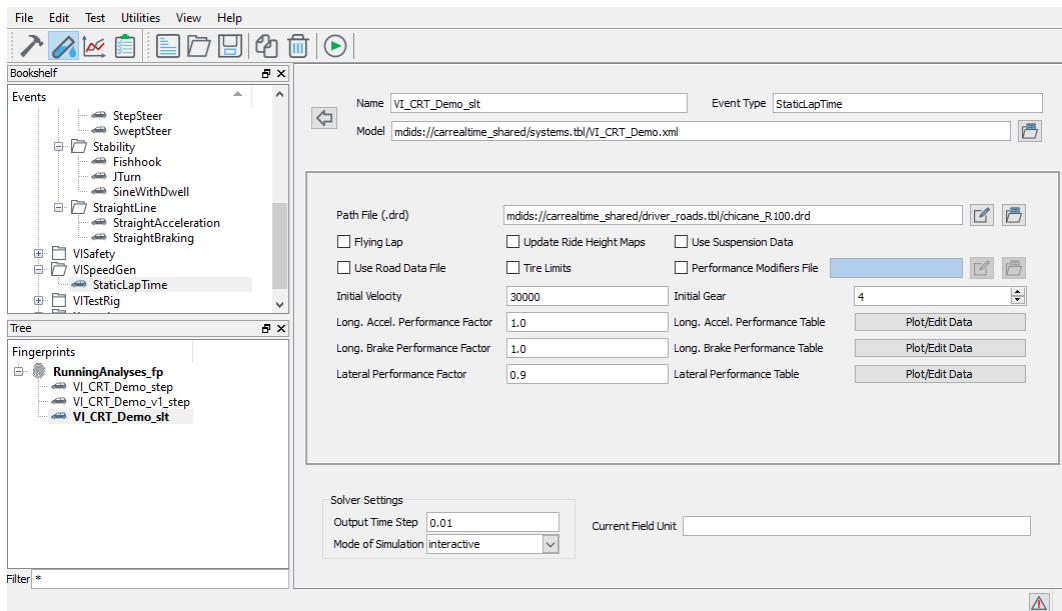
First, we will run a **SpeedGen** analysis on a chicane course which contains two turns, each with a radius of 100m. The velocity profile created by the quasi-static solver will then be fed into a **FileDriven** event for dynamic analysis.

Switch to **Test Mode** and create a new **StaticLapTime** event for the **VI_CRT_Demo** vehicle, under the **RunningAnalyses_fp** fingerprint, using the following parameters:

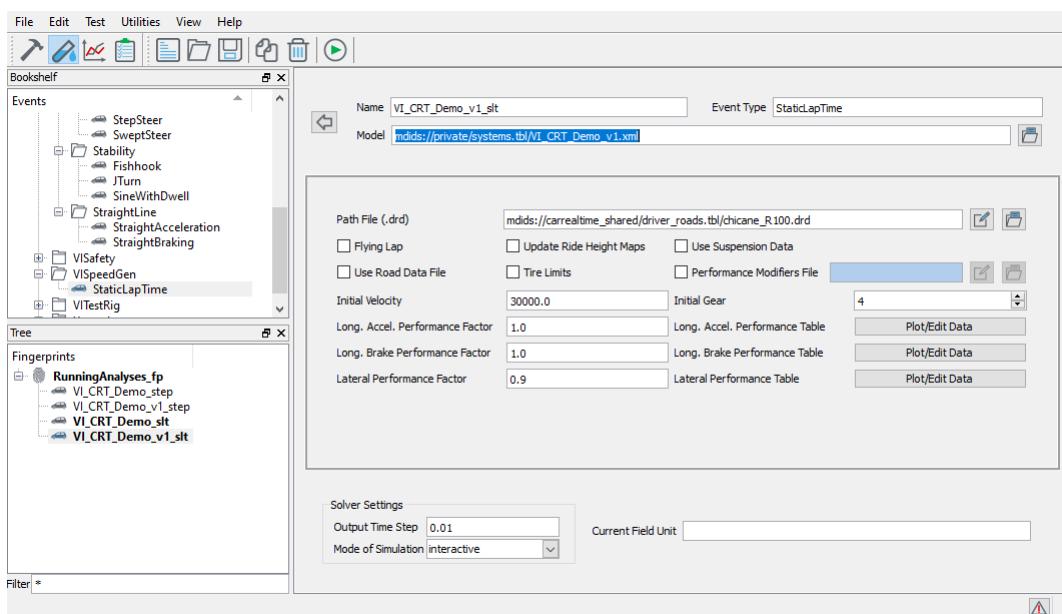
- **Model:** mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml
- **Path File (.drd):** mdids://carrealtime_shared/driver_roads.tbl/chicane_R100.drd
- **Initial Velocity:** 30000
- **Initial Gear:** 4

VI-CarRealTime

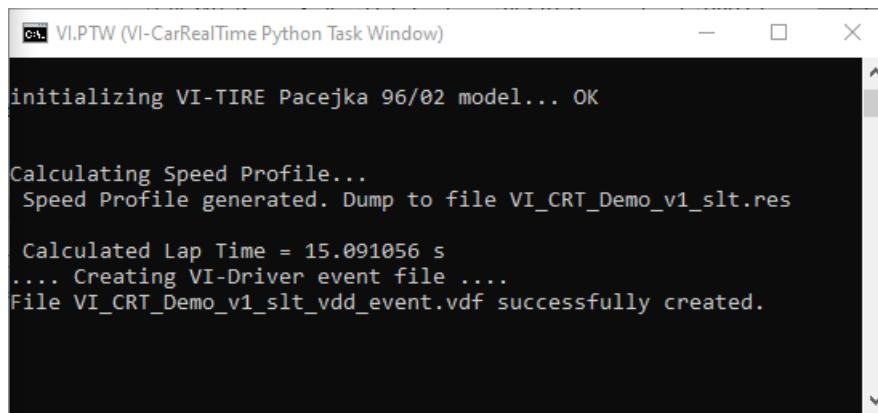
- Long. Accel. Performance Factor: 1
- Long. Brake Performance Factor: 1
- Lateral Performance Factor: 0.9



Copy the **VI_CRT_Demo_slt** event, this time using the **VI_CRT_Demo_v1** model, and set up the parameters as pictured below :



Once your **VI_CRT_Demo_v1_slt** event matches the picture above, run the two freshly created events by clicking on the button after selecting both of them.



```

VI.PTW (VI-CarRealTime Python Task Window)

initializing VI-TIRE Pacejka 96/02 model... OK

Calculating Speed Profile...
Speed Profile generated. Dump to file VI_CRT_Demo_v1_slt.res

Calculated Lap Time = 15.091056 s
.... Creating VI-Driver event file ....
File VI_CRT_Demo_v1_slt_vdd_event.vdf successfully created.

```

Both simulations, which will run in a DOS (cmd) shell as pictured above, show the same Lap Time. The reason they match is because the VI-SpeedGen underlying vehicle model is simplified; it does not capture all suspension elastokinematics data. In fact, by comparing both generated vdf (event) files side by side, it can be seen that they are identical (other than their name).

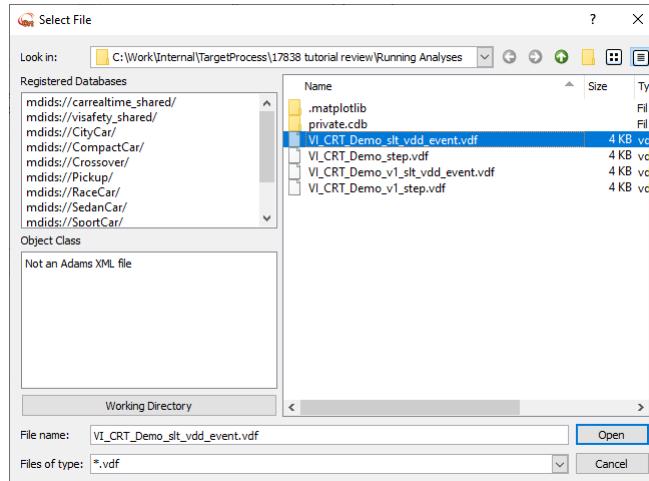
This time around, in order to properly assess the differences which were ignored in the previous simplified simulation, dynamic simulation must be performed.

To begin, create a new **FileDriven** event under the **RunningAnalyses_fp** fingerprint, using the following parameters:

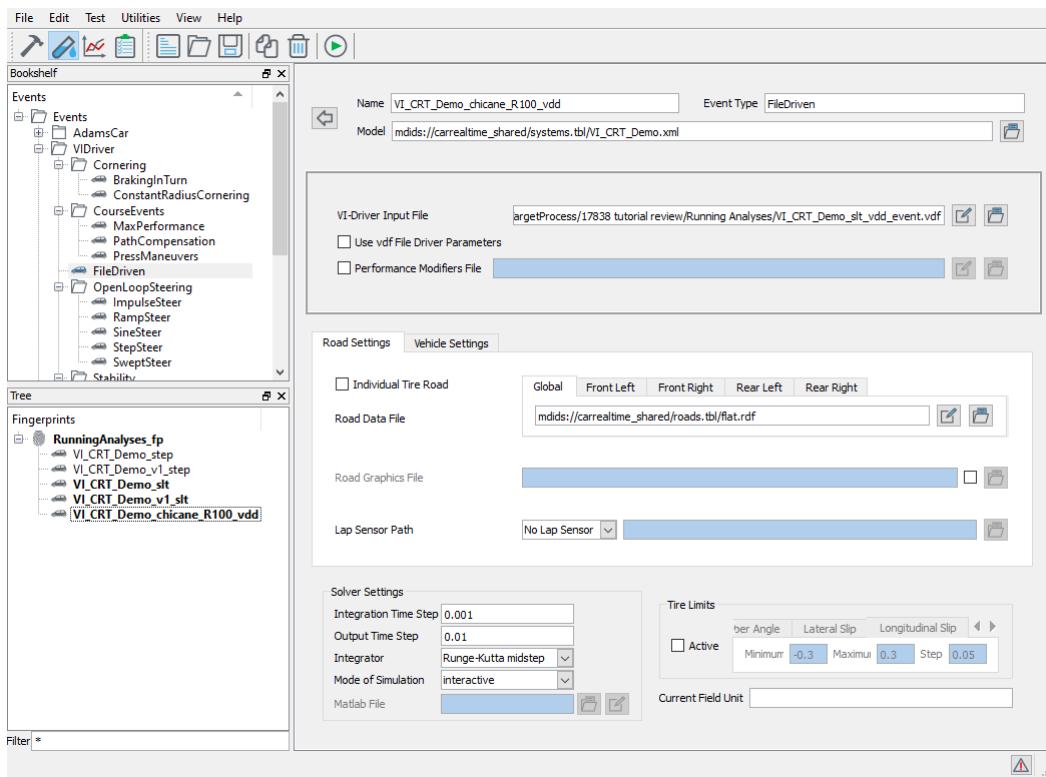
- **Model:** mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml
- **VI-Driver Input File:** VI_CRT_Demo_chicane_R100_slt_vdd_event.vdf

The input file will now have been generated by VI-SpeedGen within your working directory, and may be located easily using the **vdd_event.vdf** filter.

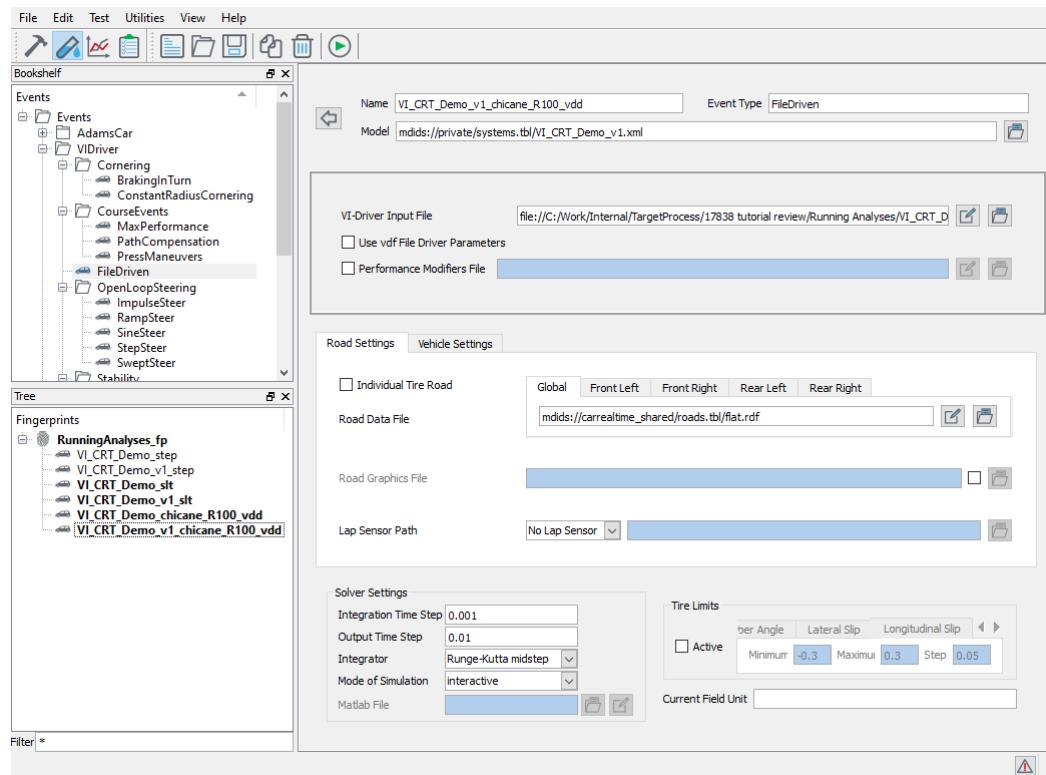
Rename the corresponding event file to **VI_CRT_Demo_chicane_R100_vdd**.



VI-CarRealTime



Copy the **VI_CRT_Demo_chicane_R100_slt_vdd** event, set the Model field to use the **VI_CRT_Demo_v1** model, and rename the copied event **VI_CRT_Demo_v1_chicane_R100_slt_vdd**.

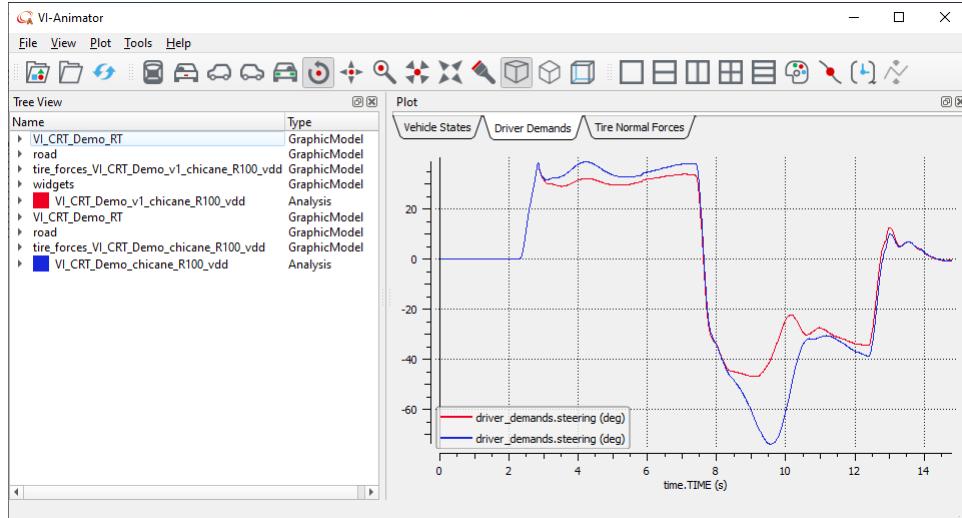


Run both events by clicking on the button after selecting them together. Events may be selected together either by control-clicking one after another, or by "lassoing" them in the menu using only the mouse.

Switch to **Review** mode and select the **VI_CRT_Demo_chicane_R100_vdd** and

VI_CRT_Demo_v1_chicane_R100_vdd events from the left hand tree view, and then click the  button.

VI-Animator will show up allowing the resulting animation to be visualized.



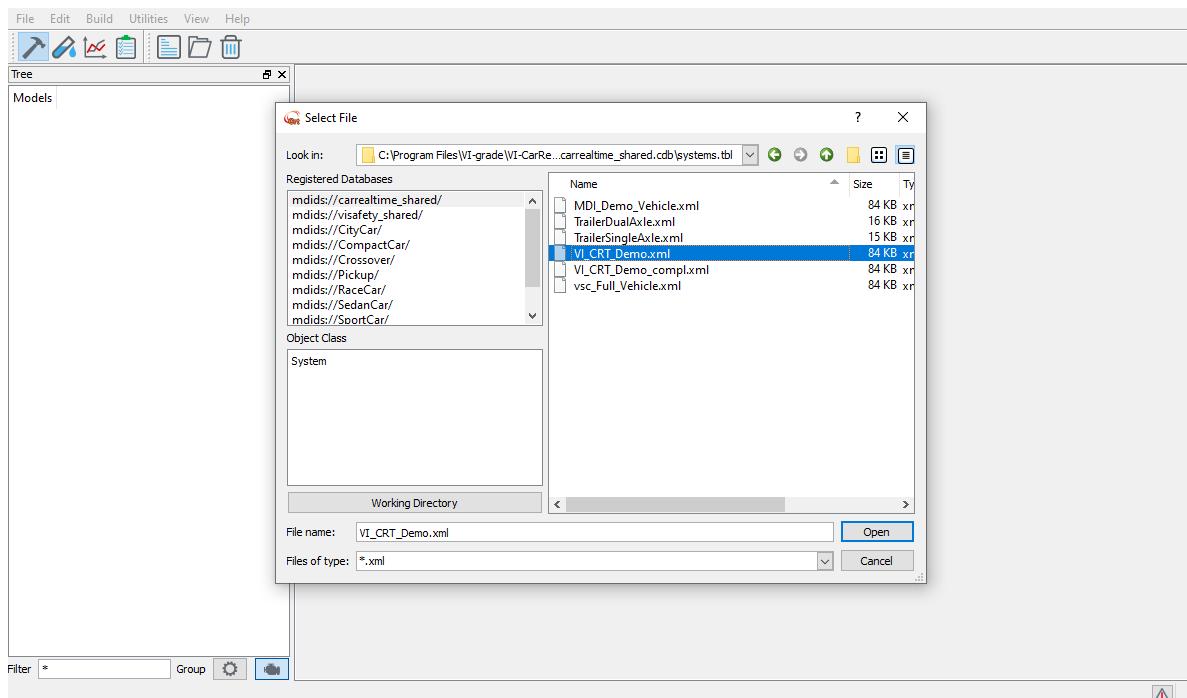
For the specific test case that we just set up and observed, dynamic simulation clearly enhances the understeering limitation effects to dynamic performance for our base vehicle (**VI_CRT_Demo** model). On the other vehicle (**VI_CRT_Demo_v1** model), the adjustment of the anti roll bar stiffness ensures better handling performance and reduced steering demand, as is obvious when viewing the animation or the corresponding plots.

Using MaxPerformance

The following tutorial shows how to run a **MaxPerformance** event in VI-CarRealTime.

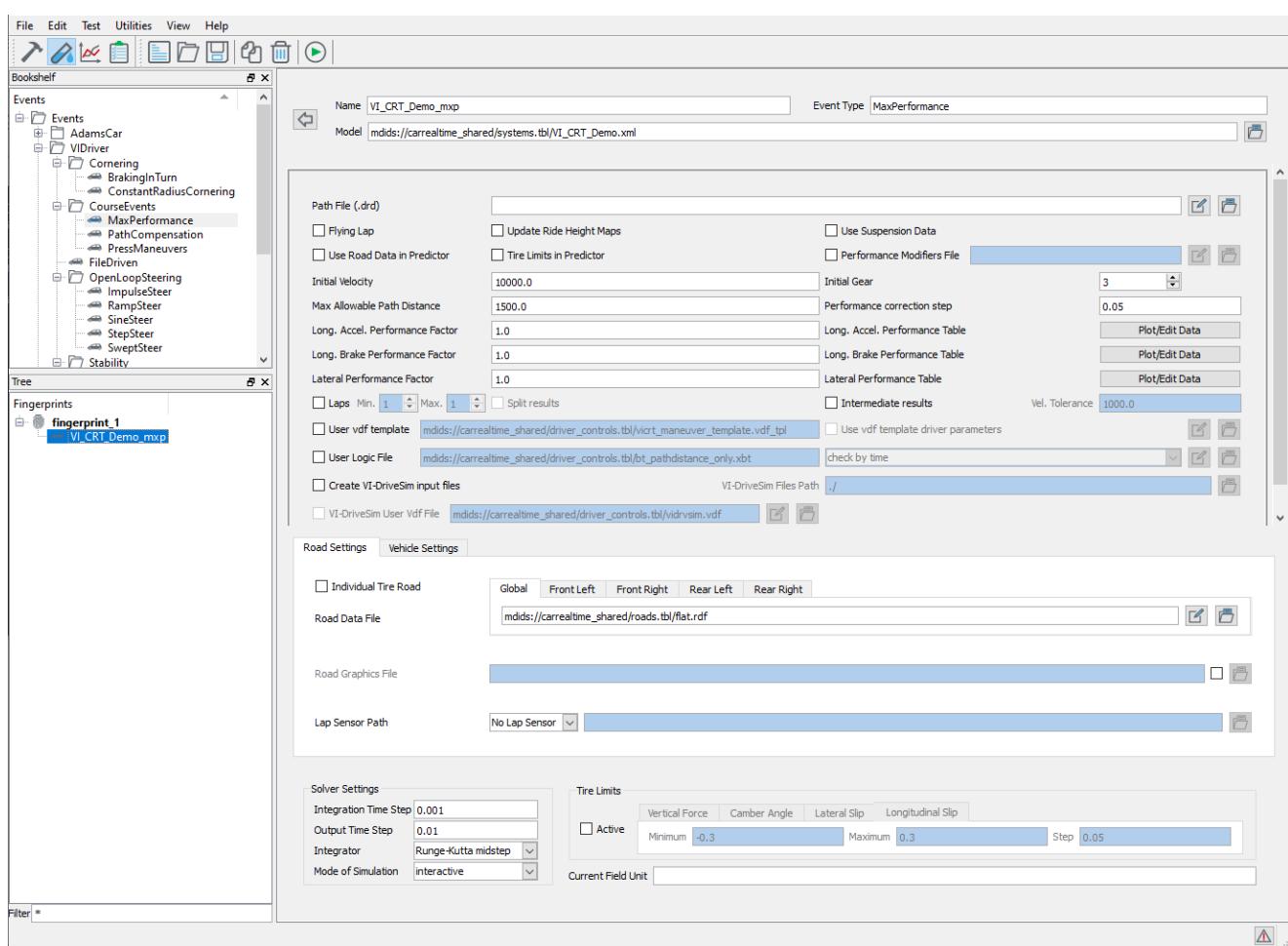
1. Load model

In **Build mode**, go to Build -> Load Model, and load the "VI_CRT_Demo.xml" vehicle model from the `carrealtime_shared` database.



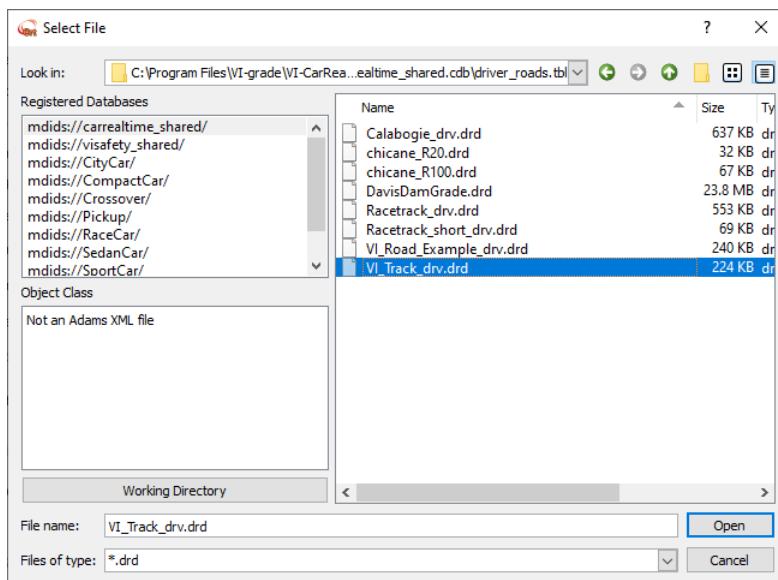
2. Create a MaxPerformance event

Switch to **Test Mode** and double click on *Event -> VIDriver -> CourseEvent -> [MaxPerformance](#)* :

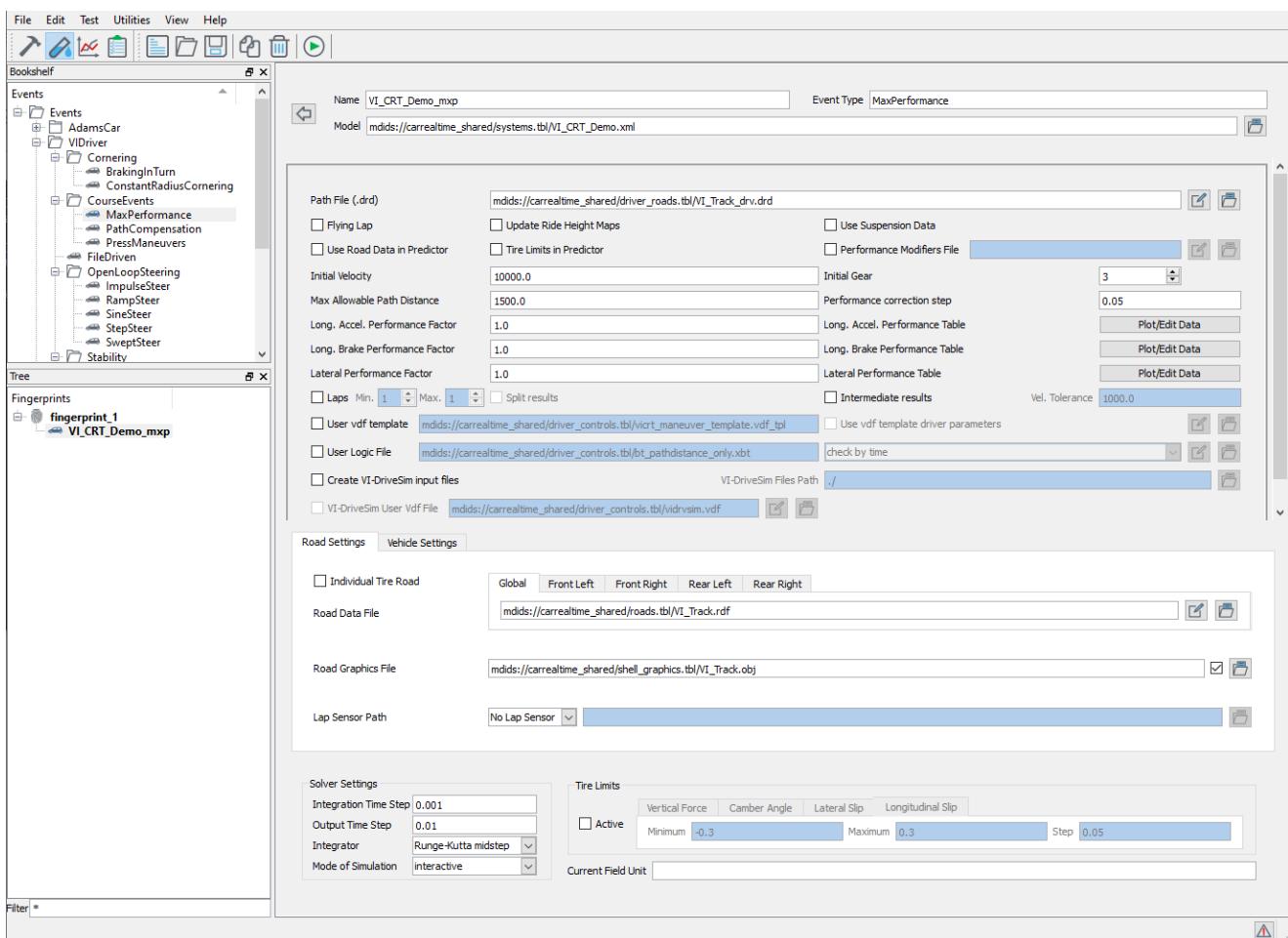


3. Set up Parameters

Using the open button near the **Path File (.drd)** field, select the "VI_Track_drv.drd" reference trajectory file and click on Open.



Set the file "VI_Track.rdf" as the Road Data File, and load the corresponding graphic model ("VI_Track.obj") into the Road Graphics File field below.



4. Run Event

Select the "interactive" mode of simulation and click on the button.

```
VI.PTW (VI-CarRealTime Python Task Window)

*** *****
*** Iteration 1 ***
*** *****

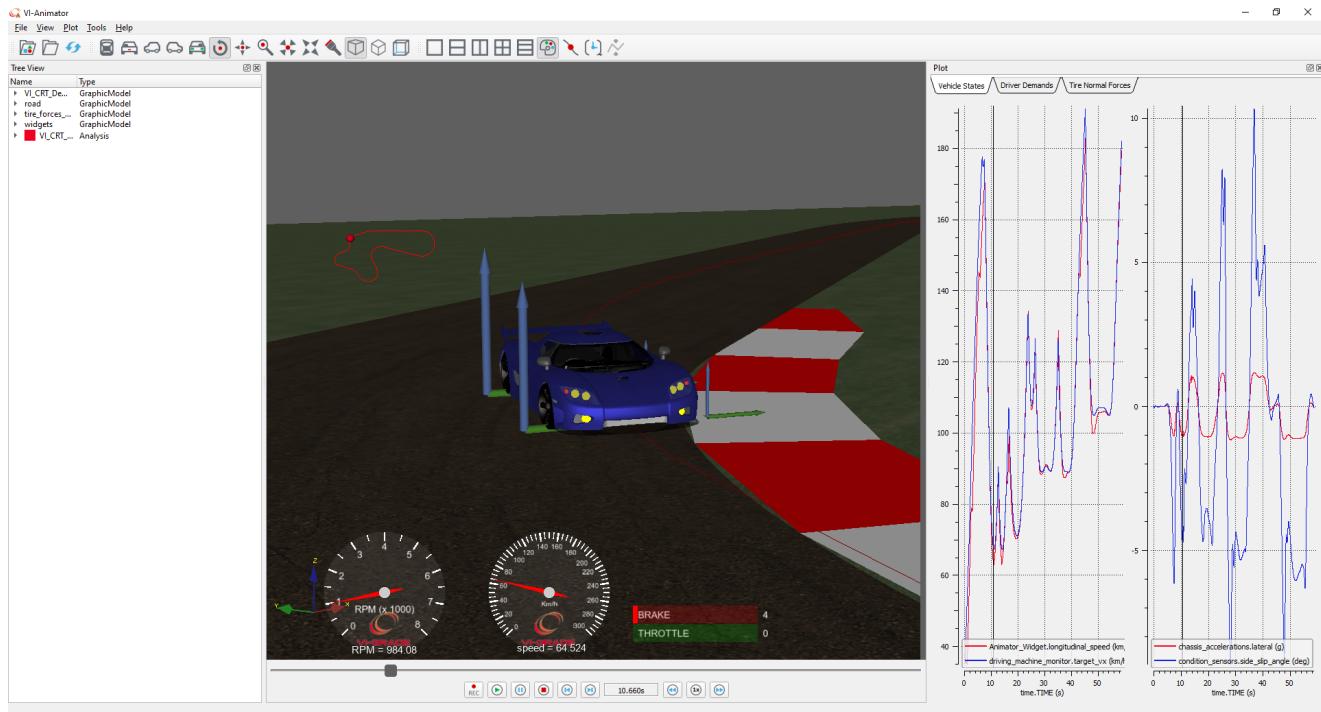
Calc Speed Profile
Predicted Laptime=57.046

Solving for static equilibrium...
Static equilibrium has been achieved.
Elapsed Simulation Time: 2.860 (sec).
Elapsed Clock Time: 0.464 (sec).
Running CRT Simulation...

Performing dynamic simulation...
Simulation Progress (percent complete):
0 50 100
---- 
--- Found unfeasibility evaluating condition 'Condition_Path_Distance' ---
--- Found unfeasibility evaluating condition 'Condition_Slip' ---
Unfeasibility at s=927.589778
--> Adding 'CustomCommand' on segment 5 [ 686.645 - 910.602]
--> Adding 'CustomCommand' on segment 6 [ 910.602 - 1190.569]
--> Scalings: Acc_performance= 0.000
```

5. Review results

After the simulation is complete, switch to **Review mode** and click on the film reel  button (labeled **Execute Postprocessing**) to automatically load the simulation result file and the corresponding graphic model into VI-Animator.

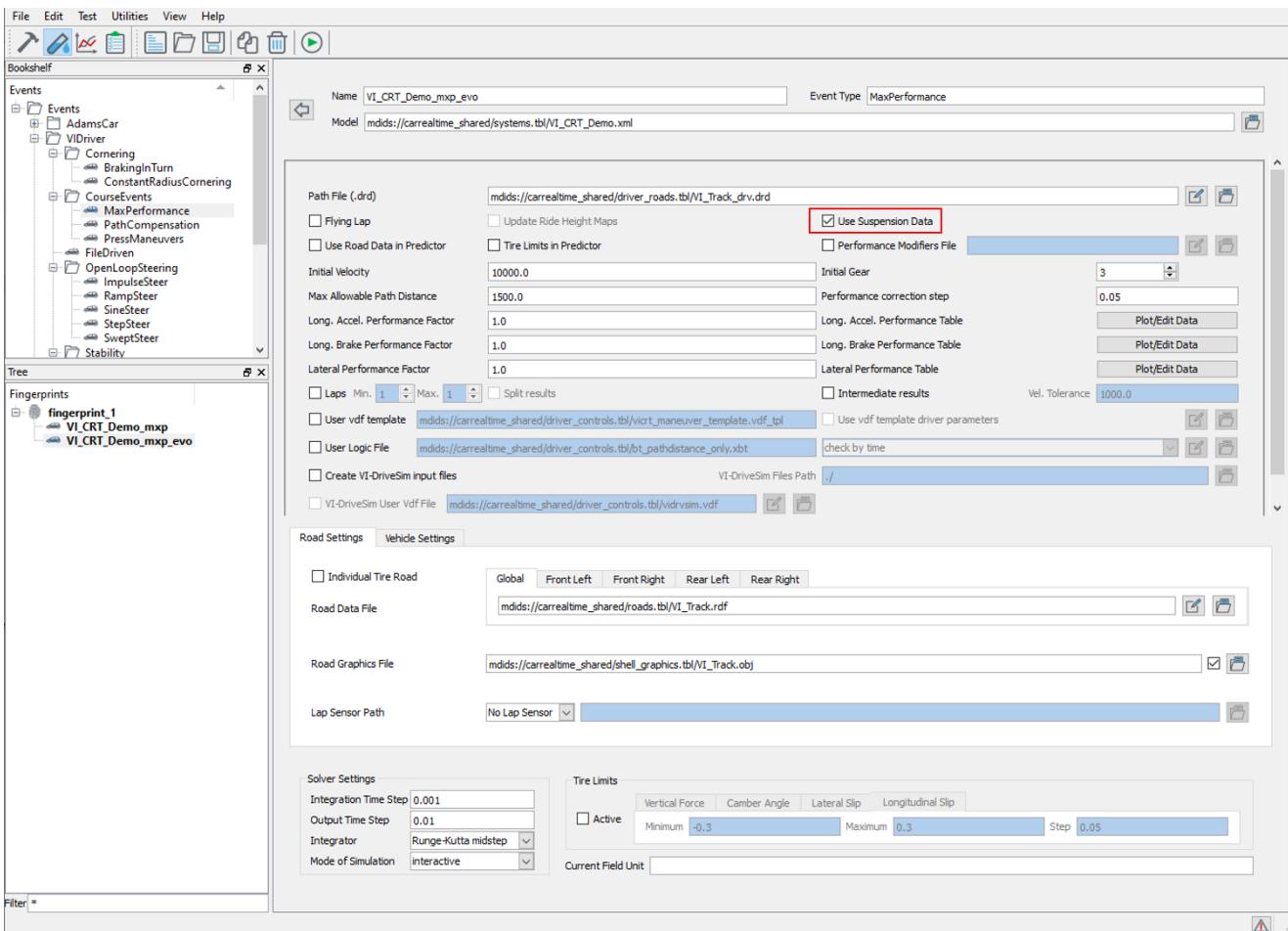


6. Run event with VI-SpeedGenEvo

Copy the event just run and rename it "VI_CRT_Demo_mxp_evo".

Check Use Suspension Data toggle button to enable VI-SpeedGenEvo internal model and keep the same values for all the other event parameters.

VI-CarRealTime

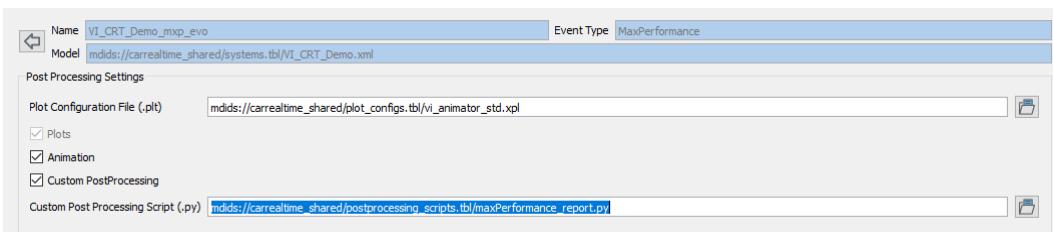


Click on the button to run the event.

7. Activate Custom PostProcessing Script and Compare the results

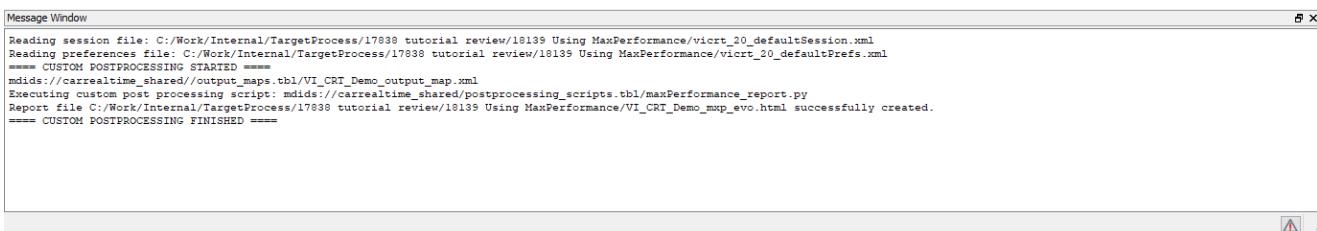
When the simulation is complete, switch to **Review mode**.

Click on `VI_CRT_Demo_mxp_evo` analysis and activate [Custom PostProcessing](#) flag:



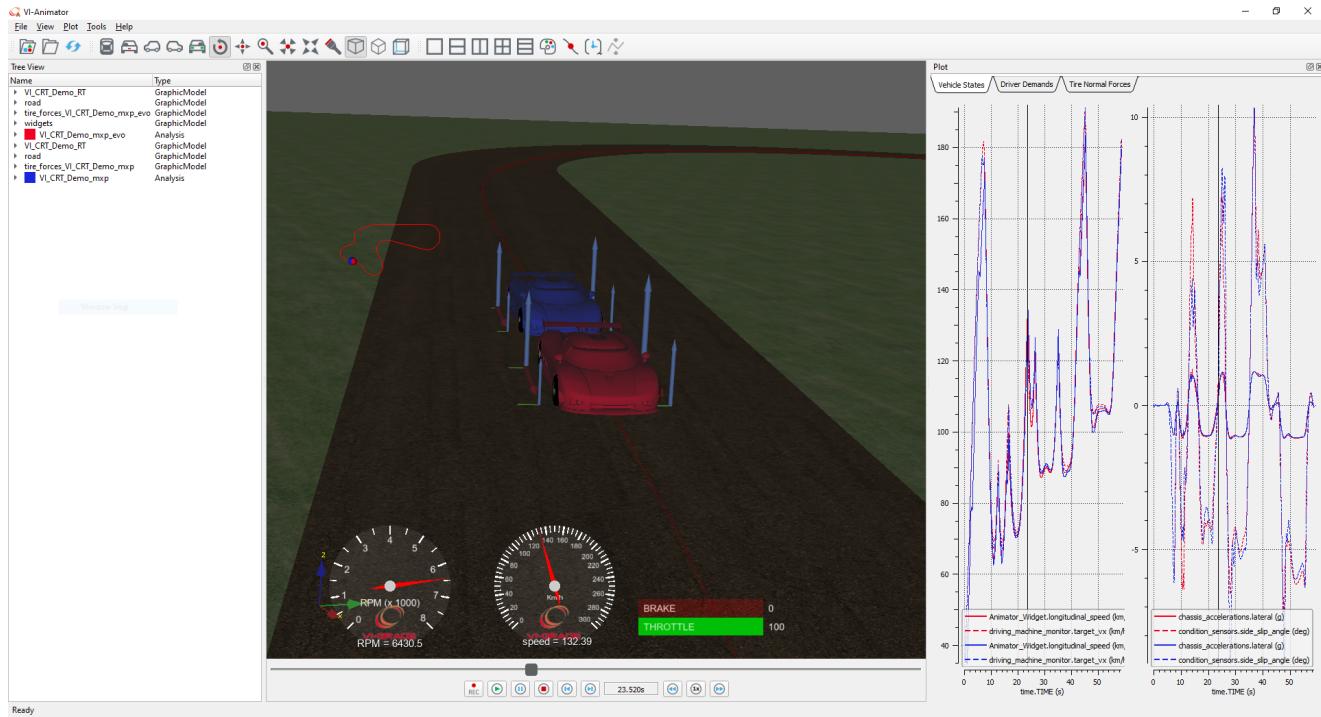
For the MaxPerformance event, VI-CarRealTime by default uses `maxPerformance_report.py`.

The script is automatically executed when the post processing button is pressed.

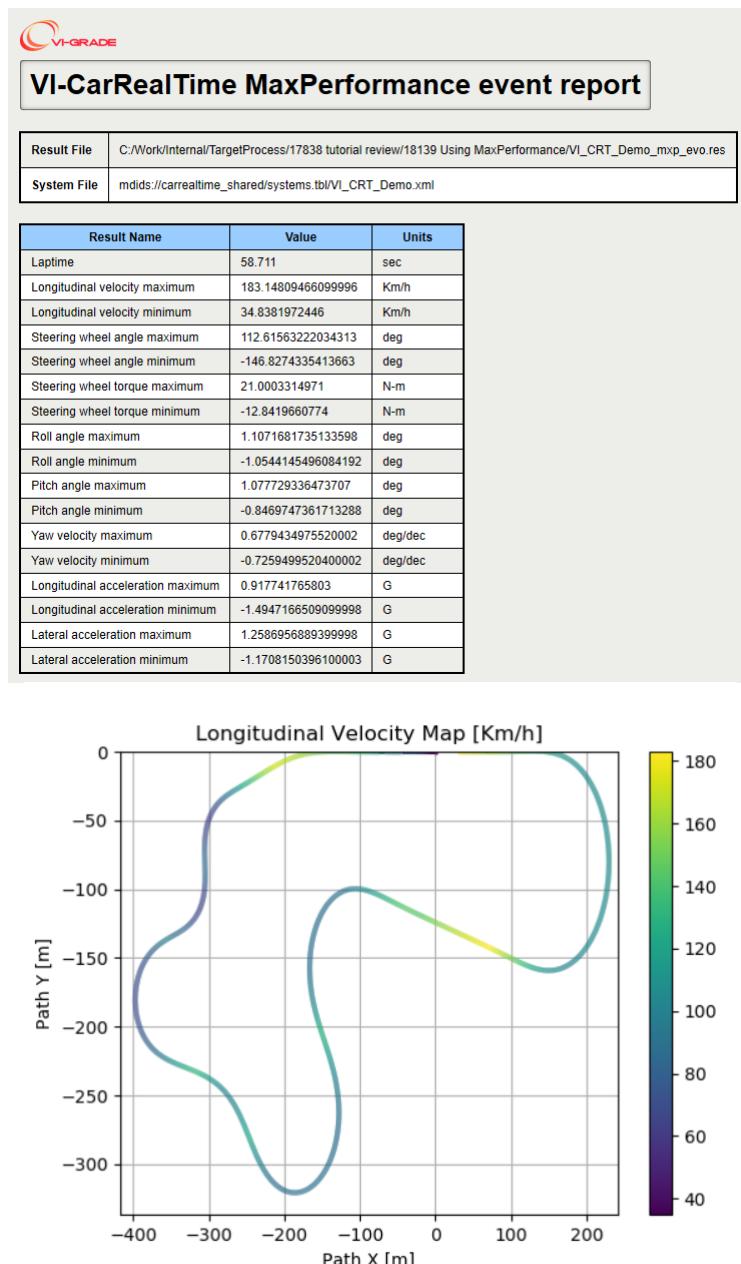


The script generates a HTML report in VI-CarRealTime working directory.

Select both analyses and click on the film reel  button (labeled **Execute Postprocessing**) to automatically load both the simulation results in VI-Animator.



Go to VI-CarRealTime working directory and open [VI_CRT_Demo_mxp_evo.html](#).



The report is composed by:

- header section
- table with maximum and minimum values of particular result channels
- plots

Note: the user can customize the python script according to his requirements.

Using MF-Tyre/MF-Swift tires

The goal of this tutorial is to show the user how to replace a standard tire with an MF-Tyre/MF-Swift one.

The MF-Tyre/MF-Swift interface allows the user to run analyses using TASS models to detect and calculate tire slip, and relevant forces between the road and a tire.

In order to accurately model and represent the tire behaviour in a wide range of analyses (from steady-state up to high frequency), MF-Tyre/MF-Swift includes two different tire modules:

- **MF-Tyre**

Implementation of Pajecka Magic Formula, an approach which can be used for both steady-state and transient events (handling analysis, control prototyping and rollover analysis).

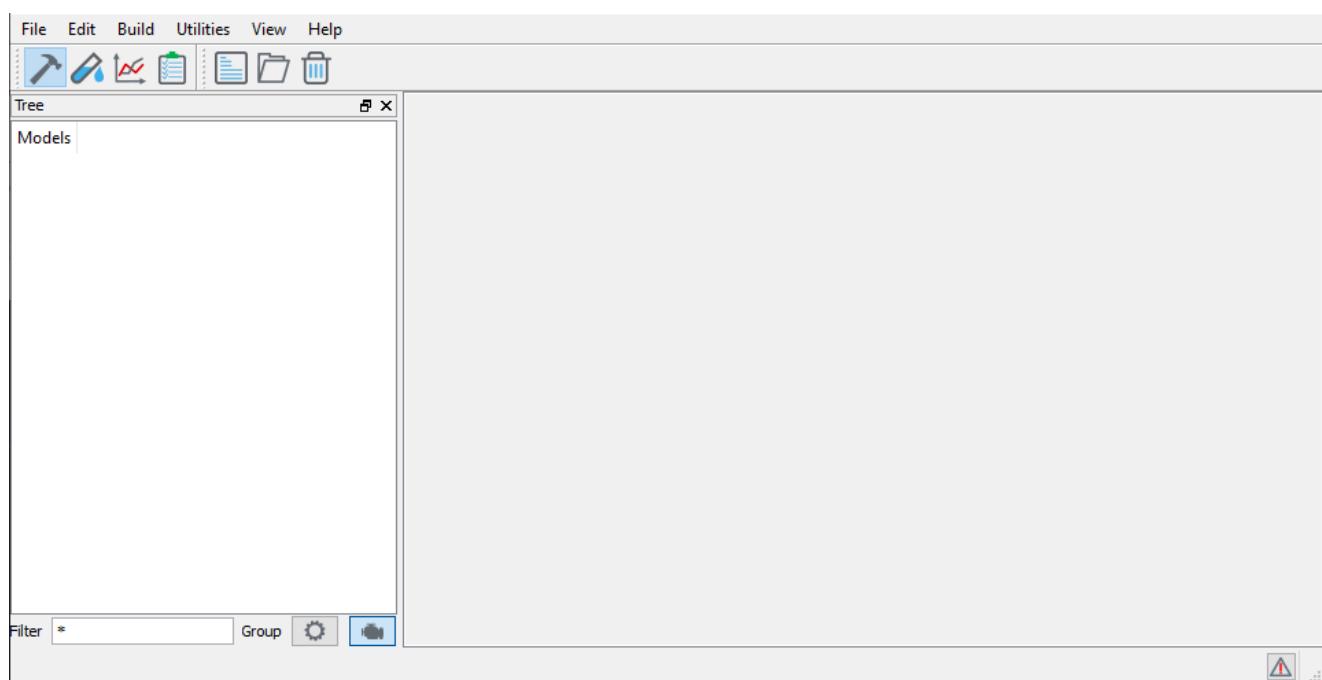
- **MF-Swift**

Extension of MF-Tyre with a rigid ring model in which the tire belt is modeled as a rigid body and an advanced tyre-road contact model; this approach can be used to simulate dynamic tire behaviour up to 100 Hz (for durability, or vibration and comfort analyses).

For a more details regarding the TASS tire models, please refer to

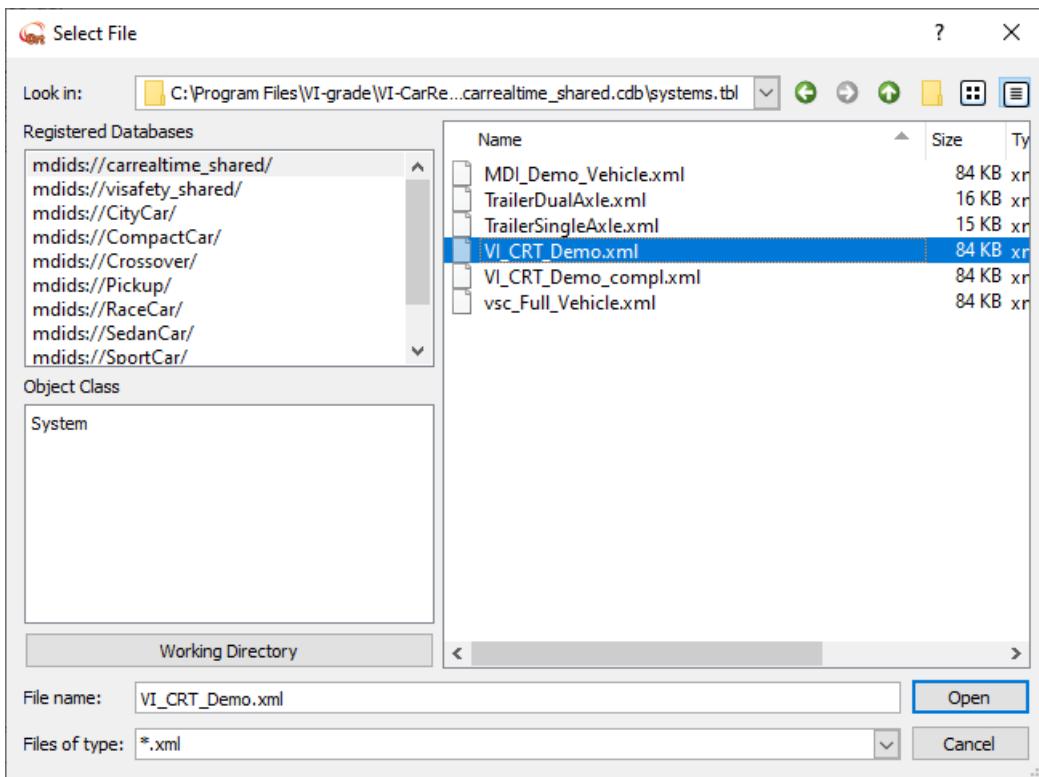
- [MF-Tyre/MF-Swift v6.x reference guide](#)
- [MF-Tyre/MF-Swift v7.x reference guide](#)

Start a new VI-CarRealTime session using the Windows Start Menu or by typing vicrt20 from a DOS (cmd) shell.



Open **VI_CRT_Demo.xml** by clicking on the icon in the VI-CarRealTime toolbar and browse to the `carrealtime_shared` database.

VI-CarRealTime

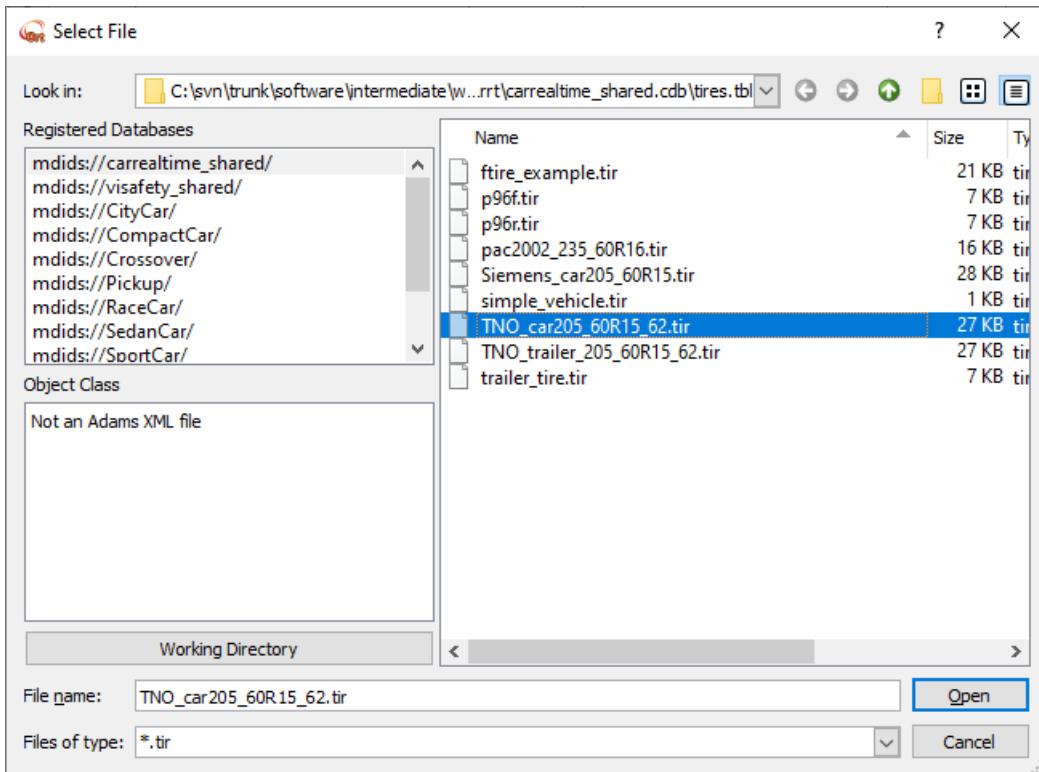


Let's now replace the standard model tires with MF-Tyre/MF-Swift tires for both the front and rear tires.

Select the **RT_VI_CRT_Demo_front_wheels** subsystem in the **tree view**.

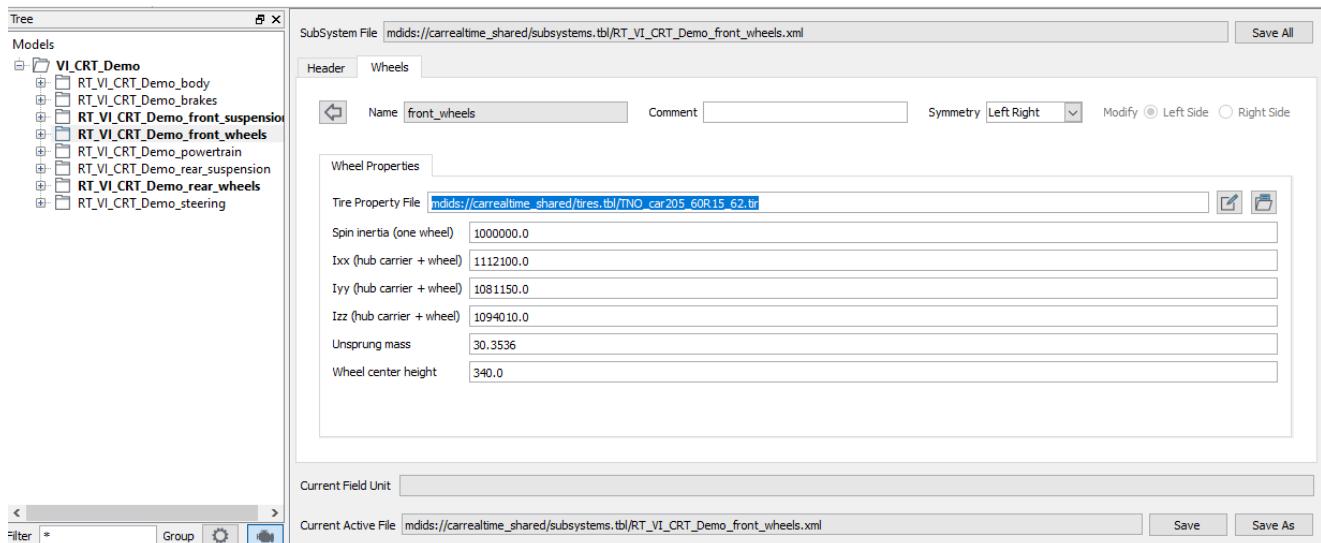
Note: the default tire model is *mdids://carrealtime_shared/tires.tbl/p96f.tir*.

Click on the icon. The following dialog box appears as shown below.



From the **carrealtime_shared** database select the **TNO_car205_60R15_62.tir** file and press **Open**.

Repeat the same procedure for the rear wheel (**RT_VI_CRT_Demo_rear_wheels**) subsystem. Now, you will have replaced the standard model tires with MF-Tyre/MF-Swift tires.



By clicking the button, you can view and/or edit the Tire Property File.

Browse through the .tir file above, and under the **[MODEL]** block, you will find the **USE_MODE** parameter, which allows you to choose the desired mode for tire force computation.

There are a lot of combinations of values which can be chosen. For a detailed explanation of all available modes please refer to the MF-Tyre/MF-Swift topic.

For this particular tutorial, change the USE_MODE value to **544**.

Note: this mode is an [MF-Swift](#) mode requiring a special license, as is elaborated upon in the [TNO reference guide](#).

The following outputs (please refer to Tire Output Map topic for the details) related to tire model will be available in Review Mode:

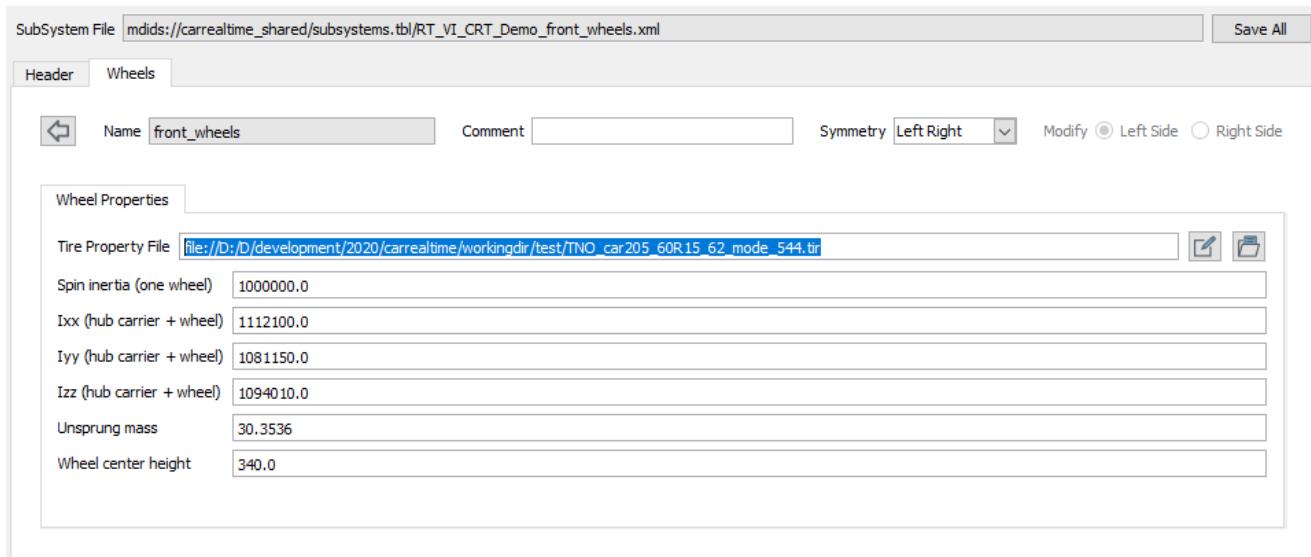
- longitudinal, lateral, vertical forces
- aligning, overturning, rolling resistance moments
- inclination angle
- longitudinal, lateral slip with and without lag
- actual and free omega
- loaded radius
- rolling radius
- contact patch location and velocity vectors
- contact patch direction cosines
- effective closing speed, effective penetration, effective rolling radius
- ground surface gyroscopic moment and residual torque
- longitudinal and lateral relaxation length
- moment arm and pneumatic trail

Save the file as **TNO_car205_60R15_62_mode_544.tir**,

do not forget to select this new tire property file for both

front and rear tires, as shown in the following figure:

VI-CarRealTime



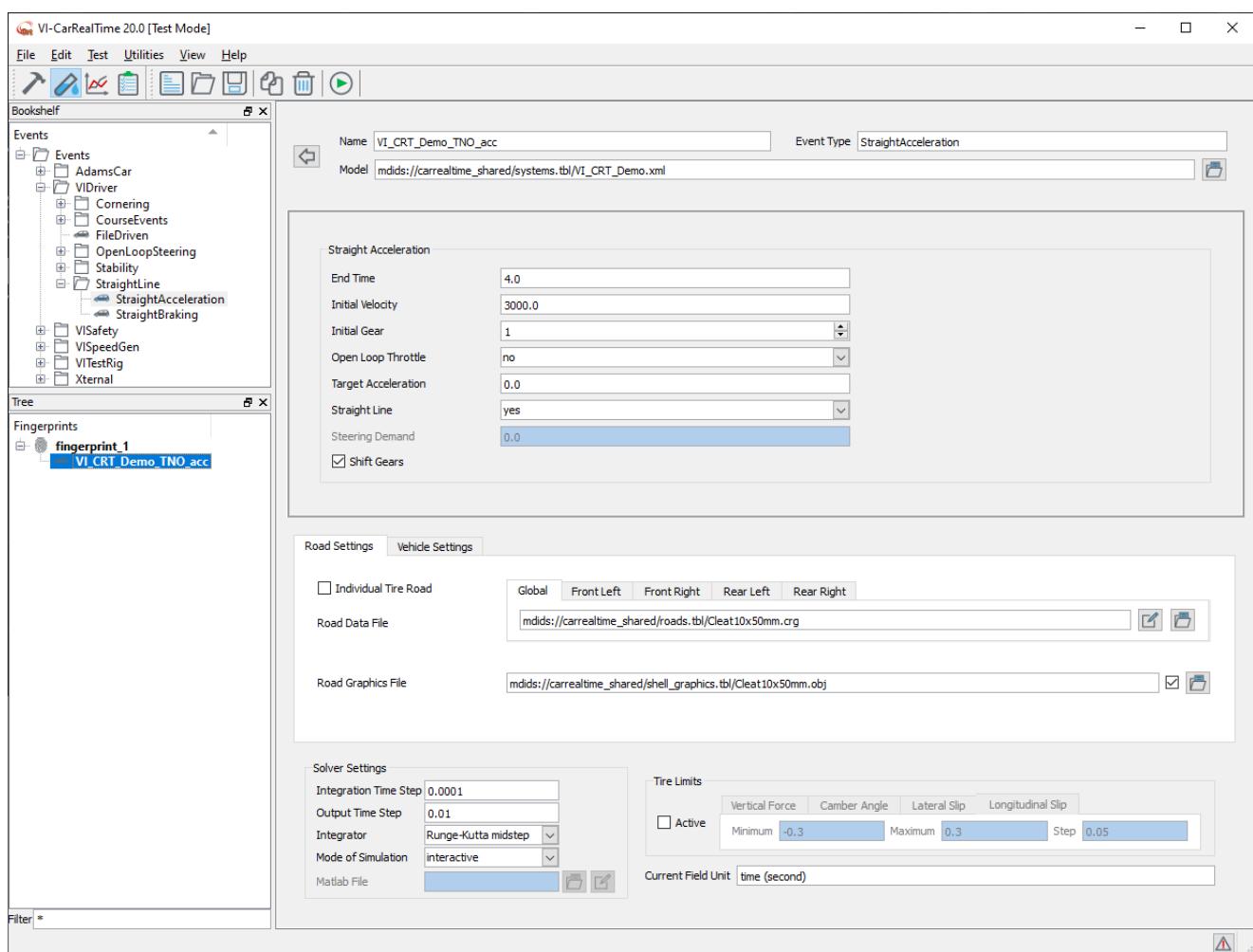
Switch to **Test** mode using the  button in the VI-CarRealTime toolbar.

Add a new **Straight Acceleration** event (go to *Events* -> *VIDriver* -> *StraightLine* -> *StraightAcceleration*) and set the following simulation parameters:

- End Time = **4.0** s;
- Initial Velocity = **3000.0** mm/s;
- Initial Gear = **1**;
- Open Loop Throttle = **no**;
- Target Acceleration = **0.0** mm/s²;
- Straight Line = **yes**;

- For the Road Data File, select the *mdids://carrealtime_shared/roads.tbl/* **Cleat10x50mm.crg** file in the *carrealtime_shared* database;
- Check the Road Graphics File checkbox, and select *mdids://carrealtime_shared/shell_graphics.tbl/* **Cleat10x50mm.obj** from the *carrealtime_shared* database;

- set **0.0001** as the Integration Time Step; this Integration Time Step is required to ensure the stability of the simulation when using the advanced tyre modelling option of MF-Tyre/MF-Swift.



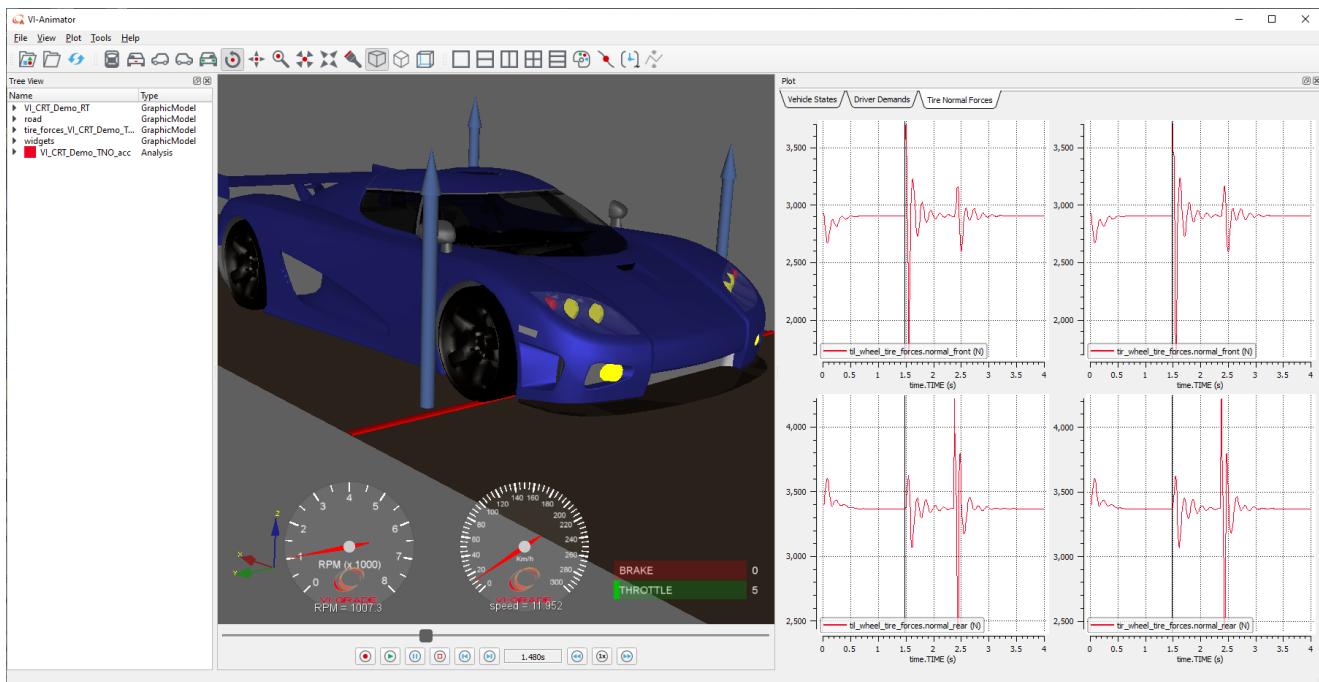
- Rename the event to **VI_CRT_Demo_TNO_acc** and run the simulation by clicking on the button in the toolbar.

Switch to Review mode using the button.

Select the **VI_CRT_Demo_TNO_acc** event with a left mouse click and then click the button. VI-Animator will start loading event results and a corresponding animation.

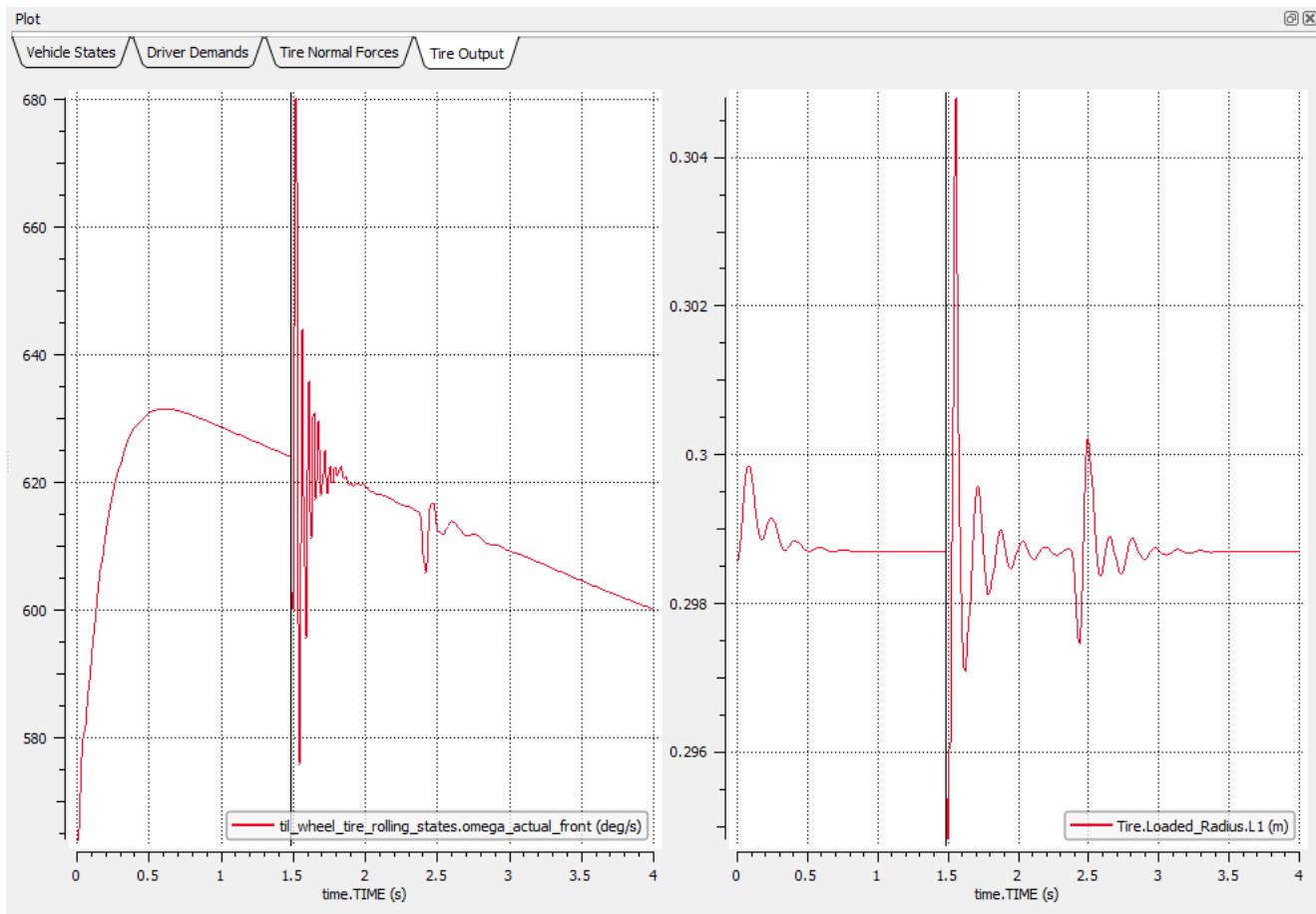
Switch to the **Tire Normal Forces** tab to see the tire normal force oscillations resulting from our particular cleat setup.

VI-CarRealTime



You may additionally add a new custom tab (2x1), and plot the following, as described and pictured below:

- *til_wheel_tire_rolling_states.omega_actual_front* vs. time;
- *Tire.Loaded_Radius.L1* vs. time;



Fuel Consumption

The purpose of this particular VI-CarRealTime tutorial is to give an overview of the application's fuel consumption calculation features.

VI-CarRealTime estimates the fuel consumption throughout a maneuver when a **fuel consumption map** is provided in the powertrain subsystem. During the various steps of the tutorial, an example of the fuel consumption map will be explored.

The Fuel Consumption tutorial divided into the following sections:

- [Model Set-up](#)
- [Run Analysis](#)
- [Review Results](#)

For the following exercise, fuel consumption for a car (equipped with a manual transmission) will be evaluated under the New European Driving Cycle (NEDC).

The NEDC consists of four repeated ECE-15 *Urban Driving Cycles* (UDC) and one *Extra-Urban Driving Cycle* (EUDC), whose steps are described in detail below:

Urban Driving Cycles (UDC)

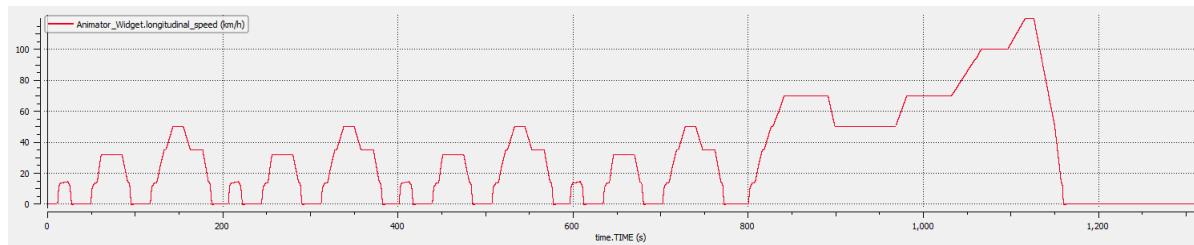
- The engine starts, and then the car pauses for 11 s; next, the car slowly accelerates to 15 km/h in 4 s, cruises at a constant speed for 8 s, brakes to a full stop in 5 s (in the last 3 s, the car's clutch is disengaged), and then stops for 21 s.
- At 49 s, the car slowly accelerates to 32 km/h in 12 s, cruises at a constant speed for 24 s, slowly brakes to a full stop in 11 s, and then pauses for another 21 s.
- At 117 s, the car slowly accelerates to 50 km/h in 26 s, cruises at a constant speed for 12 s, decelerates to 35 km/h in 8 s, cruises at a constant speed for another 13 s, brakes to a full stop in 12 s, and then pauses (stops) for 7 s.
- The **UDC** cycle ends 195 s after a theoretical distance of 1017 meters, after which it repeats three consecutive times. The total duration of the four repeated **UDC** cycles is 780 s (or 13 minutes, as can be inferred from the plot below) over a theoretical distance of 4067 meters, with an average speed of approximately 18.77 km/h.

Extra-Urban Driving Cycles (EUDC)

- After a 20 s stop (starting at 780 s) in 1st gear, with the clutch disengaged, the car slowly accelerates to 70 km/h in 41 s, cruises at a constant speed for 50 s, decelerates to 50 km/h in 8 s, cruises at a constant speed for 69 s, and then slowly accelerates to 70 km/h in 13 s.
- At 201 s (keeping in mind the initial 780 s offset), the car cruises at a constant speed of 70 km/h for 50 s, slowly accelerates to 100 km/h in 35 s, and then cruises at a constant speed for 30 s.
- At 316 s (keeping in mind the initial 780 s offset) the car slowly accelerates to 120 km/h in 20 s, cruises at a constant speed for 10 s, slowly brakes to a full stop in 34 s, and then idles for another 20 s.
- The total duration is 400 s (or 6 minutes 40 seconds, between 780 s and 1180 s) and the theoretical distance is 6956 meters, with an average speed of approximately 62.6 km/h.

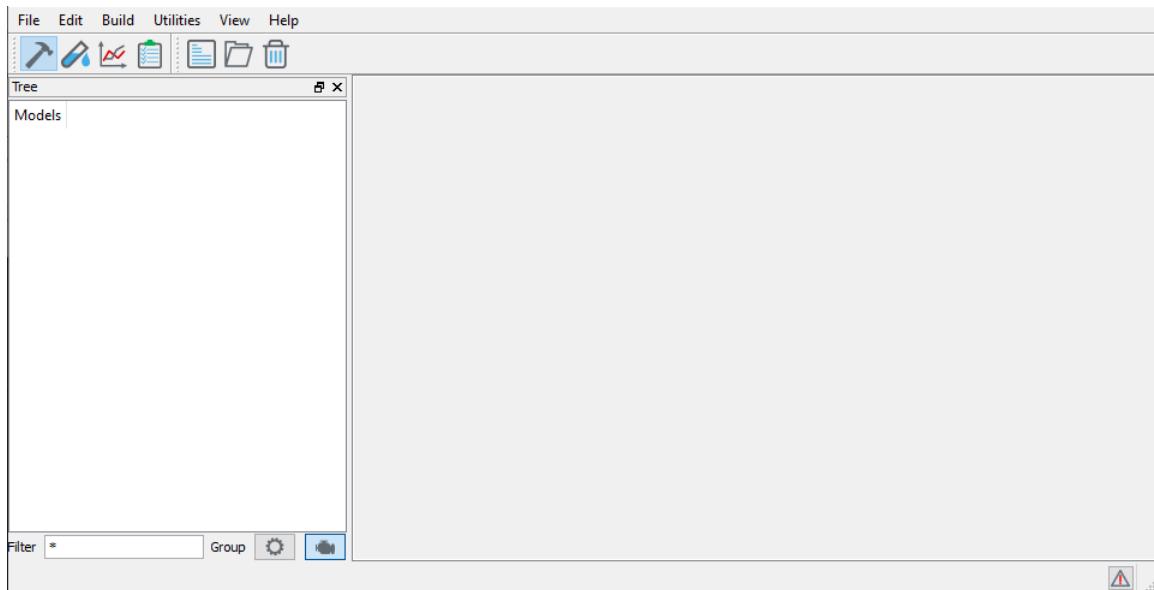
The resulting speed profile plot (in km/h) is shown beneath.

VI-CarRealTime

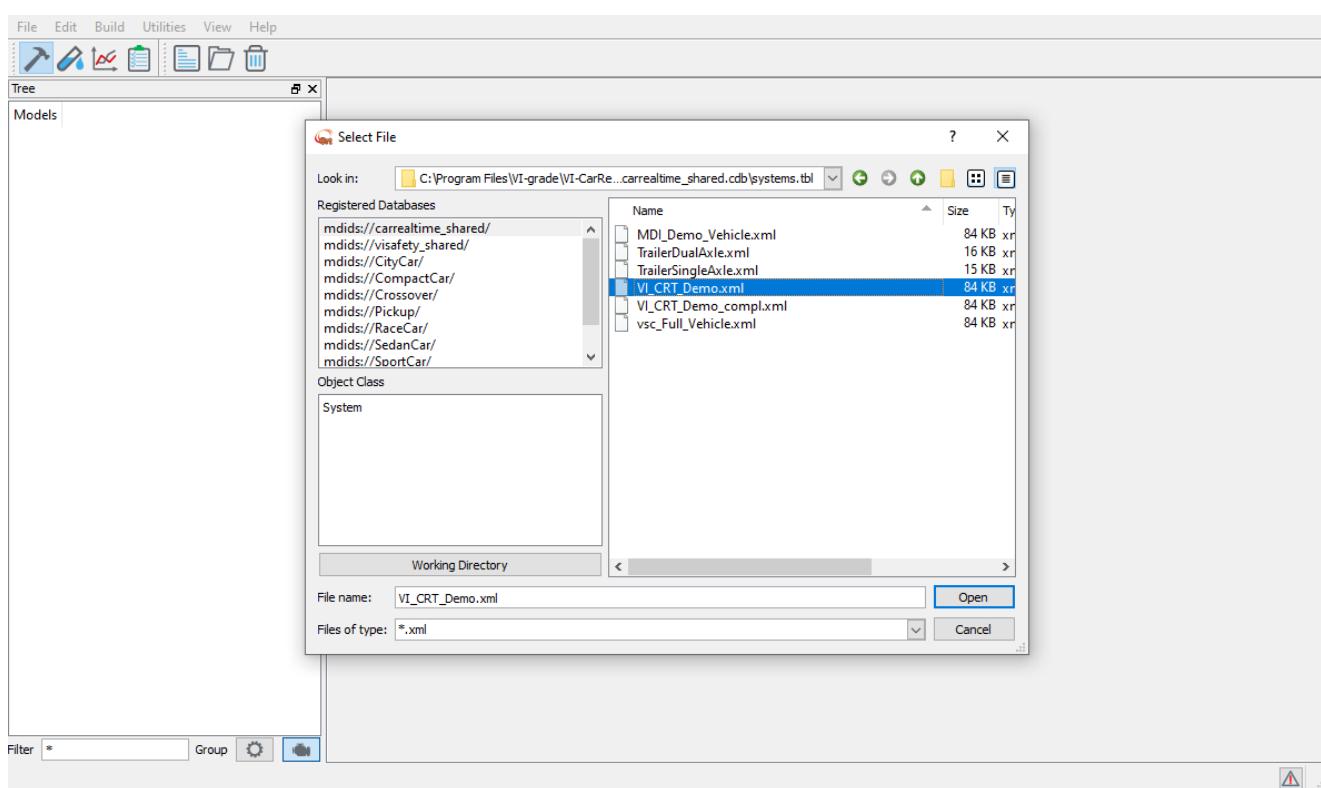


Model Set-up

Start a new VI-CarRealTime session from the Windows Start Menu or by typing **vicrt20** from a DOS (cmd) shell.

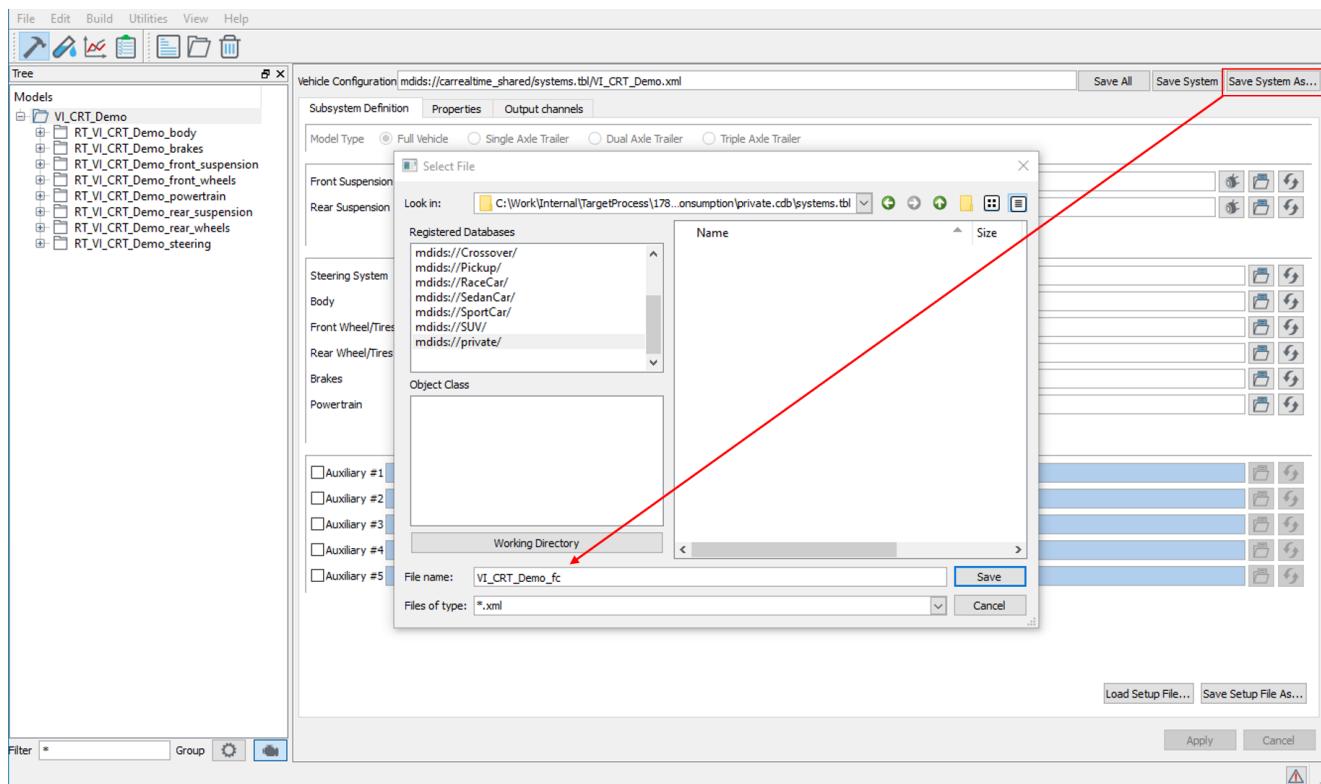


In **Build mode**, go to Build -> Load Model, and load the **VI_CRT_Demo.xml** vehicle model from the **carrealtime_shared** database.



The next step involves creating a private database, which will be used to store a modified version of the VI_CRT_Demo (so as to not disturb the shared database model, which is our default/baseline demo model).

Create a new database (please refer to the [Running Analyses](#) tutorial for more details about this procedure) and save the VI_CRT_Demo.xml model in your private database, renaming it **VI_CRT_Demo_fc.xml** as shown in the picture below:



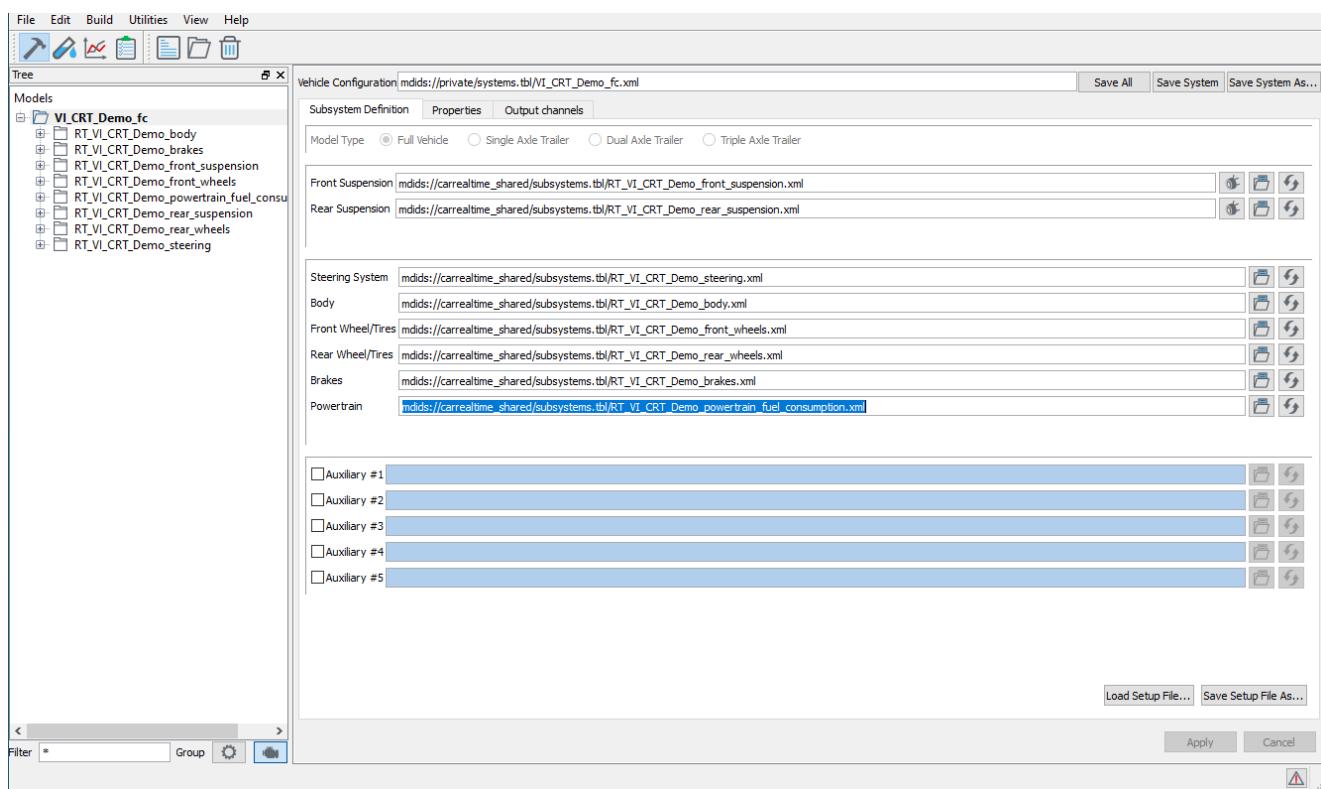
Now, we must replace the shared powertrain subsystem with another one, which embeds a fuel consumption map:

- select the VI_CRT_Demo_fc model



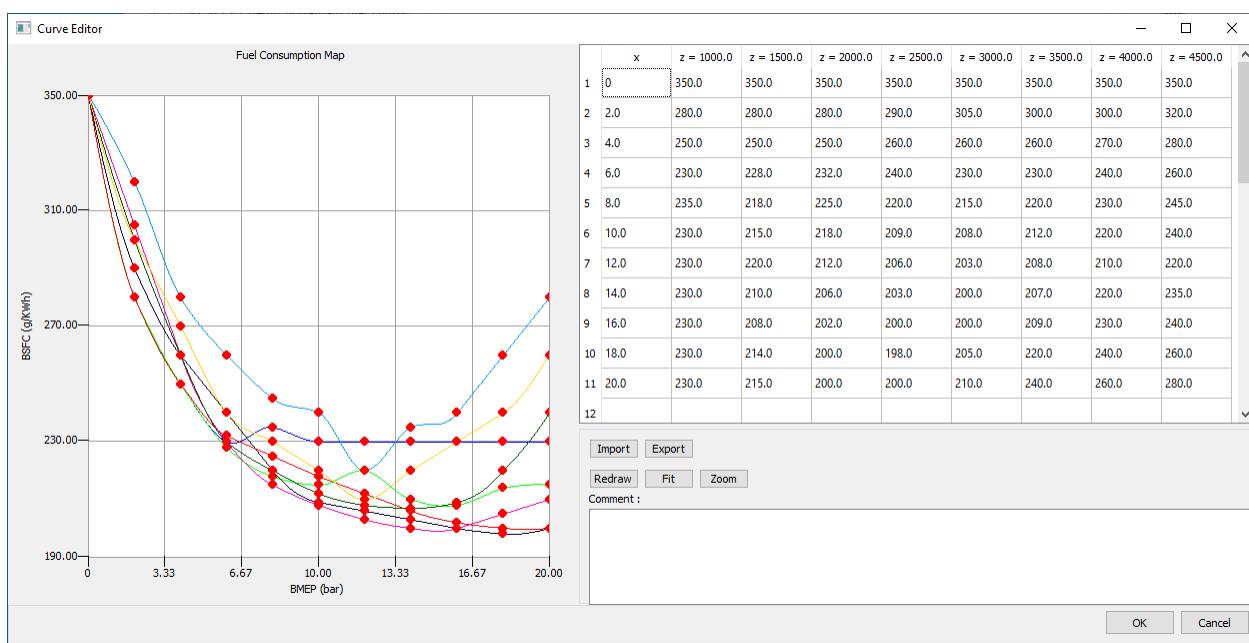
- press the button next to the **Powertrain** subsystem
- browse the `carrealtime_shared` database and select the `RT_VI_CRT_Demo_powertrain_fuel_consumption.xml` file
- press **Open** to select the subsystem
- within VI-CarRealTime's Build Mode, click on **Apply** in the bottom right of the window to use the newly tweaked fuel consumption subsystem in the **VI_CRT_Demo_fc** model.
- press **Apply** button.

The following screenshot shows the model's subsystem list updated with the new powertrain subsystem.

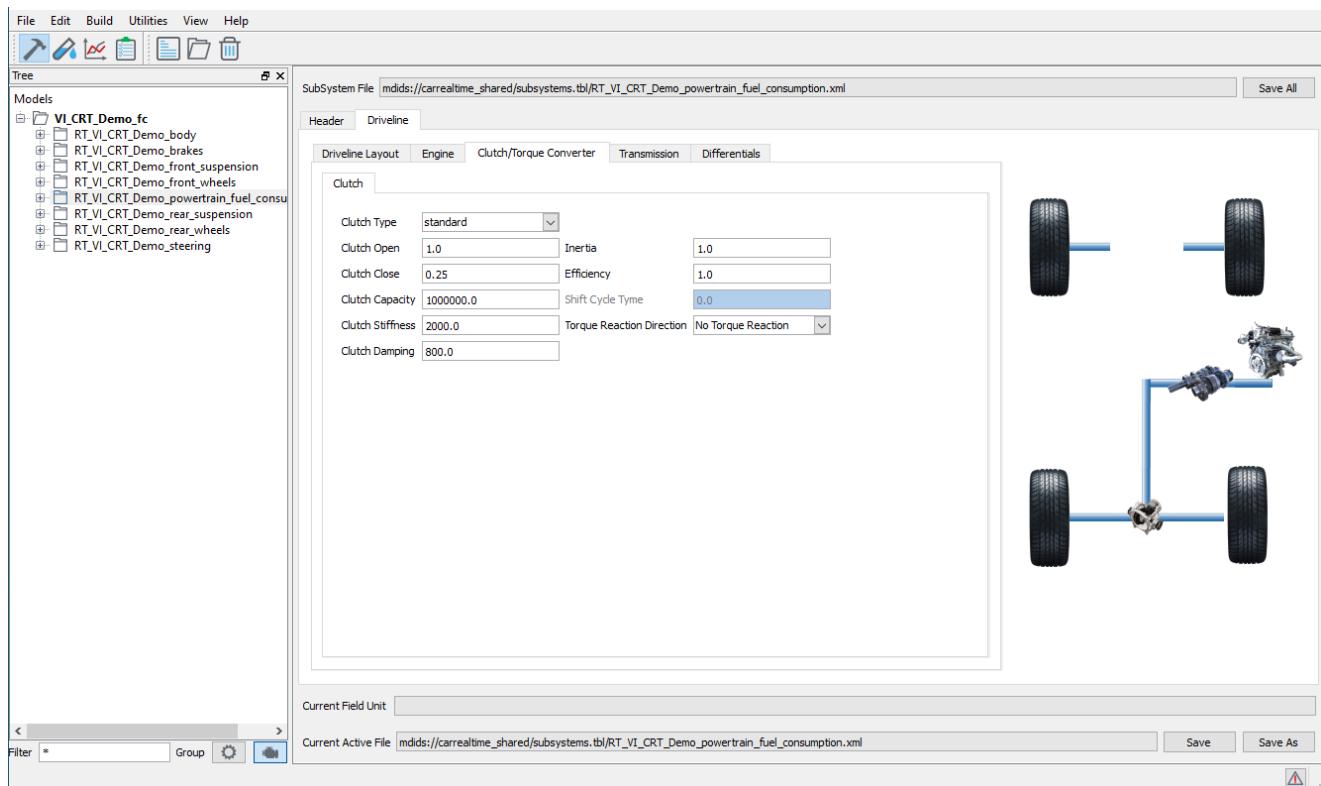


The most important settings of the `RT_VI_CRT_Demo_powertrain_fuel_consumption` subsystem, for this particular tutorial section, are:

- Fuel Consumption Map:** a spline which defines the BSFC (Brake Specific Fuel Consumption) as a function of BMEP (Brake Mean Effective Pressure) and RPM of the engine. It is stored in the Fuel Consumption table in the powertrain subsystem:



- **Clutch model:** allows the vehicle to start from standstill; must be set up as illustrated below:



Run Analysis

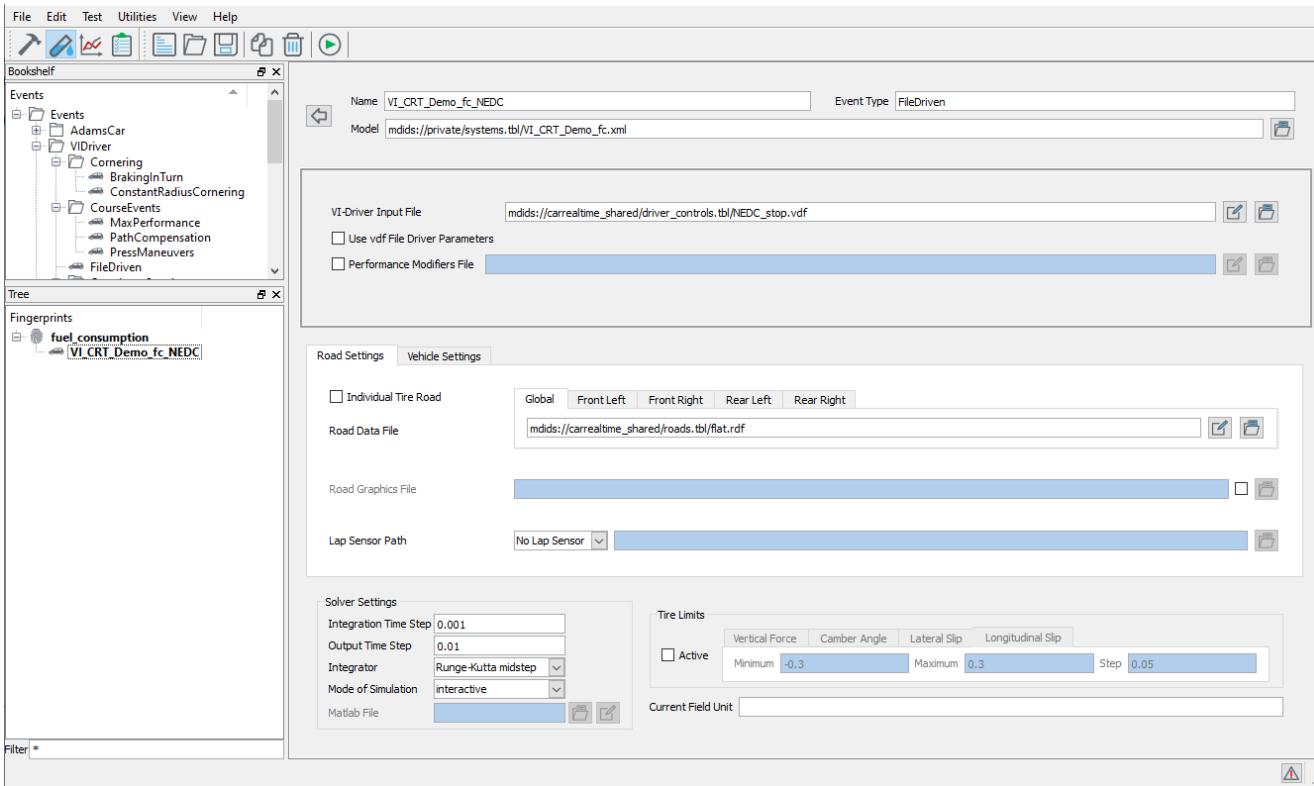
Switch to **Test** mode using the button.

Create a new fingerprint named **fuel_consumption**.

VI-CarRealTime

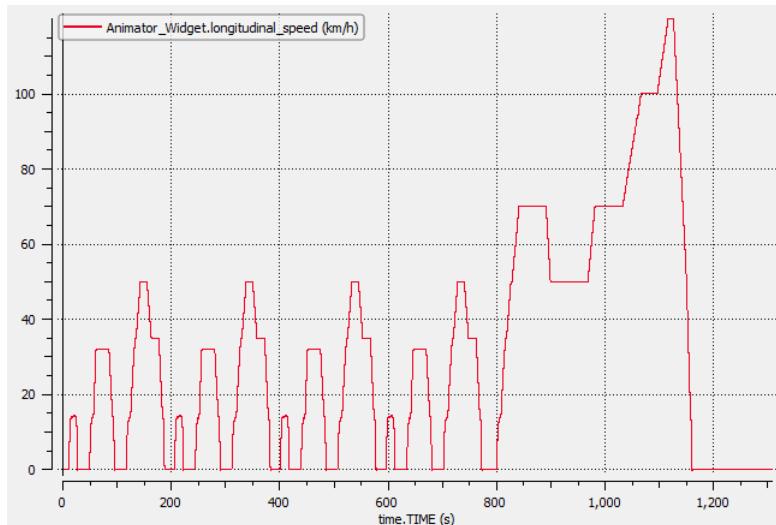
Browse to *Events* -> *VI-Driver* and add a new **FileDriven** event to the fingerprint, and rename it **VI_CRT_Demo_fc_NEDC**, using the *VI_CRT_Demo_fc* model.

Set *NEDC_stop.vdf* as the VI-Driver Input File, as shown in the picture below:

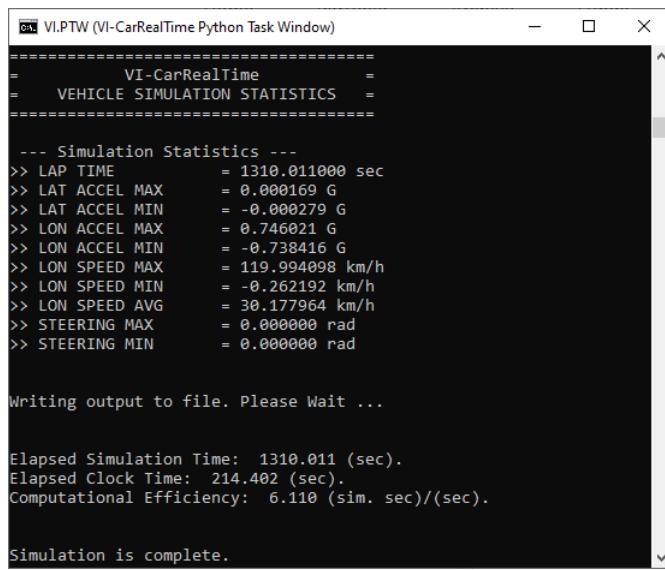


The NEDC_stop.vdf is composed of 5 maneuvers which allow the vehicle to perform all the actions required by the [New European Driving Cycle](#).

The target speed profile required by this cycle is shown in the following graph:



Run the event you just created by selecting the **fuel_consumption** fingerprint in the Test mode tree view, and the press the button in the toolbar.



```

VI.PTW (VI-CarRealTime Python Task Window)
=====
= VI-CarRealTime =
= VEHICLE SIMULATION STATISTICS =
=====

--- Simulation Statistics ---
>> LAP TIME      = 1310.011000 sec
>> LAT ACCEL MAX = 0.000169 G
>> LAT ACCEL MIN = -0.000279 G
>> LON ACCEL MAX = 0.746021 G
>> LON ACCEL MIN = -0.738416 G
>> LON SPEED MAX = 119.994098 km/h
>> LON SPEED MIN = -0.262192 km/h
>> LON SPEED AVG = 30.177964 km/h
>> STEERING MAX = 0.000000 rad
>> STEERING MIN = 0.000000 rad

Writing output to file. Please Wait ...

Elapsed Simulation Time: 1310.011 (sec).
Elapsed Clock Time: 214.402 (sec).
Computational Efficiency: 6.110 (sim. sec)/(sec).

Simulation is complete.

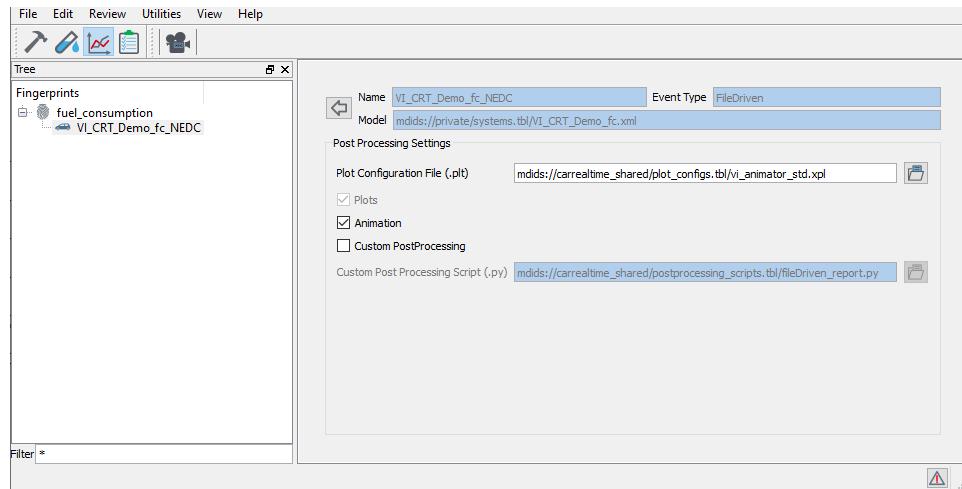
```

Wait until the analysis is completed, after which you should see a DOS (cmd) shell with an output similar to the one shown above.

Review Results

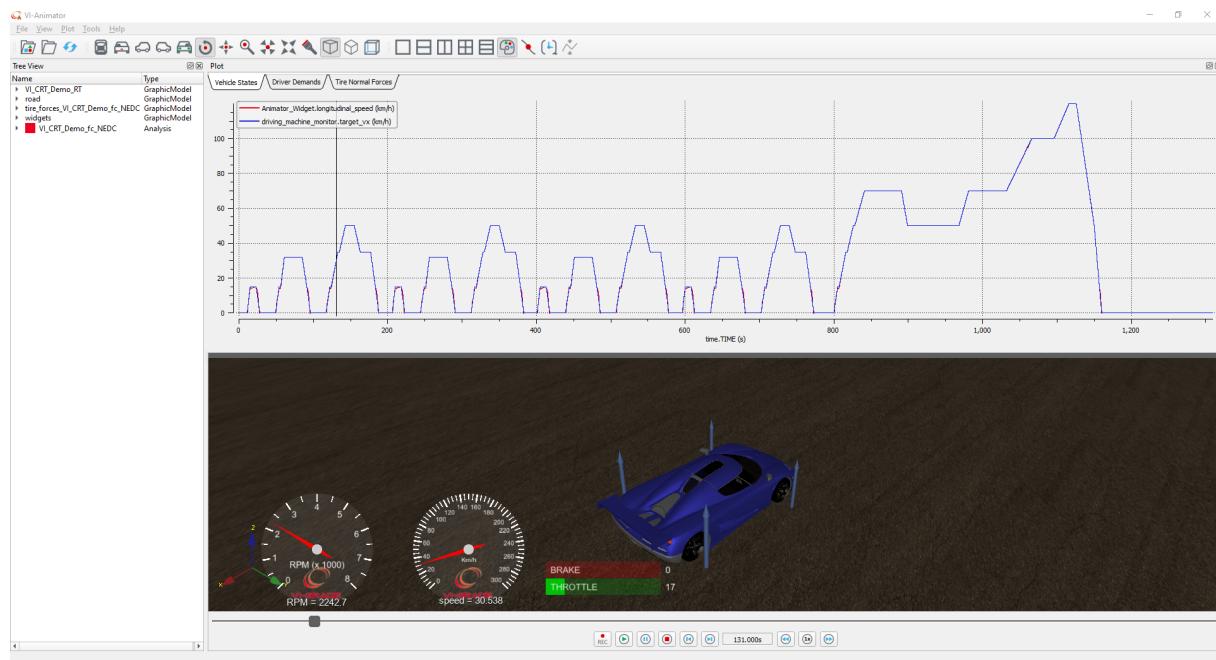
Switch to Review mode using the  button.

Select the VI_CRT_Demo_fc_NEDC analysis as shown below using the left mouse button and then click on the  button in the toolbar:



VI-Animator will start and load the required simulation results and animations.

VI-CarRealTime

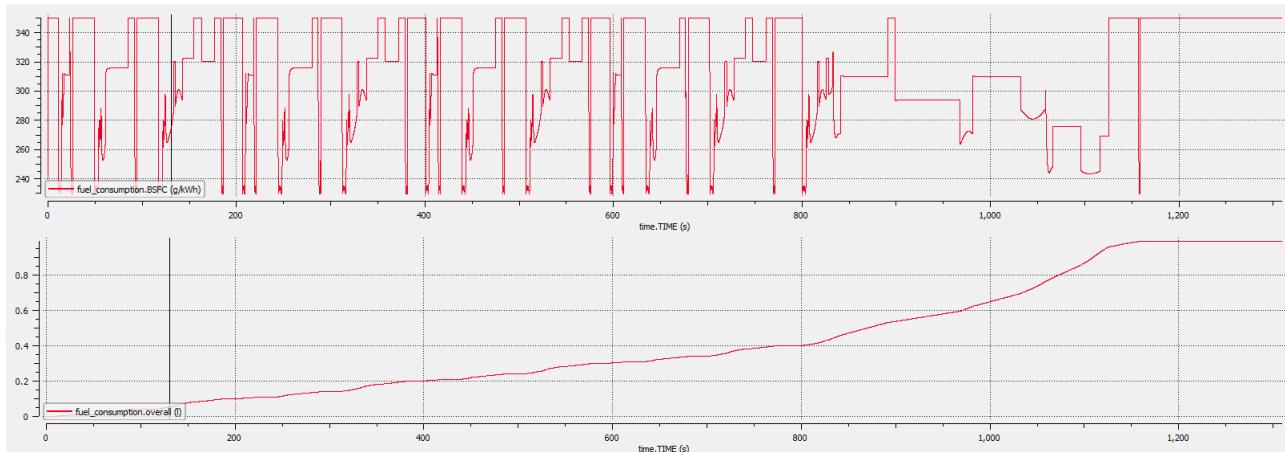


Now, let's add a new page to plot the main outputs for fuel consumption:

- right-click on the Tire Normal Forces plot tab and select **Insert After** option;
- double-click on the new tab to rename it;
- change the tab name to *Fuel Consumption* and press enter on the keyboard;
- modify the page layout by selecting **2x1** () plots. A dialog box may appear asking if you are sure that you want to switch to this **2x1** plot configuration; in that particular case, click on **Yes**.

Plot the following outputs (as a function of TIME, which is the default independent variable) in the two plots by right-clicking on the plot and selecting **Curves**. The two plots must have dependant variables as described below:

- plot 1: `fuel_consumption.BSFC`
- plot 2: `fuel_consumption.overall`



The `fuel_consumption.BSFC` output is the [Brake Specific Fuel Consumption](#), and represents the rate of fuel consumption divided by the power produced (g/kWh).

The `fuel_consumption.overall` output represents the global fuel consumption.

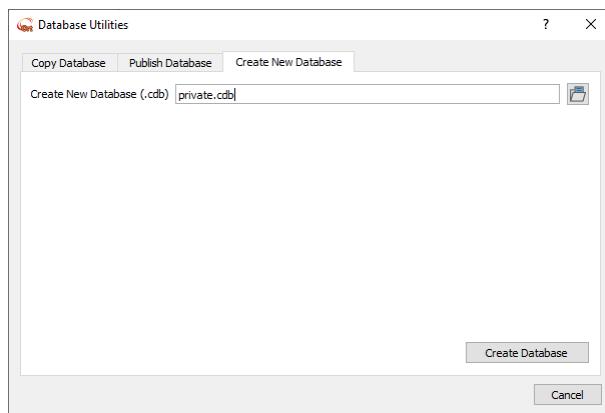
Looking at the results illustrated in the plots above, we can see that the particular vehicle analyzed (with corresponding Fuel Consumption Map values) consumes approximately 1 liters under NEDC.

Using Vehicle Wizard

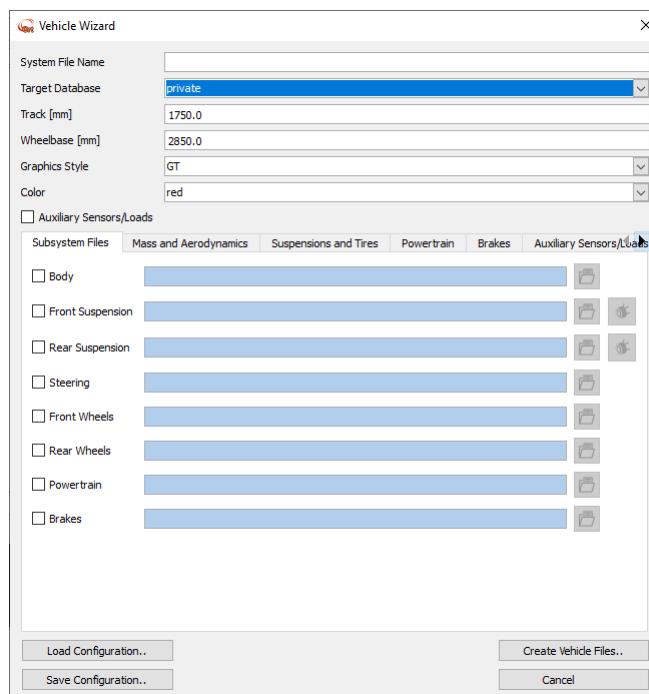
The following tutorial shows how to use the VI-CarRealTime Vehicle Wizard.

Start a new VI-CarRealTime session.

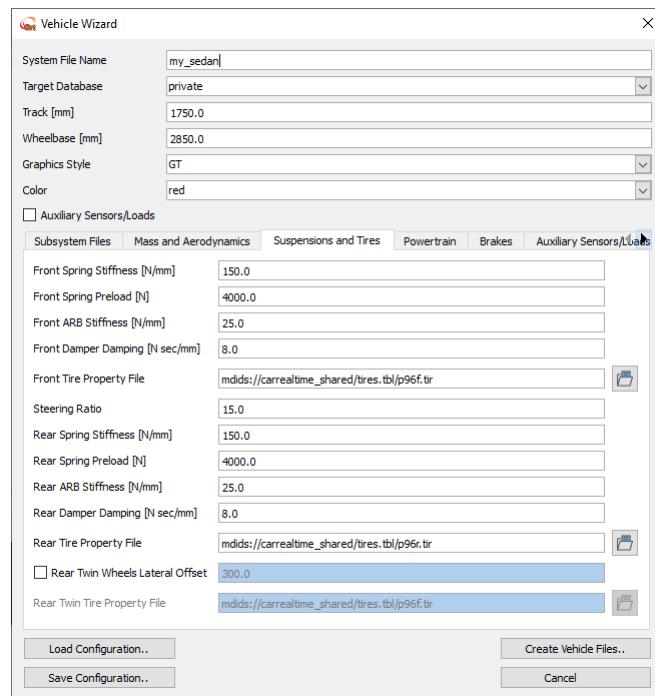
Create a new database; go to *Utilities -> Database Utilities* , and click on the **Create New Database** tab. Create a new database according to the picture below.



Open the Vehicle Wizard from the VI-CarRealTime Utilities menu:

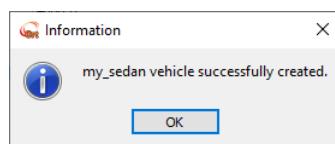


Fill the System File Name field with the name **my_sedan**, and then select *private* as target database. In the Suspensions and Tires tab, set the *mdids://carrealtime_shared/tires.tbl/p96f.tir* file from the *carrealtime_shared* database as the Rear Tire property file, as shown below.



Click on the **Create Vehicle Files..** button.

At the end of the creation process you will get the following message:



Before closing the Vehicle Wizard, please **remember to save your new *my_sedan* vehicle** to the *private.cdb* database you created earlier using the **Save Configuration..** button in the bottom left corner of the window.

This vehicle model created can now edited as needed, loaded into a given session and can thus be used to run simulations and analyses.

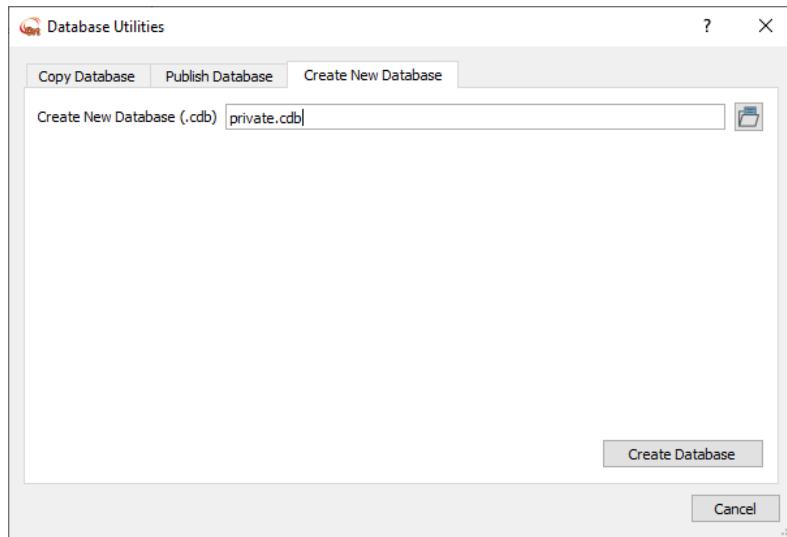


Using K&C Wizard

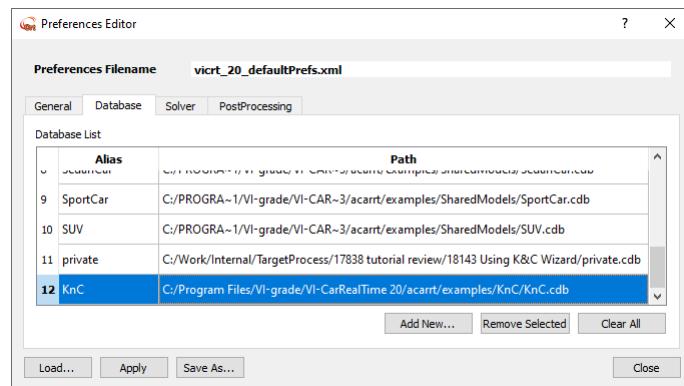
The following tutorial will show how to use the VI-CarRealTime K&C Wizard.

Start a new VI-CarRealTime session.

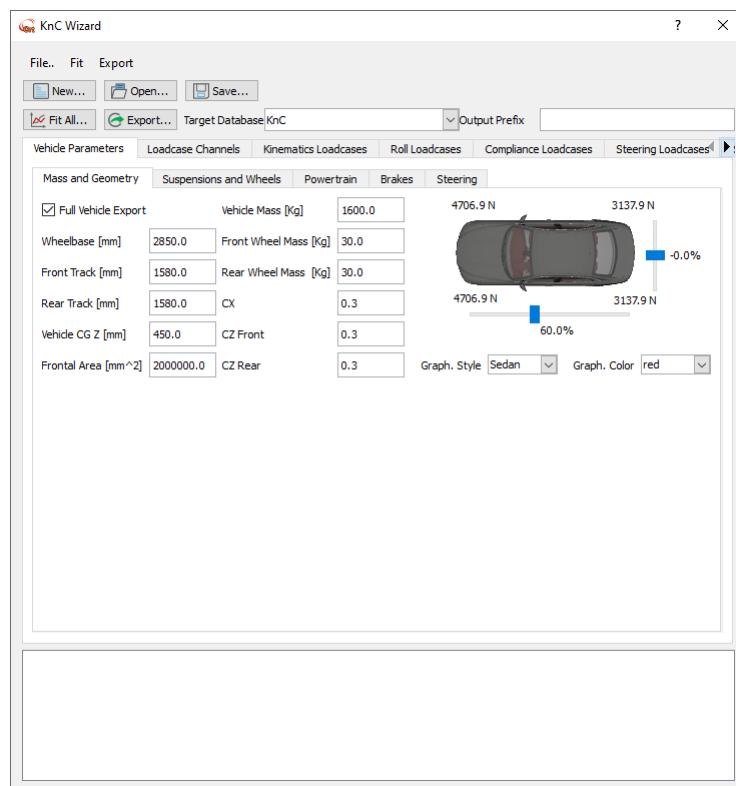
Note: Create a new database by using the *Utilities -> Database Utilities* menu in the working directory (name it **private.cdb**).



Register KnC example database into the VI-CarRealTime session, as shown in previous tutorial exercises (Click on the **Add New...** button, navigate to the `<VICRT_Install_dir>\lacart\examples\KnC` directory, and then select the **KnC.cdb** folder, and press **Apply** in the Preferences Editor).

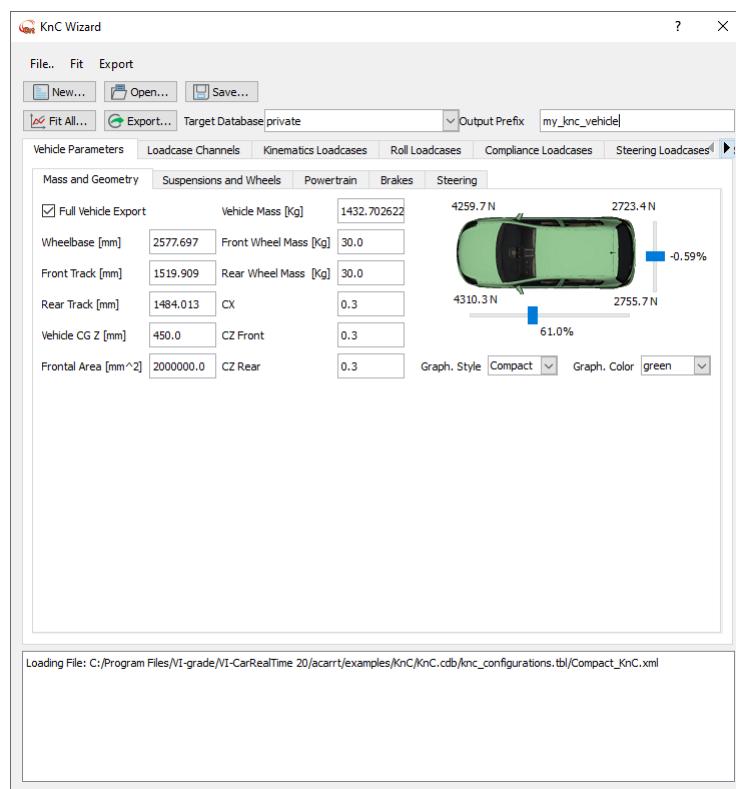


Open the K&C Wizard from the VI-CarRealTime Utilities menu:



Open example configuration file included in KnC database, mdids://KnC/knc_configurations.tbl/Compact_KnC.xml, and set up parameters as follows:

- Output Prefix: **my_knc_vehicle**
- Target Database: **private**



Click the **Export...** button.

The vehicle model just created can now be edited as needed, loaded into a given session and can thus be used to run simulations and analyses.

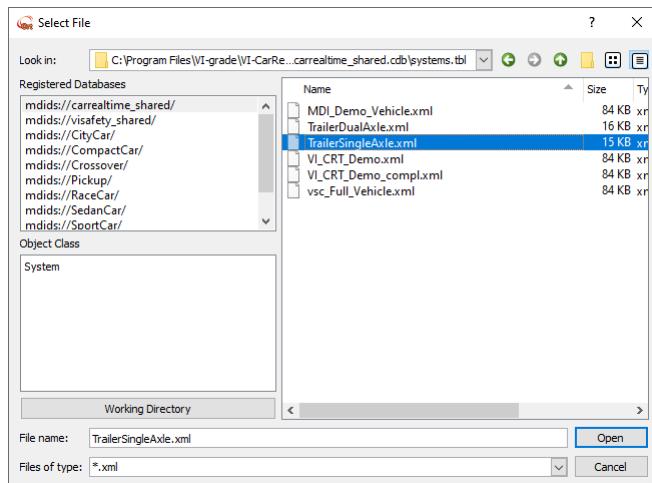


Running a Vehicle-Trailer analysis

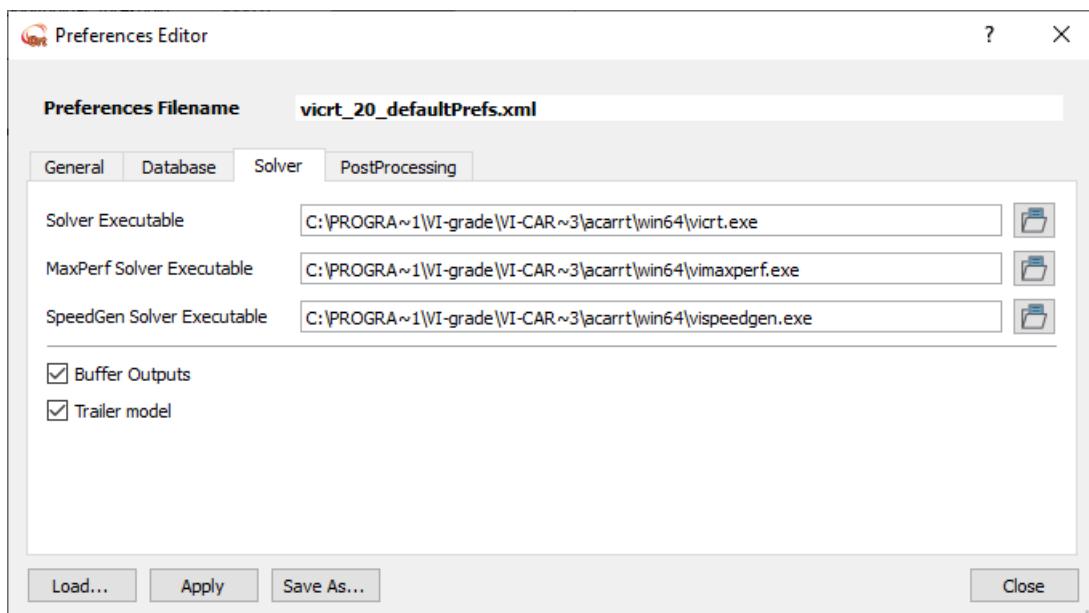
To run a trailer simulation, you will need a vehicle and a trailer model.

Start a new VI-CarRealTime session using the **vicrt20** command from a DOS (cmd) shell or through the shortcut in the Windows Start Menu.

In **Build Mode**, open the **TrailerSingleAxle.xml** model from the **carrealtime_shared** database. Open the **MDI_Demo_Vehicle.xml** model from the same directory as well.



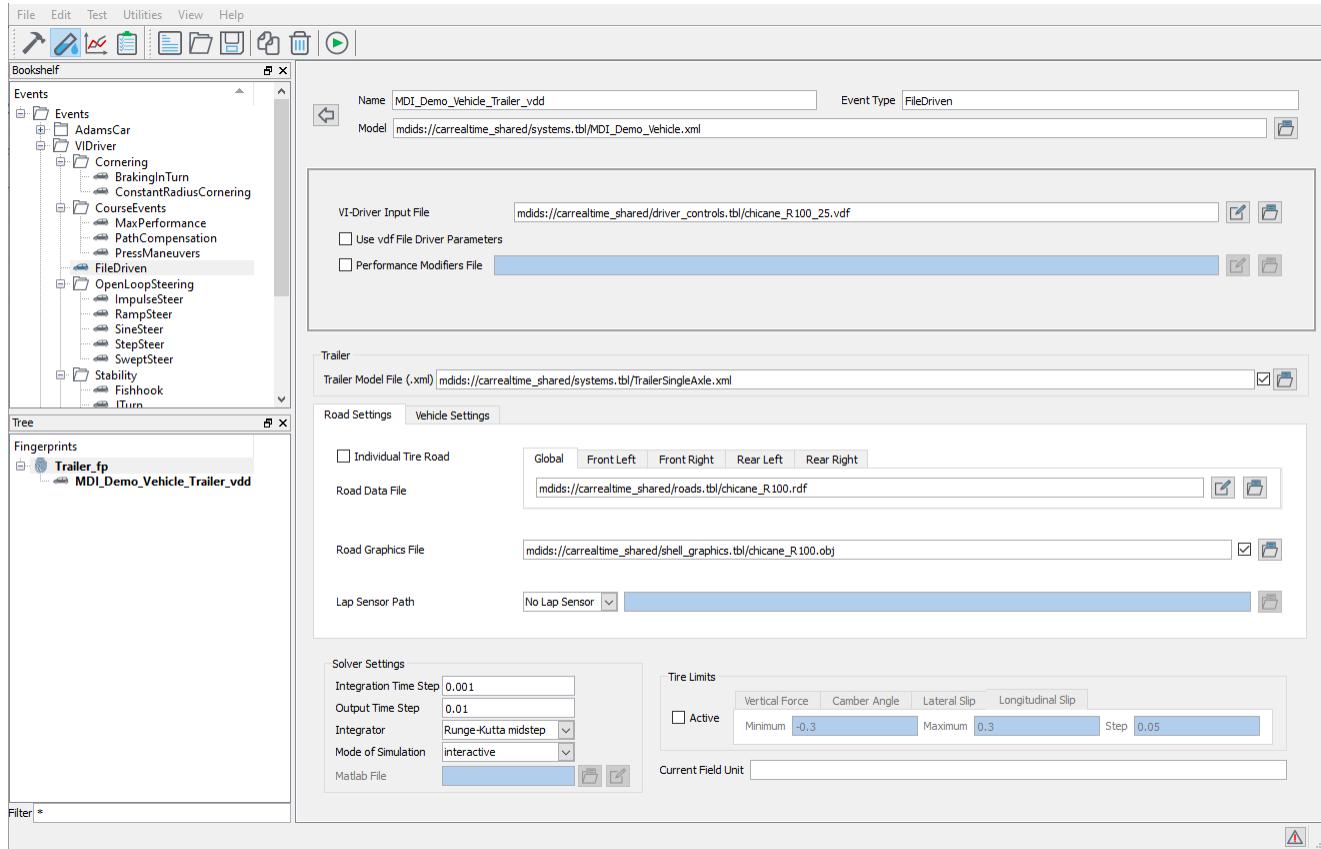
Open the Preferences window from the Edit menu and under the **Solver** tab check the **Trailer model** check-box.



Switch to **Test Mode** using the button, and follow the steps below:

- Create a new fingerprint, and name it **Trailer_fp**
- Add a new **FileDriven** event to fingerprint from the **MDI_Demo_Vehicle** model and select the event file named **chicane_R100_25.vdf** from the **carrealtime_shared** database;
- Rename the event **MDI_Demo_Vehicle_Trailer_vdd**;
- Check the **Trailer Model File** checkbox and select the **TrailerSingleAxle.xml** model file from the **carrealtime_shared** database;

- Set **chicane_R100.rdf** as **Road Data File**;
- Check the **Road Graphics File** checkbox and select the **chicane_R100.obj** graphics model file from the *carrealtime_shared* database;
- Run the simulation by clicking on the  button.



Now we will run the same simulation but this time we will first change the following trailer properties in order to simulate a load on the trailer:

1. the z coordinate of CG
2. the mass

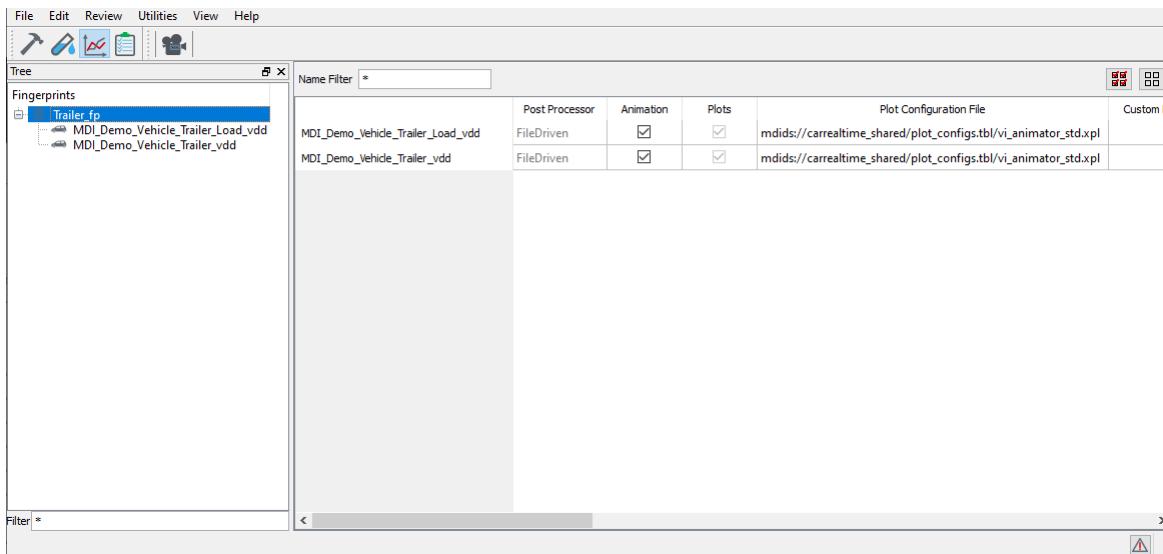
Switch back to **Build Mode**, and follow the steps below:

- Select the **RT_TrailerSingleAxle_body** subsystem under the **TrailerSingleAxle** model, and go to the **Sprung Mass** tab.
- Set the **CG height** field to 530 (mm) and set the **Mass** field to 260 (kg) as shown in the picture below.

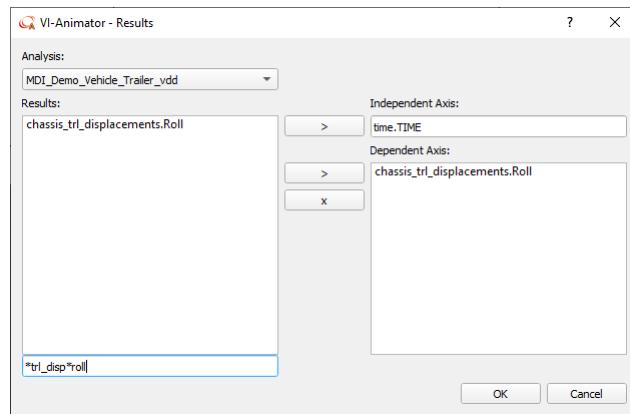
Geometry	
X	Y
Hitch Ball Location	-3500.0
Hitch Ball to Axle Longitudinal Offset	0.0
Z	260.0
Mass and Inertia	
CG longitudinal hitch distance	375.0
CG lateral position	0.0
CG height	530
Mass	260
Ixx	53960000.0
Iyy	252700000.0
Izz	254600000.0
Ixy	19760000.0
Ixz	-64410000.0
Iyz	-6137000.0
Cargo	
Cargo longitudinal position	1.0
Cargo longitudinal dimension	1.0
Cargo lateral dimension	1.0
Cargo minimum vertical position	1.0

Note: Do not save these changes in the trailer model to the `carrealtime_shared` database; save it to another database so as to not alter the baseline/default trailer model.

- Switch to **Test Mode** using the  button;
- Right click on the **MDI_Demo_Vehicle_Trailer_vdd** event and select **Copy Selected Event**;
- Rename the newly created event: **MDI_Demo_Vehicle_Trailer_Load_vdd**;
- **Run** this event by clicking on the  button;
- Switch to **Review mode** using the  button;
- Select both the **MDI_Demo_Vehicle_Trailer_vdd** and **MDI_Demo_Vehicle_Trailer_Load_vdd** analyses;



- Now, click on the  button in **Review Mode**. A VI-Animator window will open showing the two analyses together.
- Right click on the **Tire Normal Forces** tab and select **Insert After**. This operation will add a new page with empty plots.
- Select the  button to change the current layout to 1 plot per page.
- Right click anywhere on the plot, and select **Curves**.
- Type the string ***trl_disp*roll** into the text box in the bottom left corner to filter the Results.
- Select **chassis_trl_displacements.Roll** and add it to "Dependent Axis".



Press the **OK** button. Finally, you should see the following plots along with the corresponding animation, which compare the behavior of vehicles with different loads in their respective trailers.

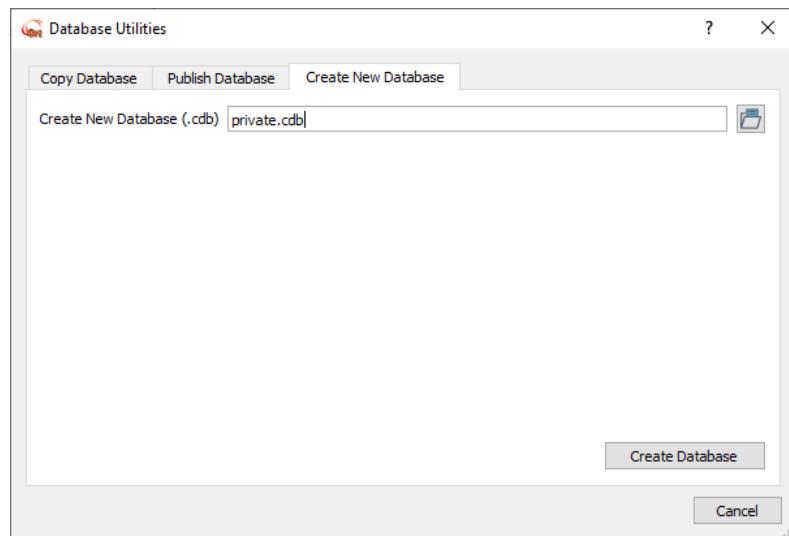


Using CarMaker converter

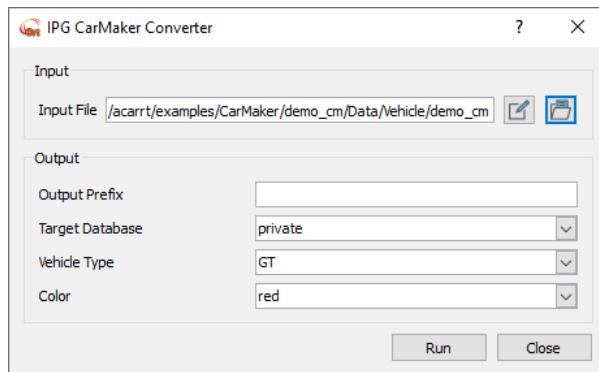
The following tutorial will show how to use the VI-CarRealTime CarMaker converter.

Start a new VI-CarRealTime session.

Note: Create a new database by using the *Utilities -> Database Utilities* menu in the working directory (name it **private.cdb**).

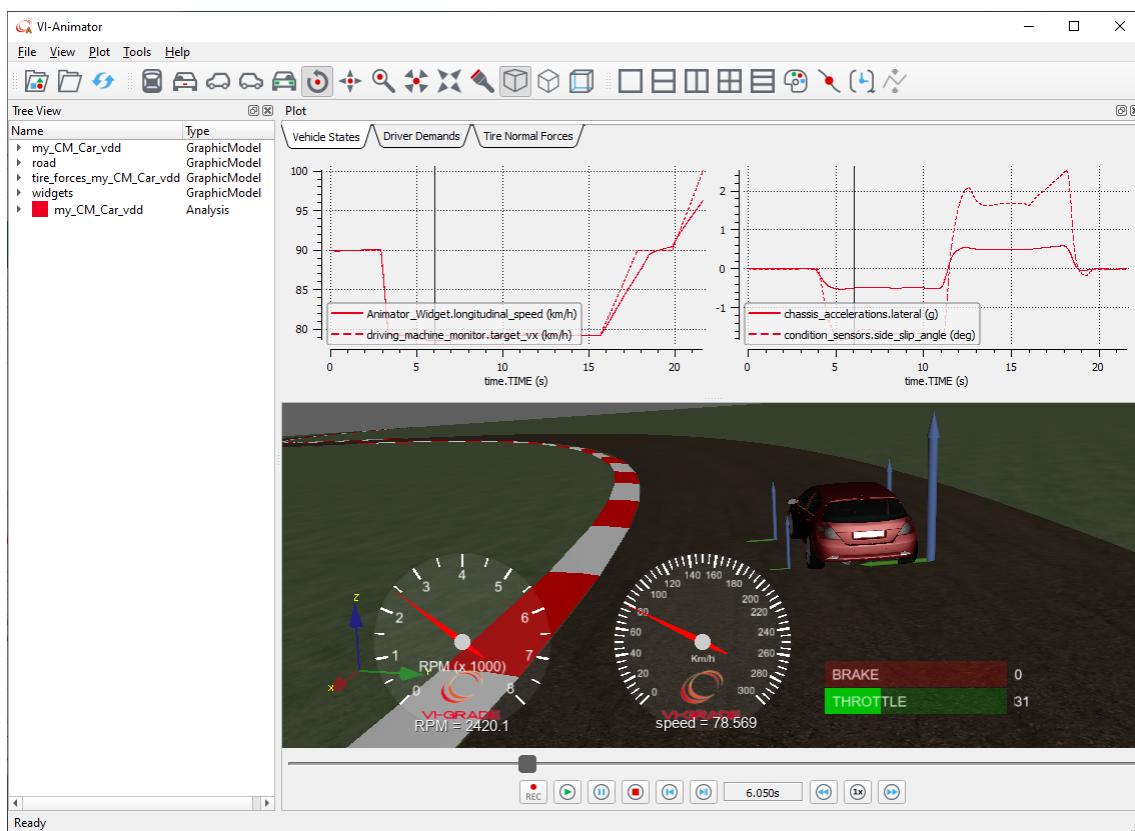


Open the CarMaker converter from the VI-CarRealTime Utilities menu, select a CarMaker-Car file as **Input File** (an example file can be found in your VI-CarRealTime installation directory in \acarrt\examples\CarMaker\demo_cm\Data\Vehicle). Then set **Output Prefix** as **my_CM_Car** and **Target Database** as **private**.



Click the **Run** button.

The vehicle model just created can now be edited as needed, loaded into a given session and can thus be used to run simulations and analyses.



Simulating a DRS System

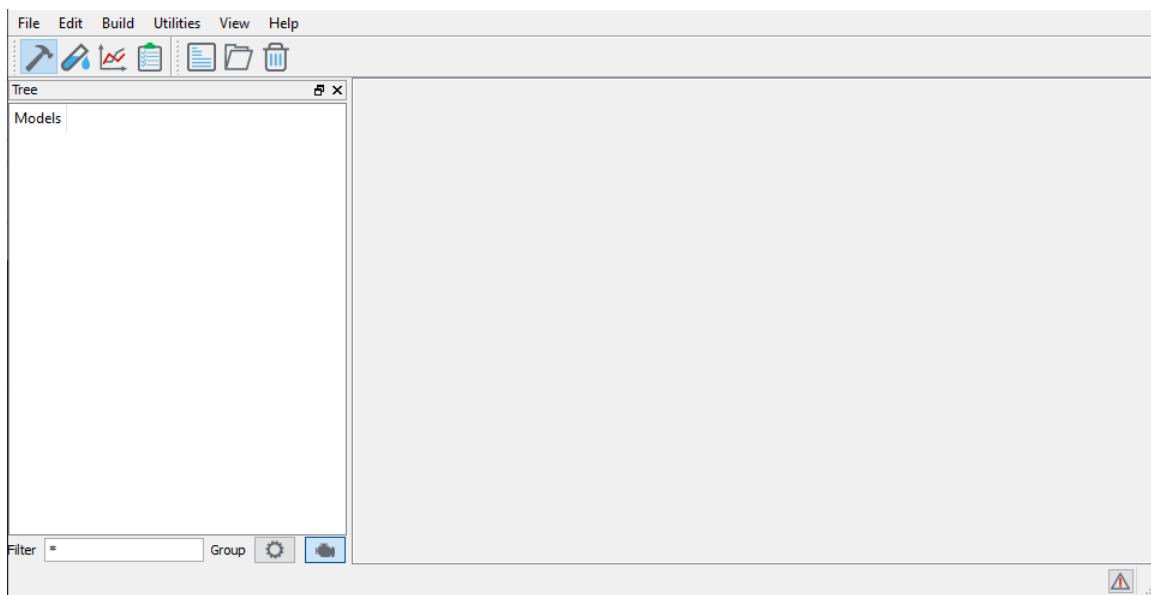
The aim of the tutorial is to instruct the user about how to manage [PMF](#) files in a MaxPerformance simulation. PMF allows to define a map of scaling factors for aerodynamic forces (down forces and drag force) as a function of path_s, such allowing to model the effect of an active aerodynamic system (DRS).

The tutorial is composed by the following sections:

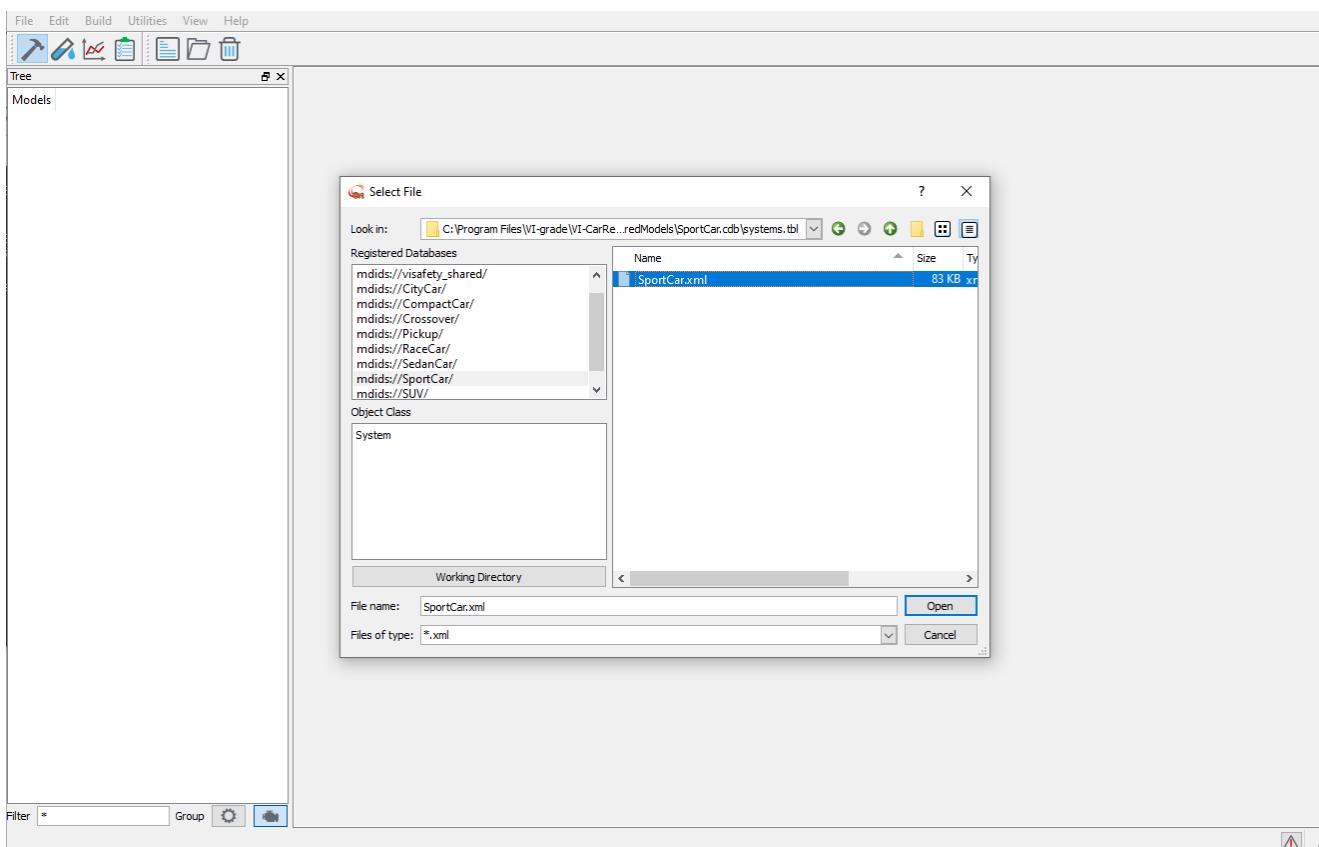
- [Model set-up](#)
- [Run MaxPerformance analyses](#)
- [Review results](#)

Model Set-up

Start a new VI-CarRealTime session from the Windows Start Menu or by typing **vicrt20** from a DOS (cmd) shell.



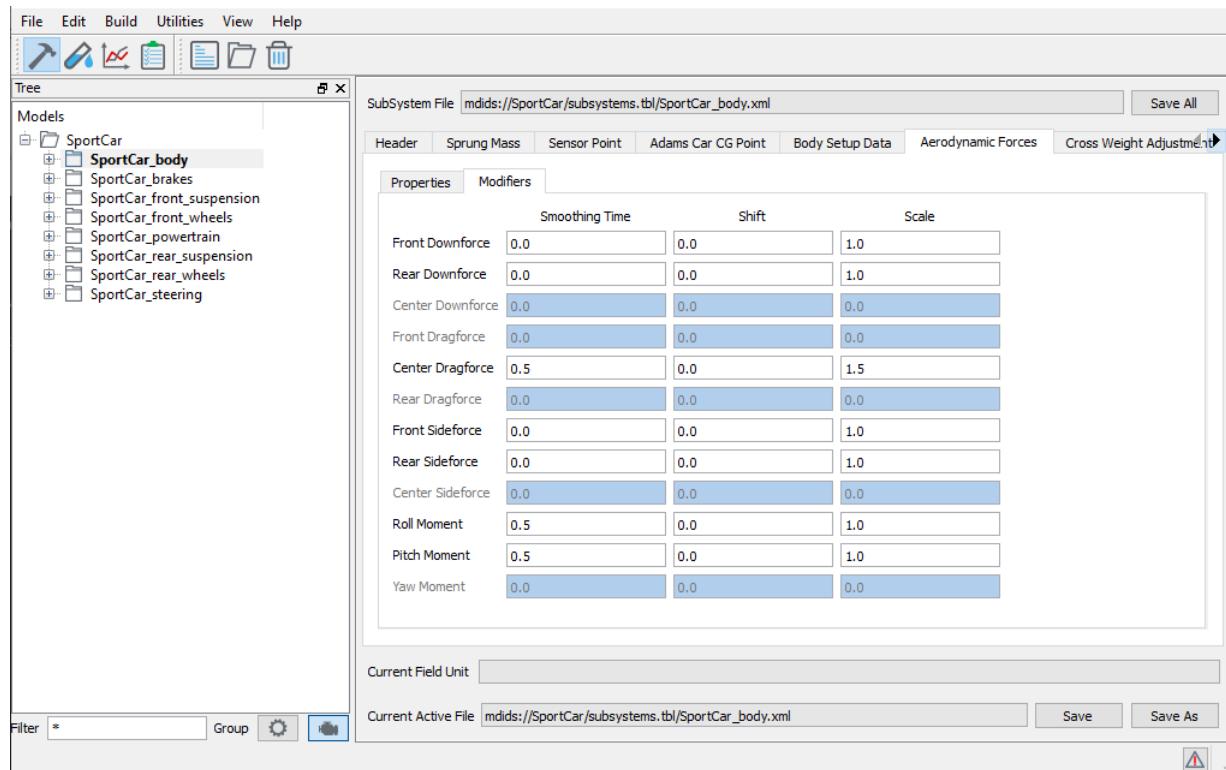
In **Build mode**, go to Build -> Load Model, and load the **SportCar.xml** vehicle model from the **SportCar** database.



The next step consists in modifying the scaling factor for the aerodynamic drag force.
To do it:

- in **Build Mode**, browse to **Body** subsystem;

- select **Aerodynamic Forces** panel;
- click on **Modifiers** container;
- set **Center Dragforce Scale** coefficient to **1.5** as shown in the picture below.



Run MaxPerformance Analyses

In this section we will move to Test Mode and we will run two MaxPerformance events:

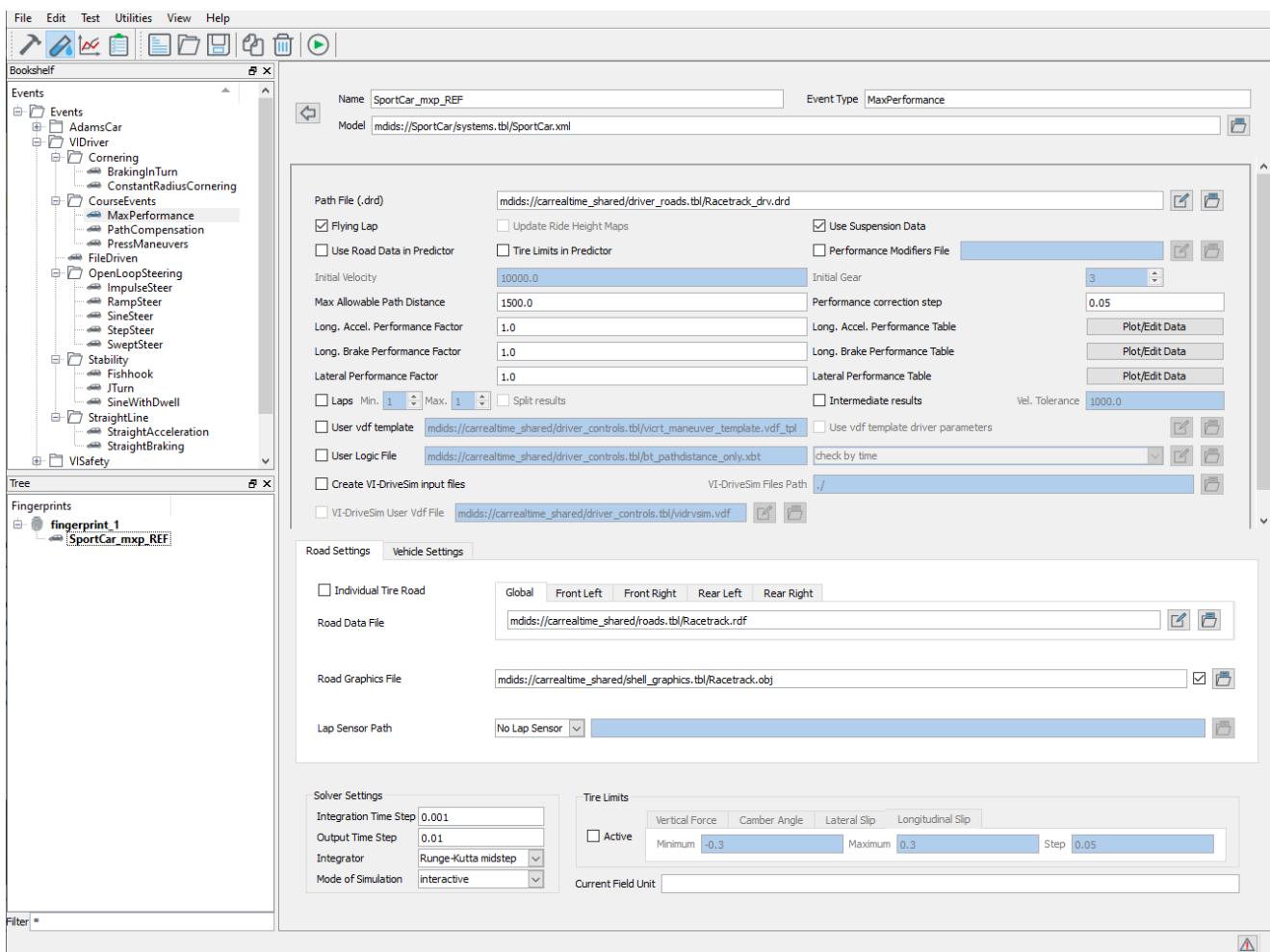
1. the first one using standard aerodynamic scaling factor
2. the second one using a [PMF](#) file to simulate the effect of a DRS system

Move to **Test Mode** and add a **MaxPerformance** event.

Set the following parameters for the event:

- change the event name to **SportCar_mxp_REF**;
- Path File to **mdids://carrealtime_shared/developer_roads.tbl/Racetrack_drv.drd**;
- check **Flying Lap** toggle button;
- check **Use Suspension Data** toggle button to enable VI-SpeedGenEvo;
- set Road Data File to **mdids://carrealtime_shared/roads.tbl/Racetrack.rdf**;
- set Road Graphics File to **mdids://carrealtime_shared/shell_graphics.tbl/Racetrack.obj**.

VI-CarRealTime

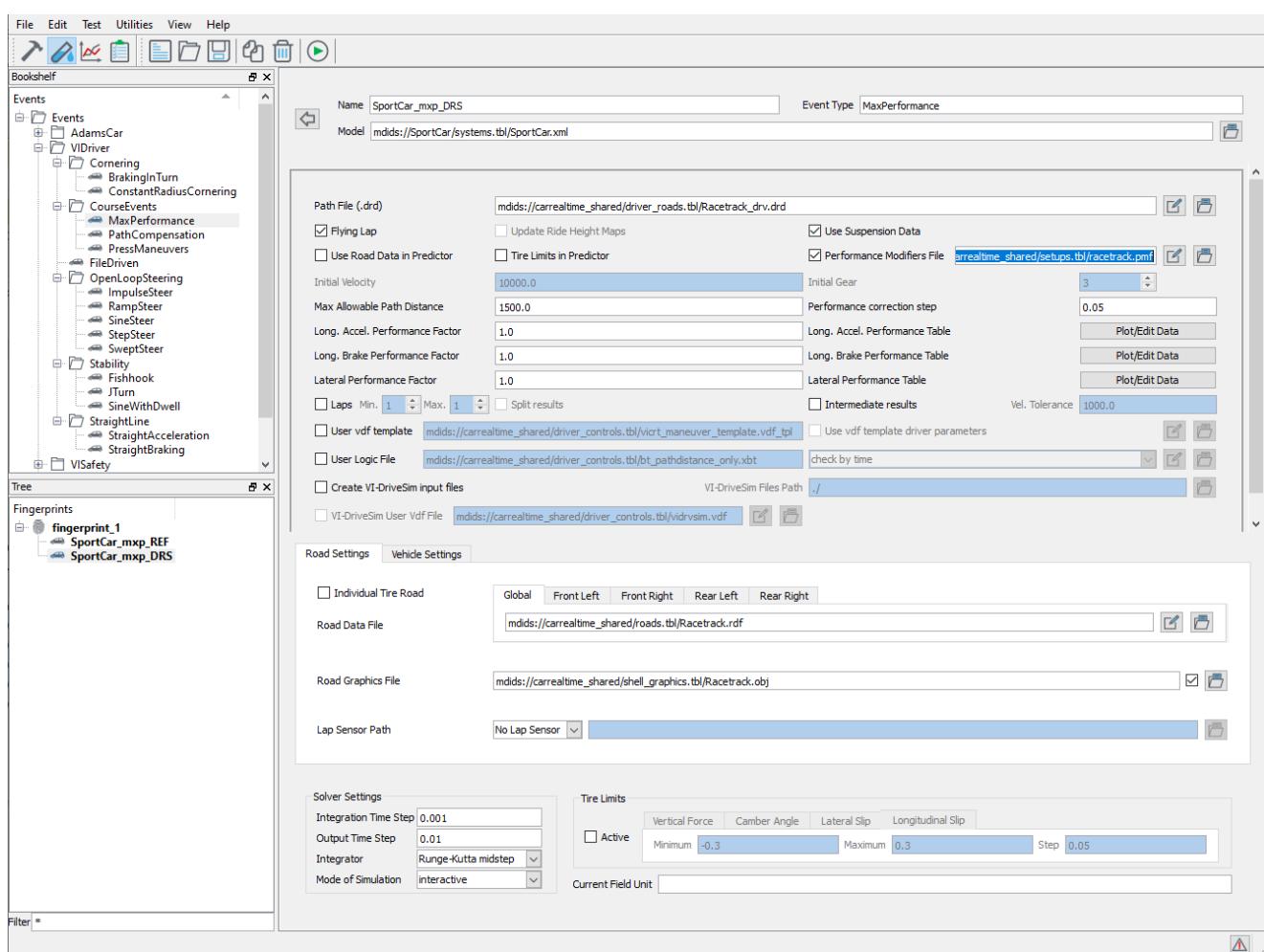


Press button in the toolbar to run the MaxPerformance event.
Wait until the simulation is completed.

Now the next step consists in running another MaxPerformance event with the same parameter as the one just run, except for the Performance Modifiers File.

In particular:

- copy the MaxPerformance event and rename it to **SportCar_mxp_DRS**;
- check **Performance Modifiers File** toggle button and select **mdids://carrealtime_shared/setupstbl/racetrack.pmf** file.



The PMF file scales the aerodynamic forces in specific sections of the track, simulating the DRS effect:

```
[AERODYNAMIC_SCALE_MAP]
{ path_s    air_drag_scale    air_front_down_scale    air_rear_down_scale}
0          .9                  .97                   .9
50         .9                  .97                   .9
100        .9                  .97                   .9
410        .9                  .97                   .9
440        1                  1                     1
550        1                  1                     1
3460       1                  1                     1
3480       .9                 .97                   .9
3790       .9                 .97                   .9
3810       1                  1                     1
4310       1                  1                     1
4330       .9                 .97                   .9
4500       .9                 .97                   .9
```

Press again button in the toolbar to run the variant MaxPerformance event.
Wait until the simulation is completed.

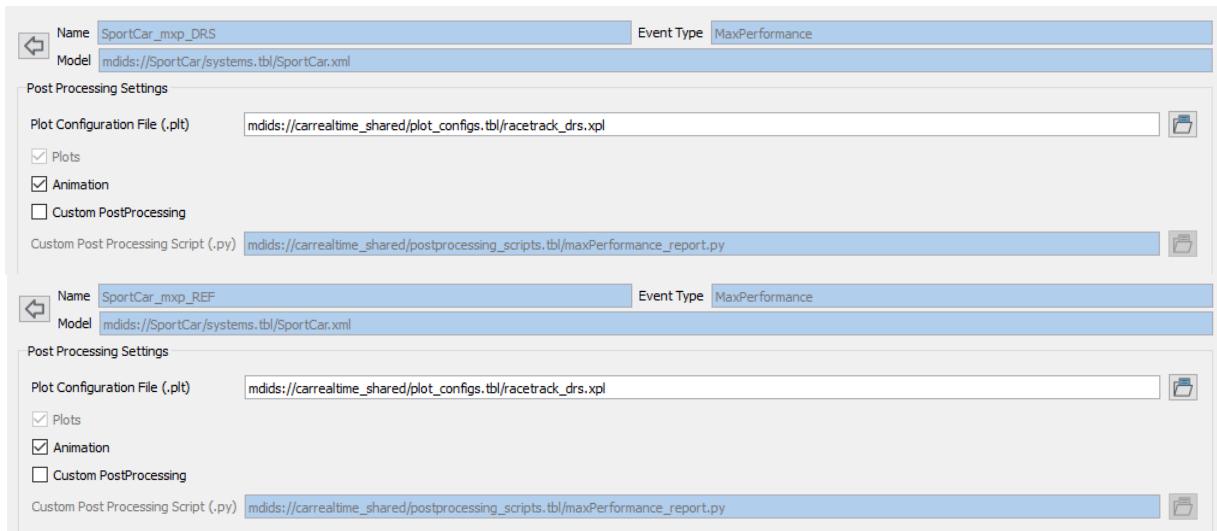
Review Results

In this section we will review the results of the MaxPerformance simulations.

Browse to **Review Mode** .

VI-CarRealTime

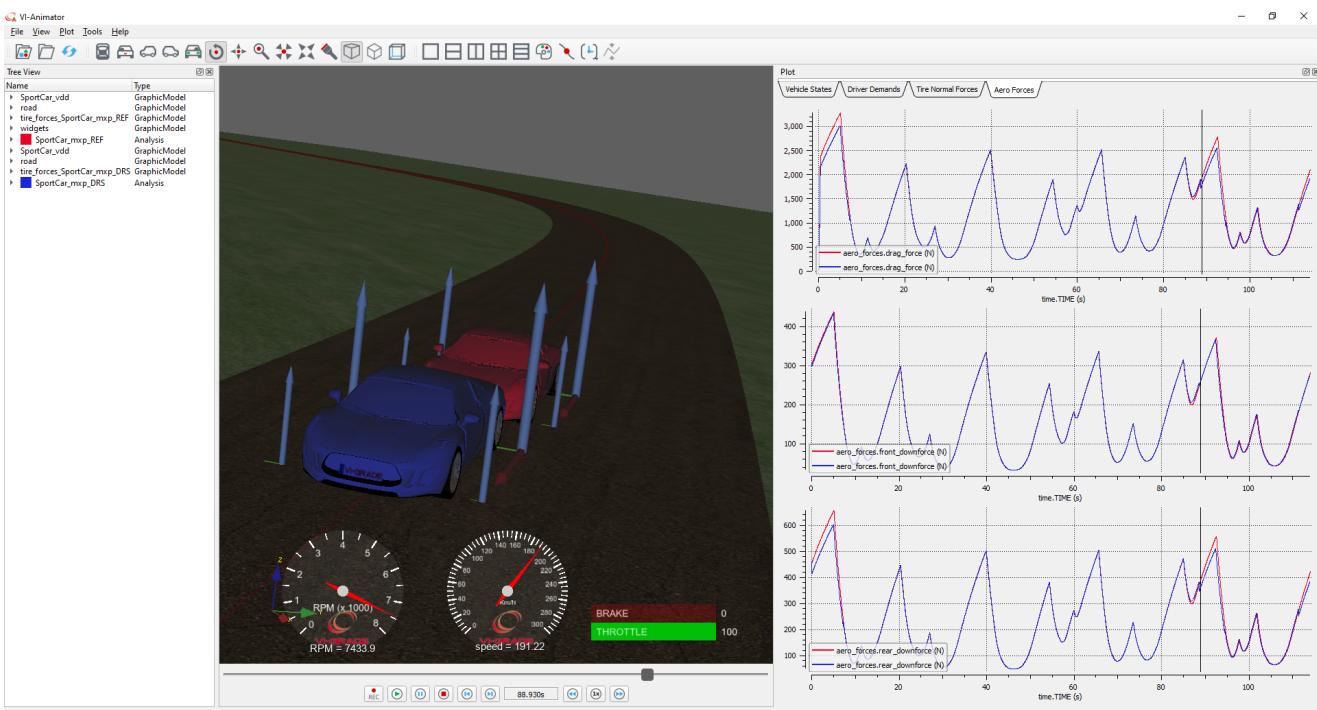
Select `mdids://carrealtime_shared/plot_configs.tbl/racetrack_drs.xpl` in **Plot Configuration File** field for both events as shown in the picture below:



Then select both the events and press button in the toolbar to launch VI-Animator.



Review results and plot from VI-Animator:



From **Aero Forces** plot page it's possible to notice the effect of the performance factor modifiers on the aerodynamic forces (front downforce, rear downforce, drag force) of the vehicle.

In particular, the blue vehicle (*SportCar_mxp_DRS*) is faster than the red one (*SportCar_mxp_REF*) because of the reduction of drag force (and down-force) in certain sections of the track.

Running Investigation

The tutorial shows the user how to use the [Investigation Mode](#) in VI-CarRealTime.

The aim of the tutorial is to improve the vehicle lateral performance by changing its anti-roll behavior. In particular, front and rear [Auxiliary Roll Forces](#) will be scaled in order to simulate the effect of changing the ARB stiffness.

A first [MaxPerformance](#) analysis will be run with the base vehicle.

The investigation analysis will be performed using a [Ramp Steer](#) event as reference event; the responses carried out from such event will be analyzed in order to find out the vehicle set up that exalts the vehicle lateral performance.

The set up will be applied to the vehicle and used to run a second [MaxPerformance](#) simulation.

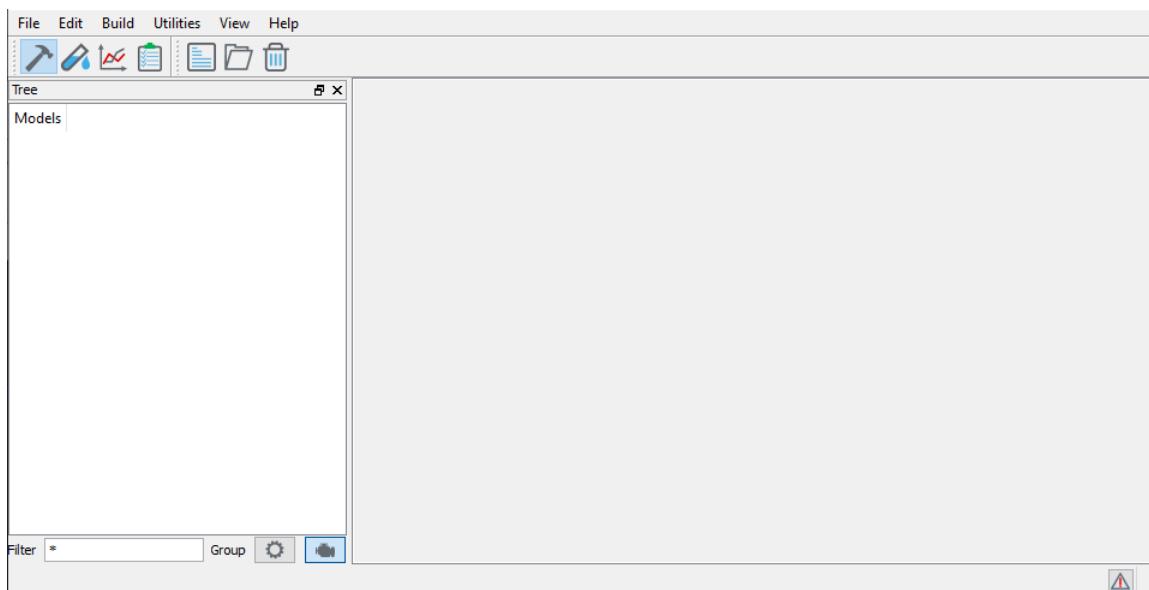
Finally the results will be compared to see the differences between the baseline vehicle and the optimized one.

The tutorial consists of the following sections:

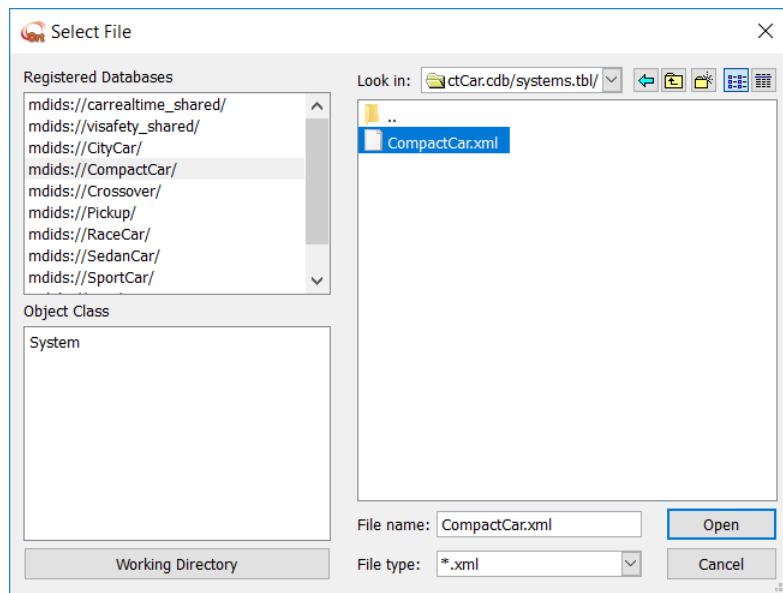
- [Run MaxPerformance with base model](#)
- [Set up Ramp Steer event](#)
- [Set up the Investigation](#)
- [Run the Investigation](#)
- [Analyze Investigation Results](#)
- [Run MaxPerformance with Optimized Model](#)
- [Compare the Results](#)

Run MaxPerformance with Base Model

Start a new VI-CarRealTime session from the Windows Start Menu or by typing **vicrt20** from a DOS (cmd) shell.

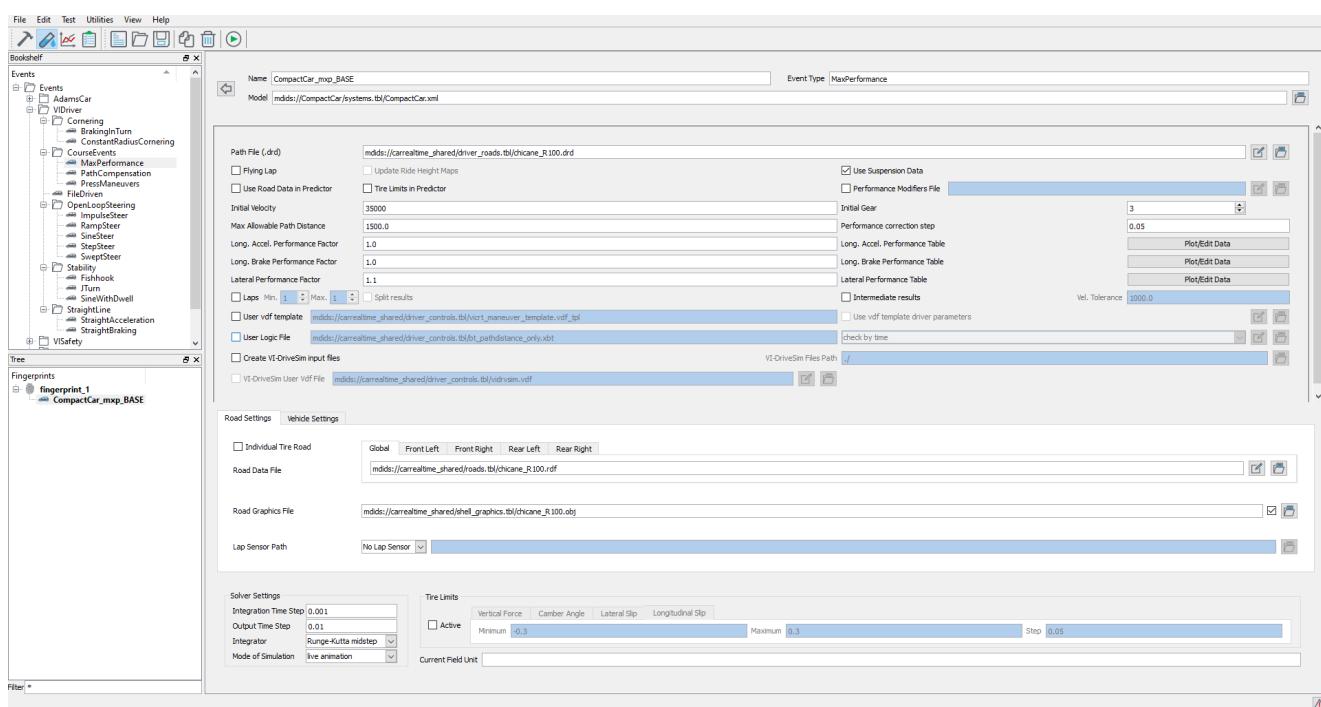


In **Build mode**, go to Build -> Load Model, and load the **CompactCar.xml** vehicle model from the *CompactCar* database.



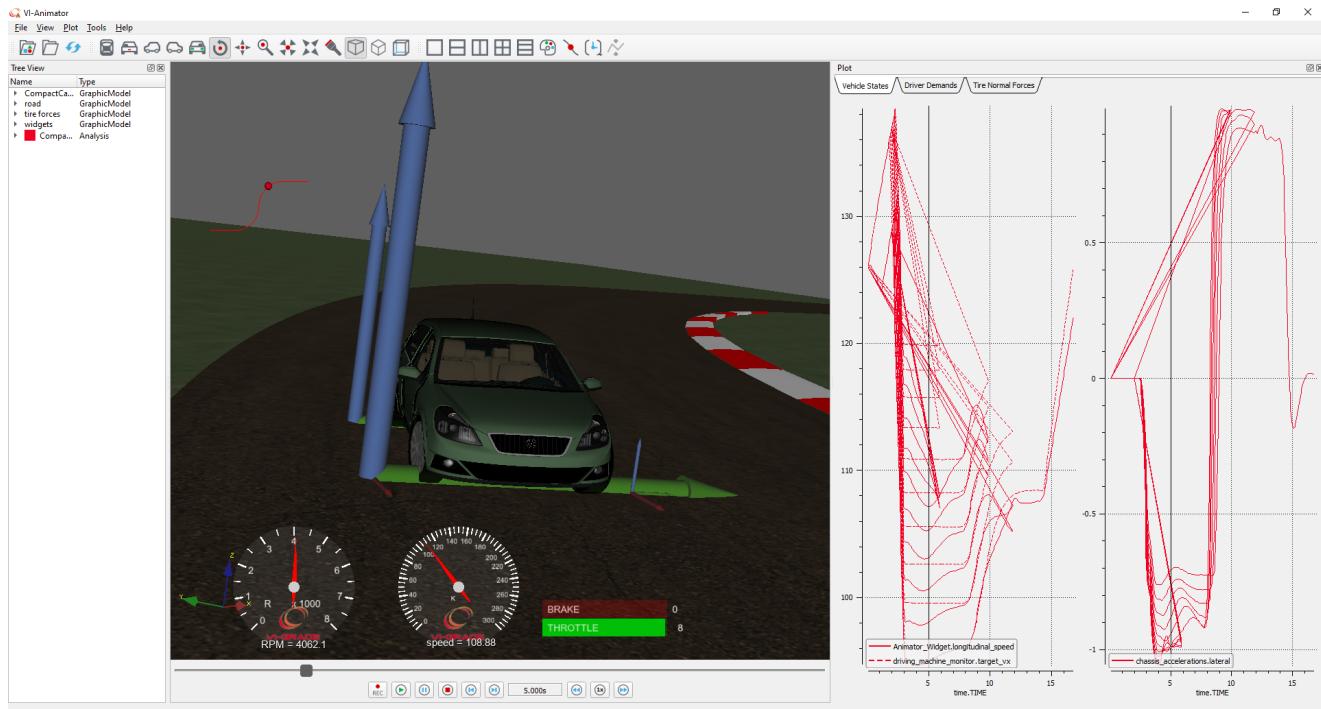
Move to **Test Mode** and add a **MaxPerformance** event using **CompactCar** model.
Set the following parameters for the event:

- change the event name to **CompactCar_mxp_BASE**;
- Path File to **mdids://carrealtime_shared/driver_roads.tbl/chicane_R100.drd**;
- check **Use Suspension Data** toggle button to enable VI-SpeedGenEvo;
- set **Initial Velocity** to **35000** (35 m/s);
- set **Lateral Performance Factor** to **1.1**;
- set **Road Data File** to **mdids://carrealtime_shared/roads.tbl/chicane_R100.rdf**;
- set **Road Graphics File** to **mdids://carrealtime_shared/shell_graphics.tbl/chicane_R100.obj**;
- set **Mode of Simulation** to **Live Animation**.



Press button in the toolbar to run the MaxPerformance event.

Since the mode of simulation is live animation, it's possible to see runtime from VI-Animator the MaxPerformance iterations and understand which are the limits of the vehicle.



Note: looking at the animation and at the plots, it's clear that the vehicle exhibits an understeer behavior.

Set up Ramp Steer event

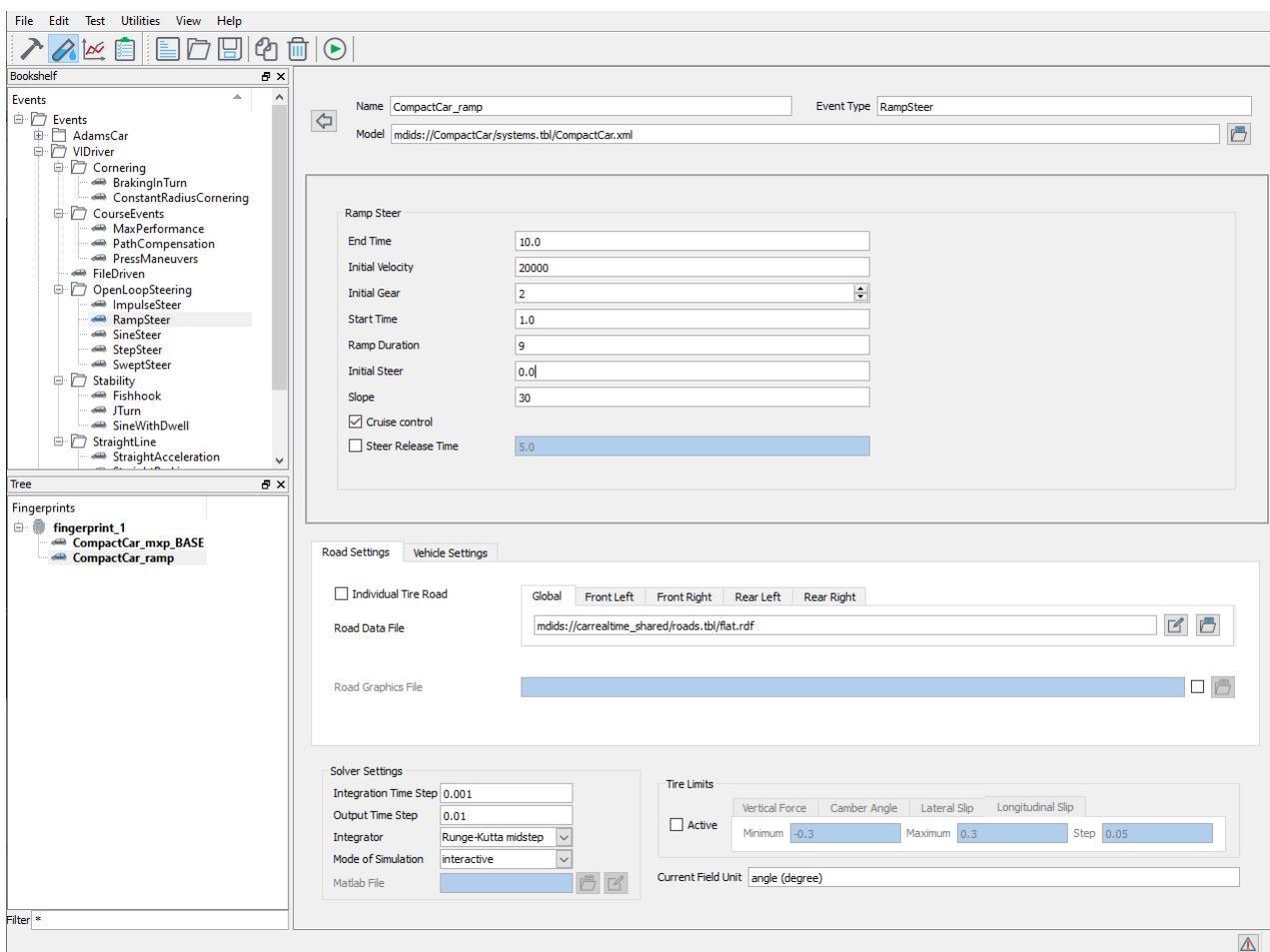
The next step consists in setting up an event which will be used as reference event for the investigation analysis that we're going to run.

Since the aim of the tutorial is to optimize the vehicle performance, and in particular its lateral performance, the goal is to increase its maximum lateral acceleration.

From **Test Mode** add a **RampSteer** event.

Set the following parameters for the event:

- set **Initial Velocity** to **20000** (20 m/s);
- set **Initial Gear** to **2**;
- set **Ramp Duration** to **9.0 s**;
- set **Slope** to **30.0** (deg/s);

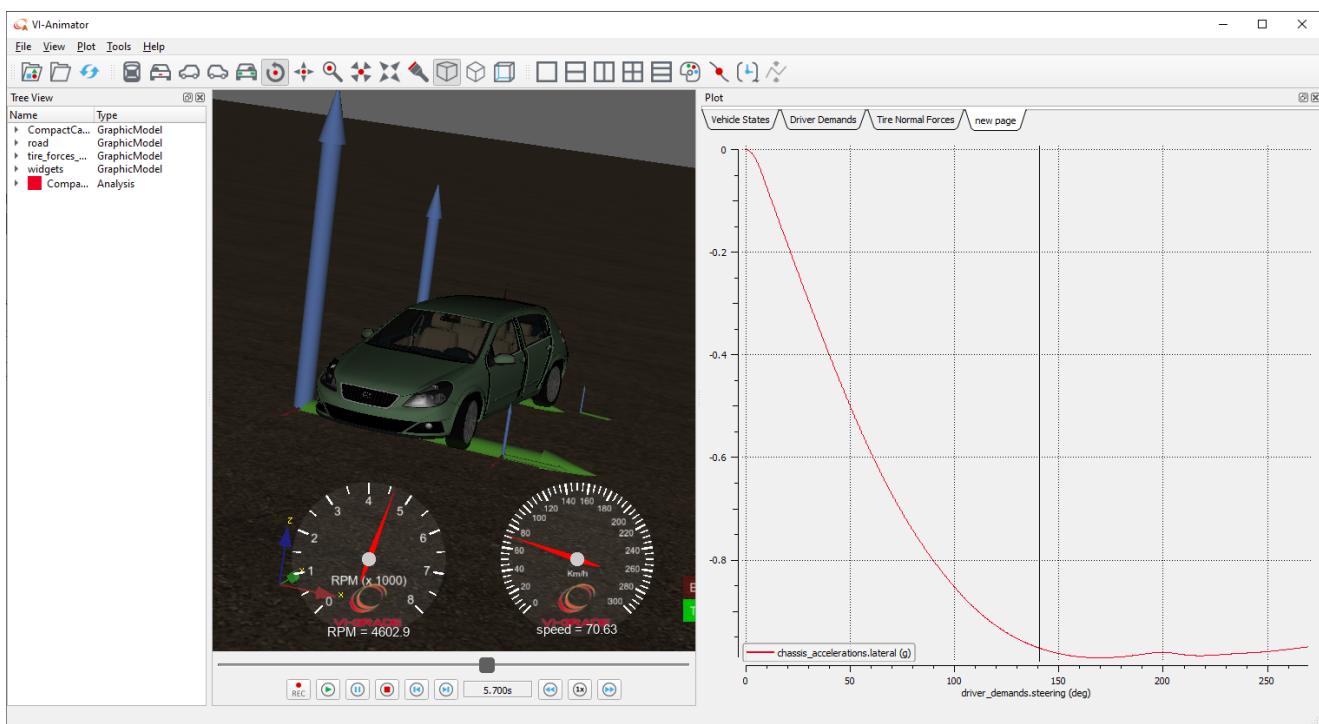


Note: it's not strictly necessary to run the event. We'll use such event in the next section to create the investigation.

Anyway, in order at least to check that the reference event for the investigation has been correctly set-up, it can be run by pressing .

Once the simulation is complete, from **Review Mode** it's possible to launch **VI-Animator** and check the simulation results.

In particular, we can look at the Understeer Gradient:



Set up the Investigation

In this section, we'll set up the investigation.

The investigation will use the RampSteer event as reference event and we'll vary the anti-roll scaling factor coefficients (front and rear) to explore how the vehicle variants behaves in such event.

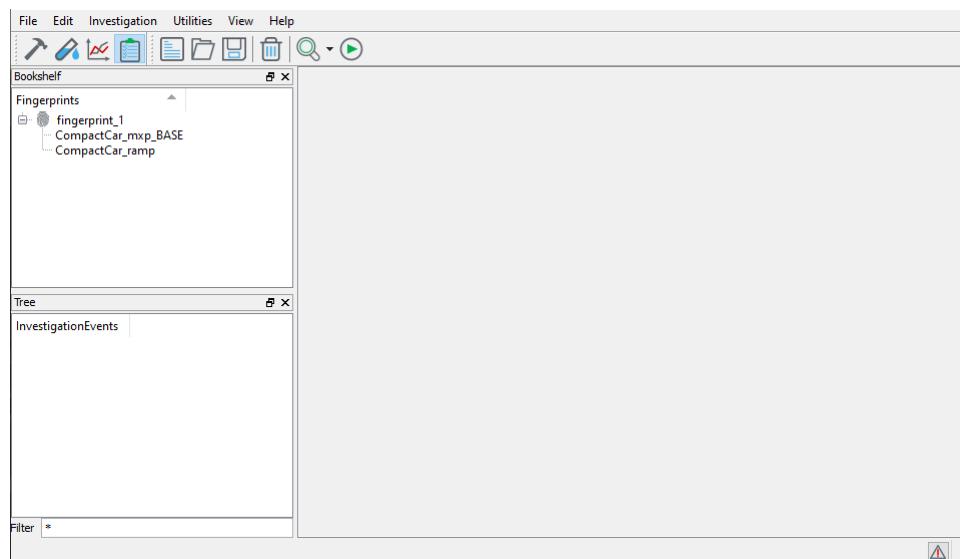
The aim is to find the scaling factors combination which provides the maximum lateral acceleration.

Switch to **Investigation Mode** .

The Bookshelf shows the fingerprint and the events that have been defined in Test Mode.

In particular, the following events are available to be used for an investigation:

1. *CompactCar_mxp_BASE*
2. *CompactCar_ramp*

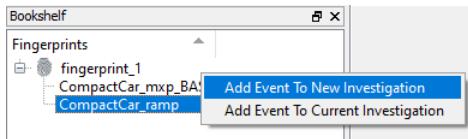


VI-CarRealTime

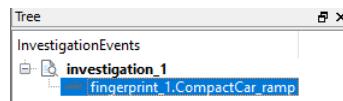
For our purpose, we need only the second one (`CompactCar_ramp`) for the investigation.

To create a New Investigation:

- right-click on **CompactCar_ramp** event and select **Add Event To New Investigation**;



The investigation will be added in the Tree panel:



Now we have to set the investigation parameters. In particular:

- [Factors](#)
- [Responses](#)
- [Investigation Parameters](#)

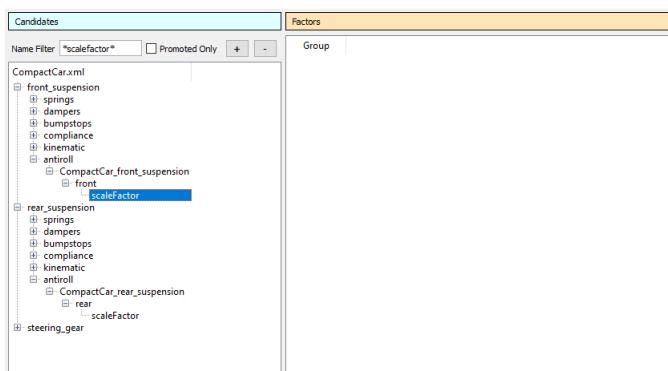
Factors

Since we want to investigate the effect of a changing in the anti-roll bar stiffness on the vehicle lateral performance, we need to promote the following two candidates to factors:

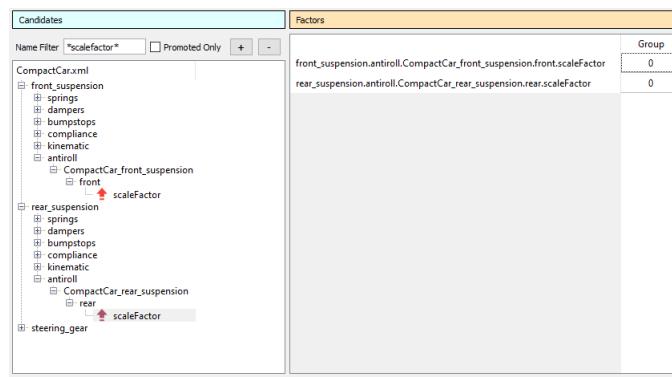
- `front_suspension.antiroll.CompactCar_front_suspension.front.scaleFactor`
- `rear_suspension.antiroll.CompactCar_rear_suspension.rear.scaleFactor`

To promote a candidate to a factor:

- expand the candidates tree to find the targets (enter `*scaleFactor*` in **Name Filter** field to narrow the search)



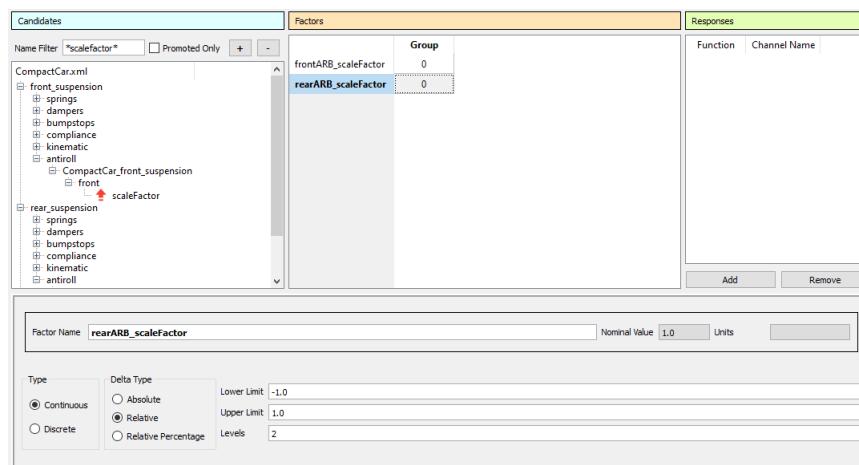
- right-click on the candidate and select **Promote to Factor** (otherwise simply double click on it)
- do it for both `antiroll` scale factors



The name of the factor can be very long since its name is composed by the complete tree structure where it's nested; it's possible to rename each factor assigning a custom name.

To do it:

- click on the first factor;
- edit the name of the **Factor Name** field changing it to **frontARB_scaleFactor**;
- press **Enter** to apply the change;
- repeat the same operation for the second factor entering **rearARB_scaleFactor** as **Factor Name**.

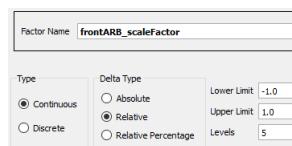


Now we need to set the range of variation of the two factors for the investigation.

In particular, we'll vary the scaling factors in order to check the anti-roll bars impact on the vehicle when each scaling factor is swept from **0** (no anti-roll effect) up to **2** (double anti-roll effect).

To do it:

- click on **frontARB_scaleFactor**;
- set **Type** to **Continuous**;
- set **Delta Type** to **Relative**;
- set **Lower Limit** to **-1.0**;
- set **Upper Limit** to **1.0**;
- set **Levels** to **5**;



- repeat the same operation for **rearARB_scaleFactor**.

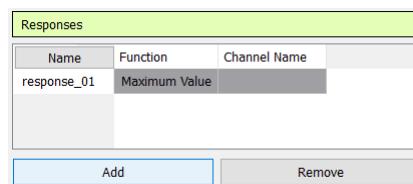
Responses

Our target is to monitor the maximum lateral acceleration of the chassis, so we need to create one response which monitors such output and computes its maximum value.

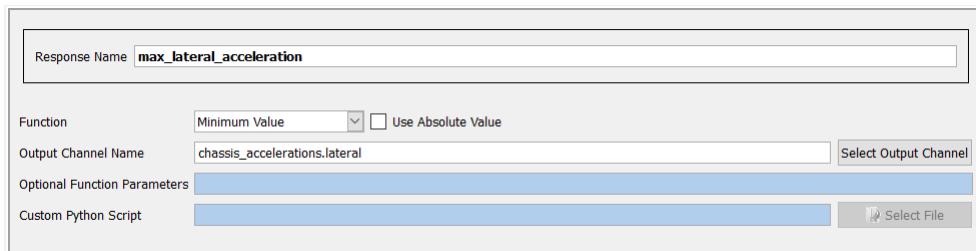
Note: since in Set up Ramp Steer event topic we've submitted an event in which the vehicle turns on the left, the chassis lateral acceleration will be negative, so the target for our response must be the **Minimum Value**.

To create a new Response:

- click **Add** button in responses section



- double click on **response_01** to open the response section panel;
- in **Response Name** field, change the response name to **max_lateral_acceleration**;
- set **Function** option menu to **Minimum Value**;
- click on **Select Output Channel** button and select **chassis_accelerations.lateral** output.

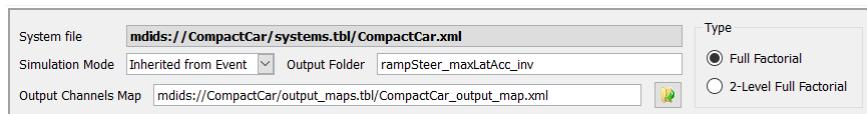


Investigation Parameters

The bottom part of the investigation section defines general investigation parameters.

In particular for our purpose we set:

- Output Folder** name to **rampSteer_maxLatAcc_inv**;
- Type** to **Full-Factorial**.



The investigation is now set-up. 25 combinations of factors will be run as can be seen in the HTML Investigation Design Space  report.

Investigation Name	investigation_1
System File	mdids://CompactCar/systems.tbd/CompactCar.xml
Strategy	Full Factorial
Output Folder	rampSteer_maxLatAcc_inv

Selected Event: fingerprint_1.CompactCar_ramp

Identifier	frontARB_scaleFactor	rearARB_scaleFactor	Success	max_lateral_acceleration
v0001	0.00000	0.00000	?	N/A
v0002	0.00000	0.500000	?	N/A
v0003	0.00000	1.00000	?	N/A
v0004	0.00000	1.50000	?	N/A
v0005	0.00000	2.00000	?	N/A
v0006	0.500000	0.00000	?	N/A
v0007	0.500000	0.500000	?	N/A
v0008	0.500000	1.00000	?	N/A
v0009	0.500000	1.50000	?	N/A
v0010	0.500000	2.00000	?	N/A
v0011	1.00000	0.00000	?	N/A
v0012	1.00000	0.500000	?	N/A
v0013	1.00000	1.00000	?	N/A
v0014	1.00000	1.50000	?	N/A
v0015	1.00000	2.00000	?	N/A
v0016	1.50000	0.00000	?	N/A
v0017	1.50000	0.500000	?	N/A
v0018	1.50000	1.00000	?	N/A
v0019	1.50000	1.50000	?	N/A
v0020	1.50000	2.00000	?	N/A
v0021	2.00000	0.00000	?	N/A
v0022	2.00000	0.500000	?	N/A
v0023	2.00000	1.00000	?	N/A
v0024	2.00000	1.50000	?	N/A
v0025	2.00000	2.00000	?	N/A

Note: in the report Success column values have ? and the related response is set to N/A since the investigation hasn't been run yet.

Click to save the investigation xml file in the working directory.

Run the Investigation

Click button to run the investigation.

All the combinations will be run in the investigation folder (`rampSteer_maxLatAcc_inv`).

```

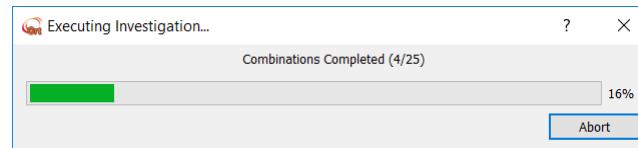
Select VI RTW (VI-CarRealTime Python Task Window)
Solving for static equilibrium...
Static equilibrium has been achieved.
Elapsed Simulation Time: 2.071 (sec).
Elapsed Clock Time: 0.088 (sec).

Setup Completed.
=====
= VI-AdvancedSteering =
=====
VERSION : 20.0
REVISION: 46607_Beta//branches/20/Main/software
BUILT ON: phoenix/18-Jun-20

=====
Initializing steering system... OK
Done.

Performing dynamic simulation...
Simulation Progress (percent complete):
0 ..... 50 ..... 100
=====
```

In VI-CarRealTime GUI, a progress dialog will be displayed to show the combinations completed and the percentage status:



Note: once the investigation xml file has been saved, it's also possible to run the investigation in batch mode. Please refer to [Run the Investigation in batch mode](#) for the details.

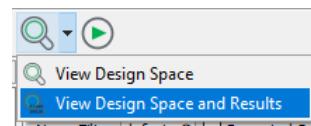
Wait until all the investigation analyses have been run.

Analyze Investigation Results

When the investigation is completed, it's possible to review and analyze the results. In particular, in VI-CarRealTime working directory the HTML report can be found.

To open it:

- double click on *rampSteer_maxLatAcc_inv_summary.html* file in VI-CarRealTime working directory or
- click on lens stack button arrow and select **View Design Space and Results**.



The report shows the response values for all the factor combinations:

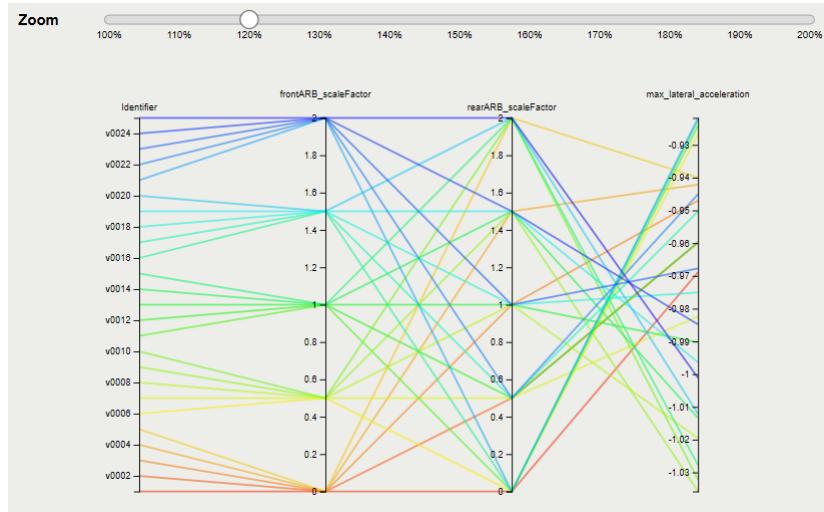
Investigation Summary Report

Investigation Name	investigation_1
System File	mdids://CompactCar/systems.tbd/CompactCar.xml
Strategy	Full Factorial
Output Folder	rampSteer_maxLatAcc_inv

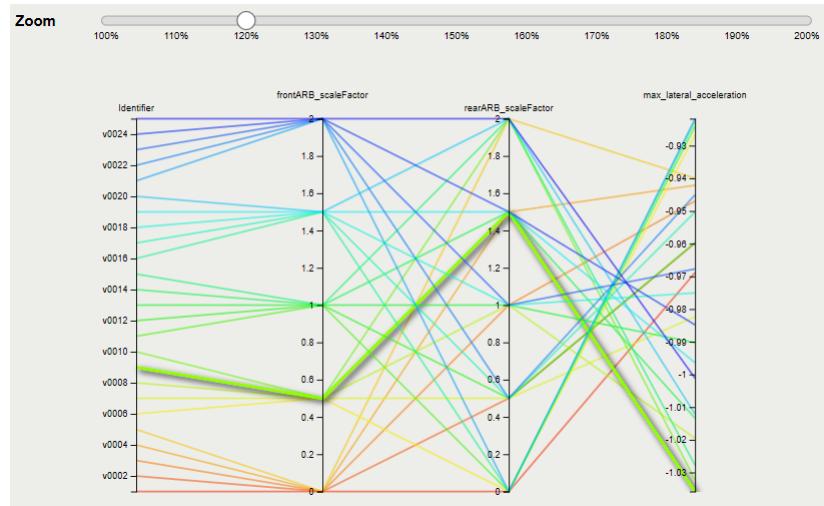
Selected Event: **fingerprint_1.CompactCar_ramp**

Identifier	frontARB_scaleFactor	rearARB_scaleFactor	Success	max_lateral_acceleration
v0001	0.00000	0.00000	✓	-0.988845
v0002	0.00000	0.500000	✓	-0.959811
v0003	0.00000	1.00000	✓	-0.946943
v0004	0.00000	1.50000	✓	-0.942118
v0005	0.00000	2.00000	✓	-0.939982
v0006	0.500000	0.00000	✓	-0.926695
v0007	0.500000	0.500000	✓	-0.982370
v0008	0.500000	1.00000	✓	-1.01956
v0009	0.500000	1.50000	✓	-1.03582
v0010	0.500000	2.00000	✓	-1.03221
v0011	1.00000	0.00000	✓	-0.924101
v0012	1.00000	0.500000	✓	-0.959882
v0013	1.00000	1.00000	✓	-0.990283
v0014	1.00000	1.50000	✓	-1.01338
v0015	1.00000	2.00000	✓	-1.02778
v0016	1.50000	0.00000	✓	-0.922564
v0017	1.50000	0.500000	✓	-0.950320
v0018	1.50000	1.00000	✓	-0.975055
v0019	1.50000	1.50000	✓	-0.996388
v0020	1.50000	2.00000	✓	-1.01214
v0021	2.00000	0.00000	✓	-0.921749
v0022	2.00000	0.500000	✓	-0.945048
v0023	2.00000	1.00000	✓	-0.987725
v0024	2.00000	1.50000	✓	-0.984327
v0025	2.00000	2.00000	✓	-1.00116

Moreover, the last part of the report shows the plot of the two factors together with the response:



From the dynamic plot, it's possible to highlight the variant combination which provides the maximum (in absolute value) lateral acceleration:



So, the combination v0009 provides the maximum lateral acceleration of the vehicle (**1.03582 g**) with respect to a value of **0.990302 g** for the baseline model:

Identifier	frontARB_scaleFactor	rearARB_scaleFactor	Success	max_lateral_acceleration
v0001	0.00000	0.00000	✓	-0.968845
v0002	0.00000	0.500000	✓	-0.959811
v0003	0.00000	1.00000	✓	-0.946943
v0004	0.00000	1.50000	✓	-0.942118
v0005	0.00000	2.00000	✓	-0.939962
v0006	0.500000	0.00000	✓	-0.926695
v0007	0.500000	0.500000	✓	-0.982370
v0008	0.500000	1.00000	✓	-1.01956
v0009	0.500000	1.50000	✓	-1.03582
v0010	0.500000	2.00000	✓	-1.03221

Note: the result is compliant for the CompactCar vehicle which has an understeer tendency: the maximum lateral acceleration has been increased by softening (50%) the front anti-roll bar and by stiffening (50%) the rear anti-roll bar.

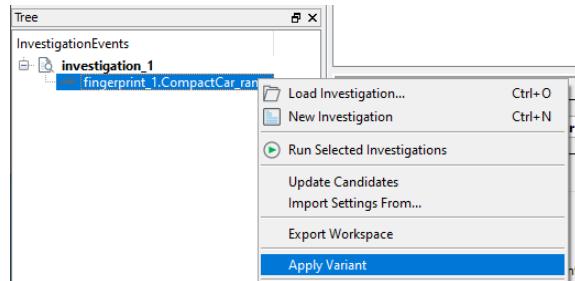
Run MaxPerformance with Optimized Model

From the investigation analysis we've found out a model combination variant which provides a bigger maximum lateral acceleration in the ramp steer event.

Let's see how the new vehicle behaves in the chicane event.

To simulate using the vehicle set up computed in investigation variant v0009 :

- right click on the investigation instance and select **Apply Variant**;

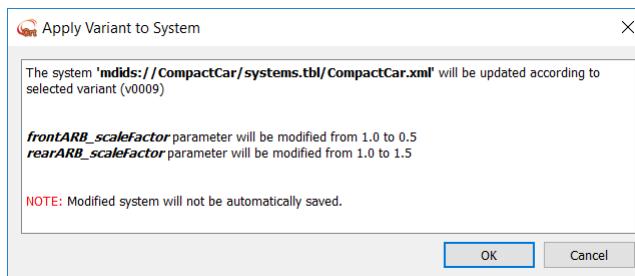


- Select Variant** panel will be displayed..

Identifier	frontARB_scaleFactor	rearARB_scaleFactor	Success	max_lateral_acceleration
v0001	0.000000	0.000000	True	-0.968845
v0002	0.000000	0.500000	True	-0.959811
v0003	0.000000	1.000000	True	-0.946943
v0004	0.000000	1.500000	True	-0.942118
v0005	0.000000	2.000000	True	-0.939962
v0006	0.500000	0.000000	True	-0.926695
v0007	0.500000	0.500000	True	-0.982370
v0008	0.500000	1.000000	True	-1.019562
v0009	0.500000	1.500000	True	-1.035822
v0010	0.500000	2.000000	True	-1.032211
v0011	1.000000	0.000000	True	-0.924101
v0012	1.000000	0.500000	True	-0.959652
v0013	1.000000	1.000000	True	-0.990283

- select v0009 rows and click **OK** button.

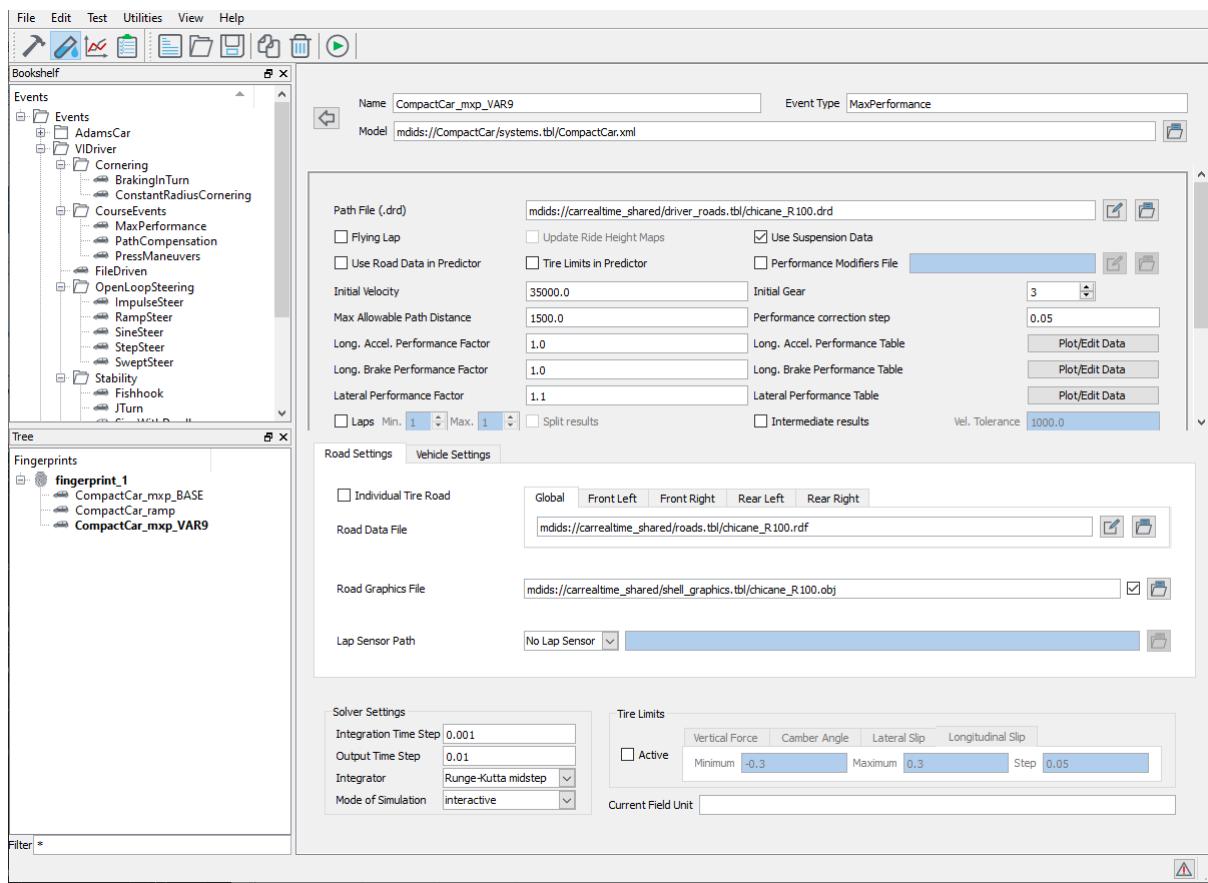
The variant v0009 parameters will be applied to the CompactCar model in Build Mode. When the process terminates, the following message appears:



To create the new MaxPerformance event:

- copy the MaxPerformance event created in [Run MaxPerformance with Base Model](#) section;
- change the event name to **CompactCar_mxp_VAR9**;
- change the **Mode of Simulation** to **interactive**;

- keep the same values for the other parameters.



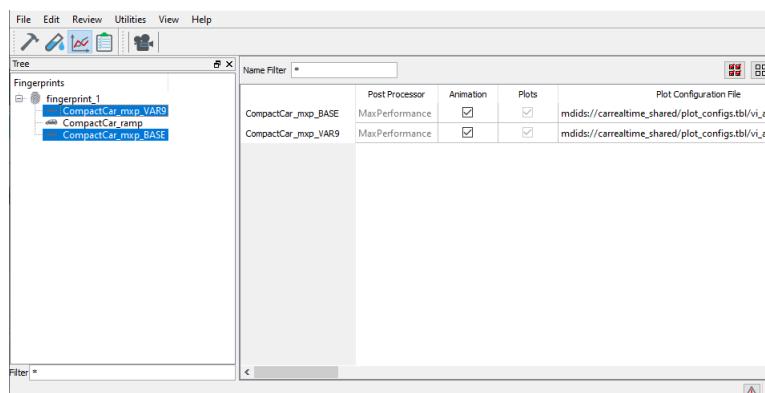
Press button in the toolbar to run the MaxPerformance event.

Compare the Results

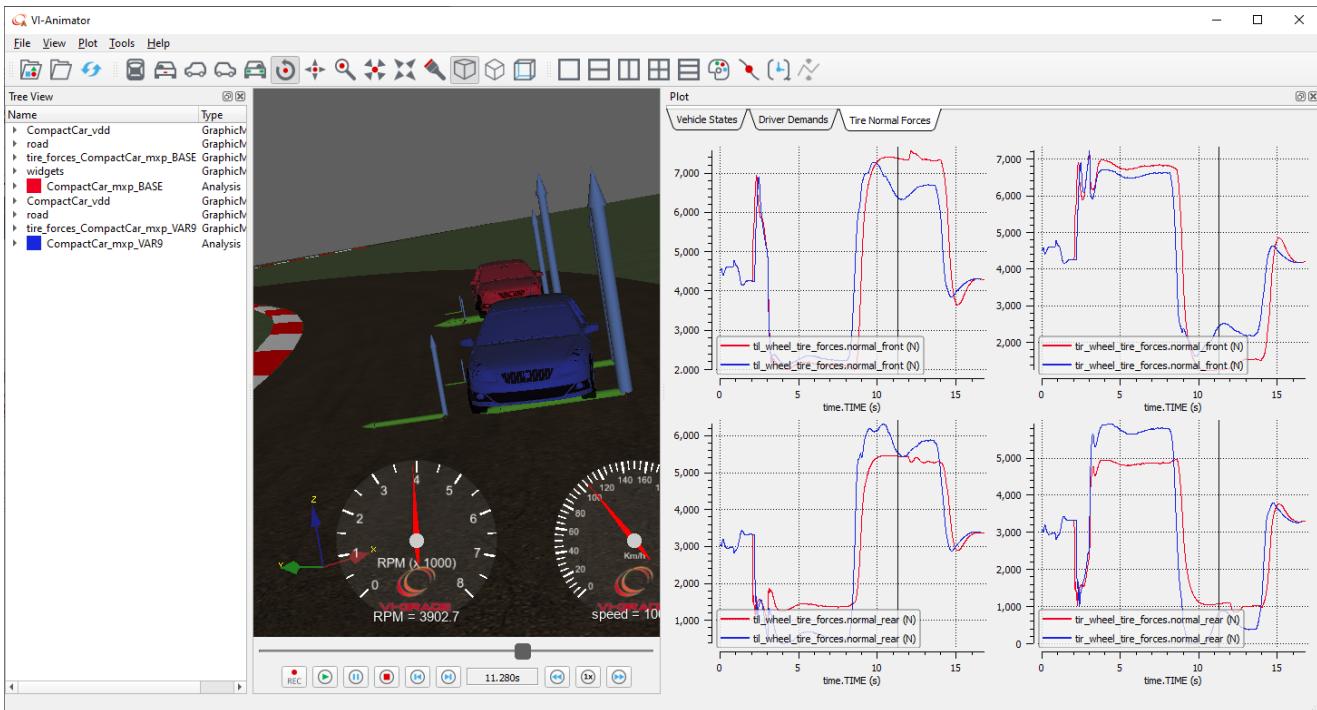
In the last section we'll compare the two MaxPerformance simulations (baseline vs. variant model).

Switch to **Review Mode** .

Select both MaxPerformance events and press button in the toolbar to launch VI-Animator.



Review the results:



The optimized model (blue) is faster than the baseline one (red).

Due to the modification in the anti-roll stiffness (softer front and stiffer rear), the optimized model presents less understeer and less lateral load transfer: it reflects in a faster curve entrance due to the reduced understeer and in an improved global performance in the chicane event.

Using 7Post Rig

The aim of this tutorial is to instruct the user about how to use [7 Post testrig](#).

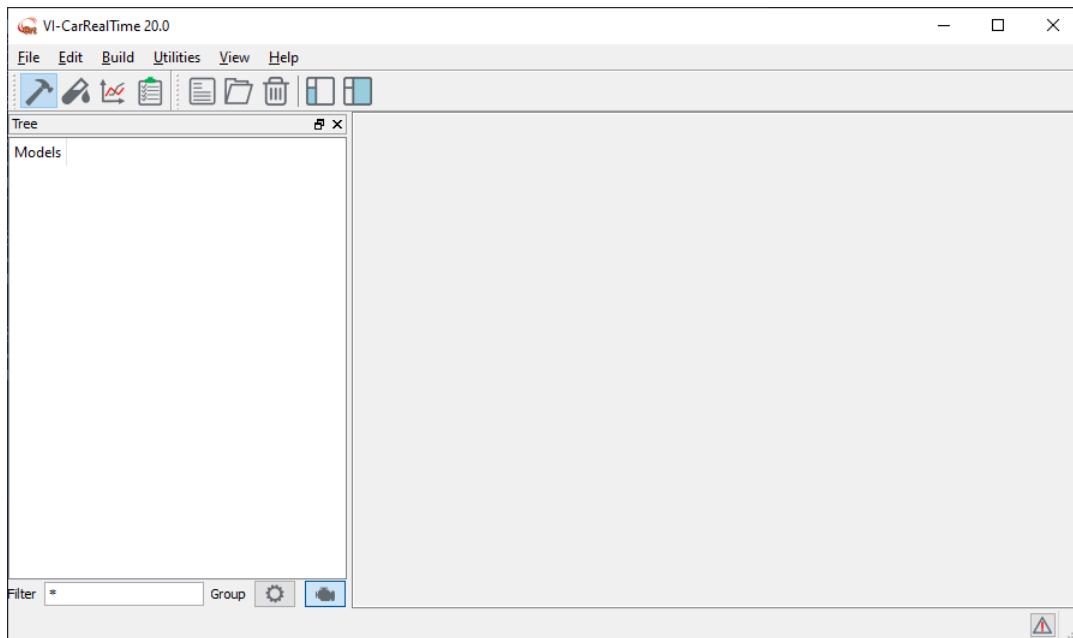
In particular, the tutorial shows how to replicate, using the 7 Post rig, the results obtained running a FileDriven analysis.

The tutorial is composed by the following sections:

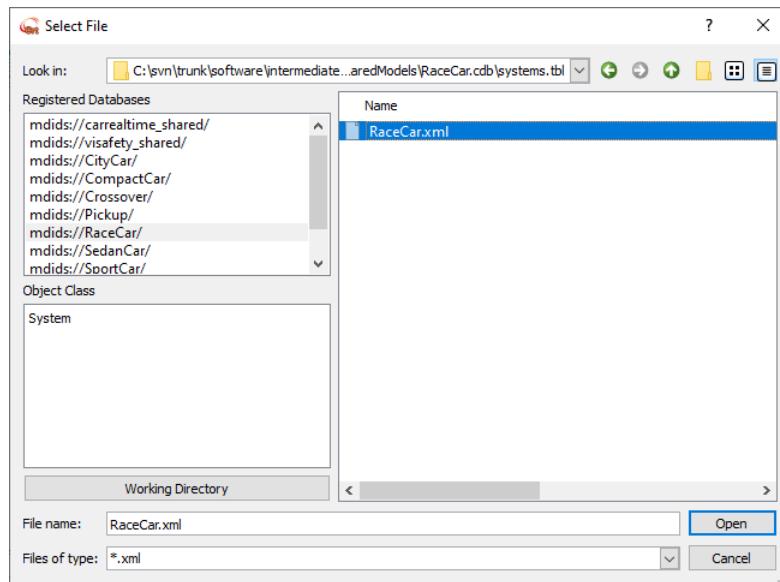
- [Model Set-up](#)
- [Run Events](#)
- [Manage SPC File](#)
- [Run 7Post Rig Event](#)
- [Review Results](#)

Model Set-up

Start a new VI-CarRealTime session from the Windows Start Menu or by typing **vicrt20** from a DOS (cmd) shell.



In **Build mode**, go to Build -> Load Model, and load the **RaceCar.xml** vehicle model from the **RaceCar** database.

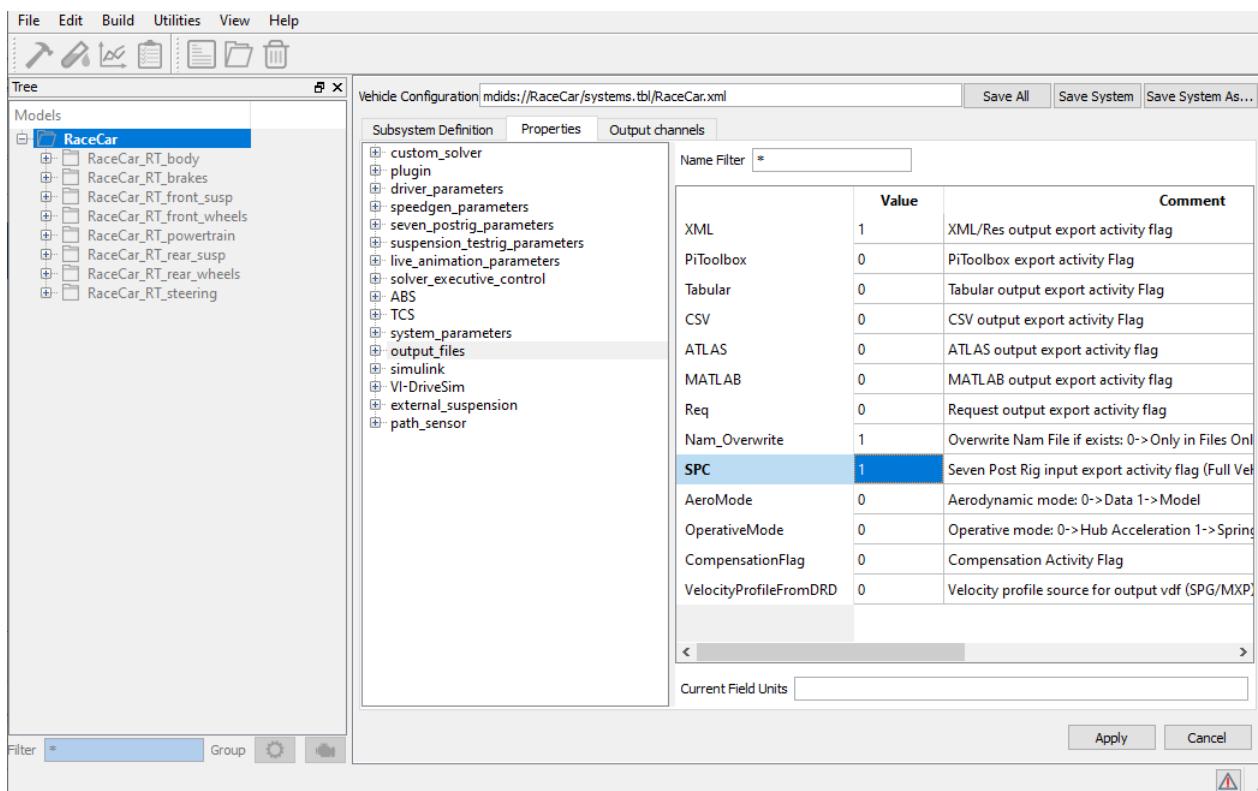


The next step consists in activating the SPC output format from vehicle system properties.

To do it:

- in **Build Mode**, click on **RaceCar** system;
- select **Properties** panel;
- click on **output_files** section;
- set [SPC output](#) to **1** as shown in the picture below;
- press **Apply** button.

VI-CarRealTime

**Run Events**

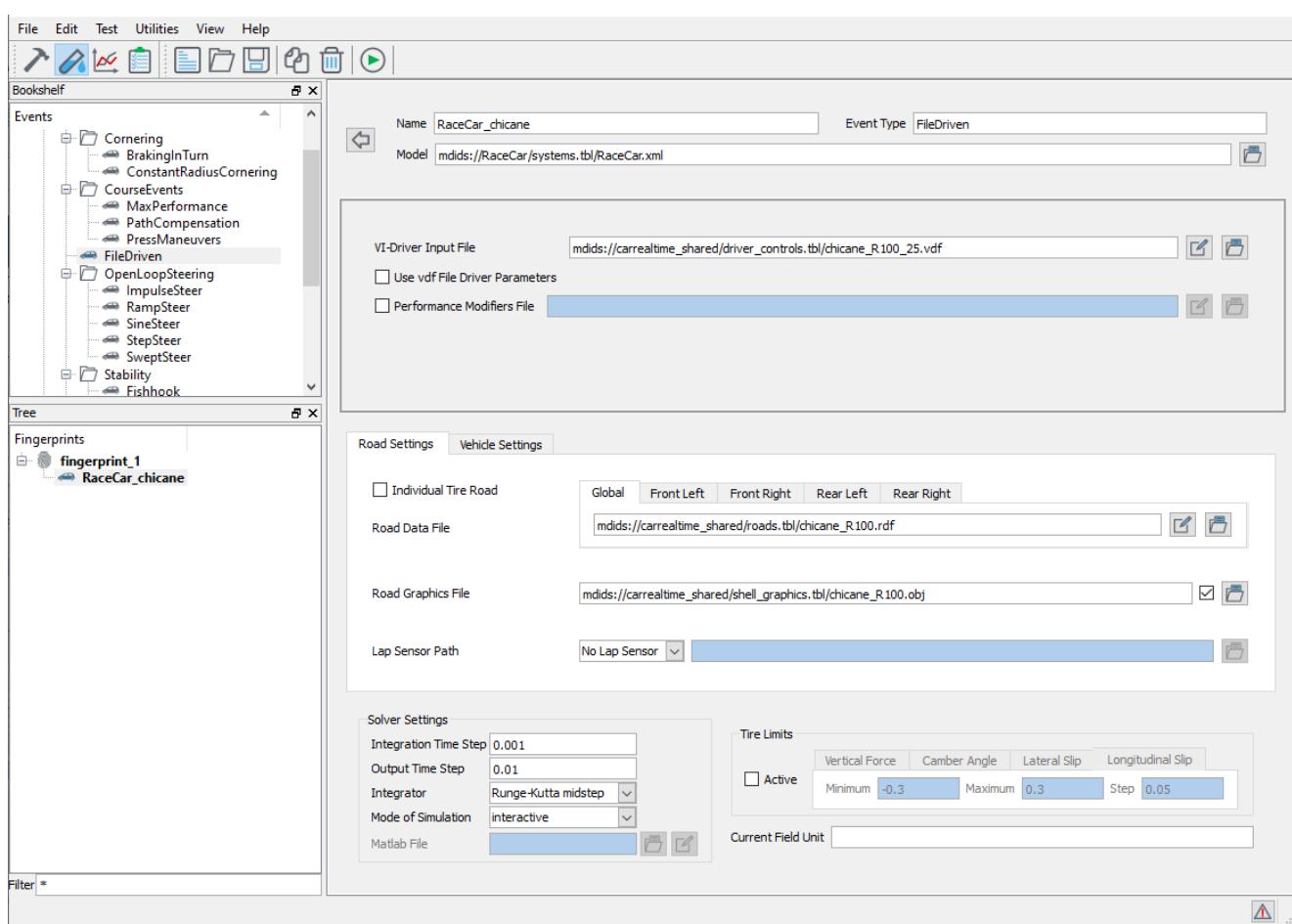
In this section we will move to Test Mode and we will run two events:

1. a [FileDriven](#) event in which the RaceCar model will be driven in a chicane
2. a [7PostRig](#) event using the SPC file generated from the previous FileDriven event

Move to **Test Mode** and add a **FileDriven** event.

Set the following parameters for the event:

- change the event name to **RaceCar_chicane**;
- set VI-Driver Input File to **mdids://carrealtime_shared driver_controls.tbl/chicane_R100_25.vdf**;
- set Road Data File to **mdids://carrealtime_shared/roads.tbl/chicane_R100.rdf**;
- set Road Graphics File to **mdids://carrealtime_shared/shell_graphics.tbl/chicane_R100.obj**.



Press button in the toolbar to run the event.
Wait until the simulation is completed.

Note: in VI-CarRealTime working directory the file *RaceCar_chicane.spc* will be written by the solver. For any detail regarding the file content, please refer to [SPC File Format](#).

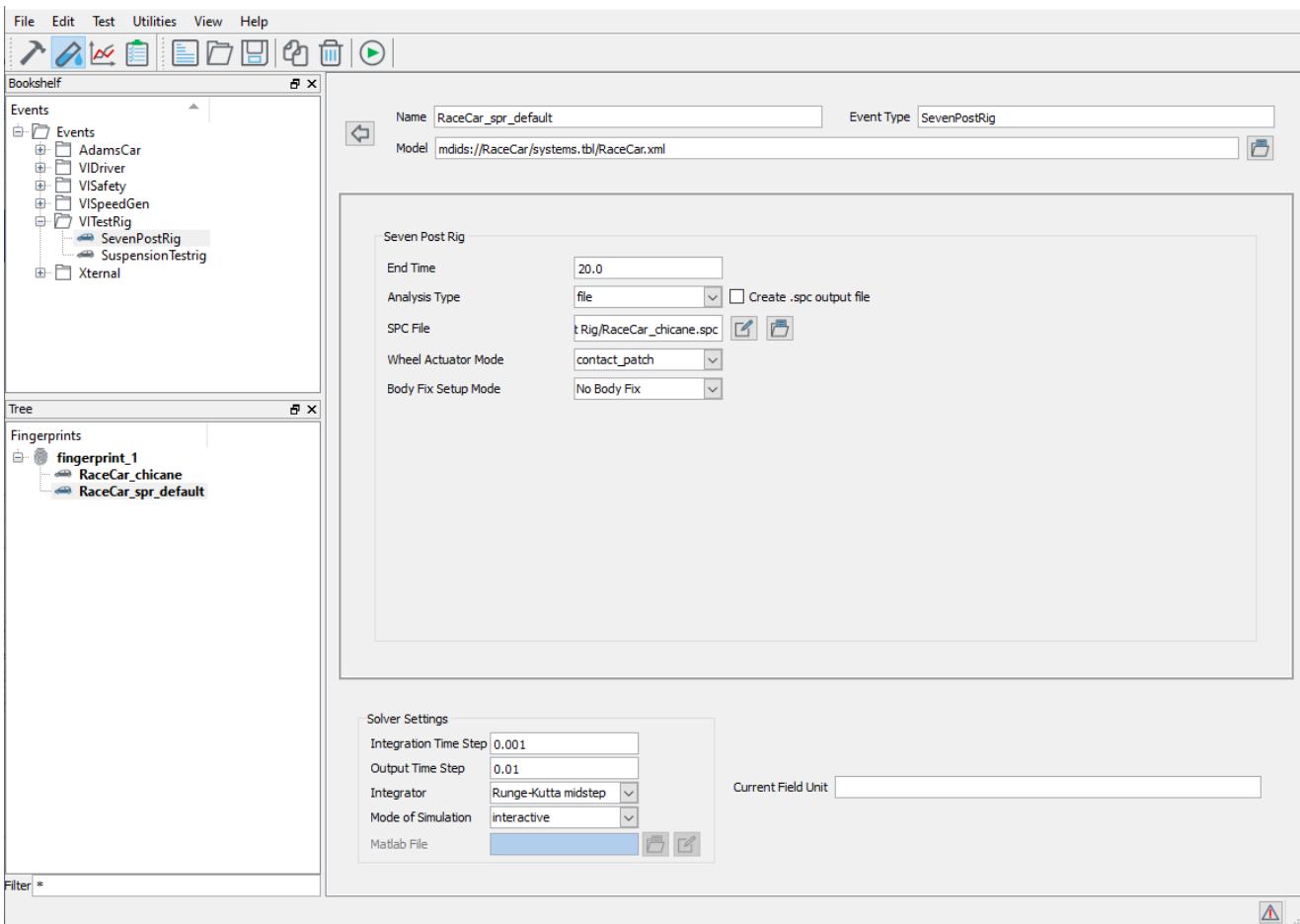
Now the next step consists in running a 7Post event, using the SPC file generated by the previous FileDriven event.

In Test Mode, add a new **SevenPostRig** event.

Set the following parameters for the event:

- change the event name to **RaceCar_spr_default**;
- set **End Time** to **20.0** seconds;
- set **Analysis Type** to **file**;
- set **SPC File** to *RaceCar_chicane.spc*;
- set **Wheel Actuator Mode** to **contact_patch**.

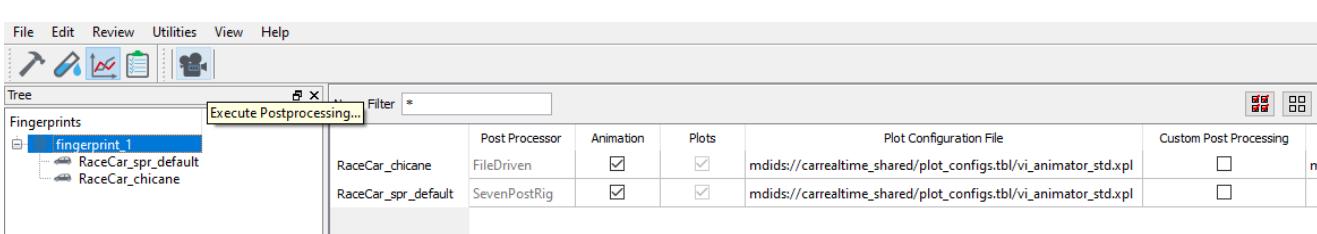
VI-CarRealTime



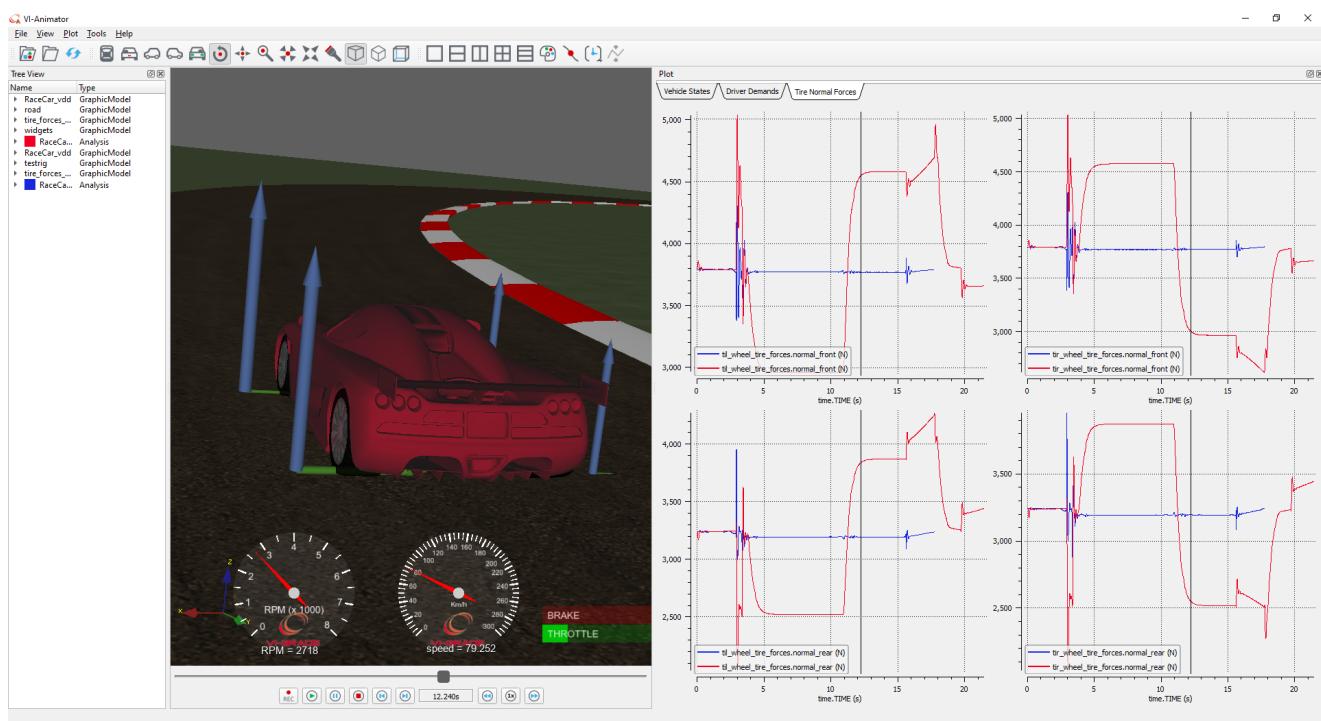
Press button in the toolbar to run the event.
Wait until the simulation is completed.

Switch to **Review Mode** in order to compare the simulation results between the FileDriven event and the 7Post event.

Select both the events and press button in the toolbar to launch VI-Animator.



Review results and plot from VI-Animator:



Manage SPC File

In this section we will check and "tune" the SPC file generated with the previous FileDriven event, in order to be able to submit a 7PostRig event whose can be compared with the original FileDriven analysis.

Browse to VI-CarRealTime working directory and edit `RaceCar_chicane.spc` file with a text editor:

```
D:\>development>2020>carrealtime>B13064>i2> RaceCar_chicane.spc
1 $----- VIGRADE_HEADER
2 [VIGRADE_HEADER]
3 FILE_TYPE = 'spc'
4 FILE_VERSION = 2.0
5 FILE_FORMAT = 'ASCII'
6 $----- UNITS
7 [UNITS]
8 LENGTH = 'm'
9 ANGLE = 'radian'
10 FORCE = 'newton'
11 MASS = 'kg'
12 TIME = 'second'
13 $----- SEVENPOST_SETUP
14 [SEVENPOST_SETUP]
15 OPERATING_MODE = 'TRACK_HUBACC'
16 AERODYNAMICS = 'DATA'
17 STEER_MODE = 'FILE'
18 ACCEL_COMPENSATION = 'OFF'
19 ROLL_COMPENSATION = 'OFF'
20 PITCH_COMPENSATION = 'OFF'
21 $----- DATA
22 [DATA]
23 [Line] FL FR RL RR F_downforce RL_downforce RR_downforce lonacc latacc lon_vel steer_angle
24 0 -0.31649593072 0.445211030618 -0.44566359943 207.78972594 200.764282502 0.53280142924 -1.22088019611e-006 24.999987147 0
25 -0.045226934614 -0.0461219771786 -0.445687288275 -0.443631859277 207.799639988 200.773217235 200.773217235 -1.31183554886e-005 25.00085930222 4.94932090716e-014
26 0.02 -0.14386219094 -0.143331149855 0.440404041373 0.440789047242 207.777756852 200.752653697 -0.710688644994 1.11481211728e-005 24.9913993824 1.82347183696e-010
27 -0.0728302713856 -0.0720263463918 0.24433195298 0.24385142182 207.618439544 200.598723012 -0.246404149712 2.0438548232e-005 24.9876813973 2.4405328529e-009
28 0.04 -0.0248088749988 -0.0239853617994 -0.839758418097 -0.840494607928 200.538706 200.5398706 -0.610420137531 3.35570791242e-005 24.9838073187 9.0027066061e-009
29 0.05 -0.00622018235982 -0.0064411453331 0.109240327235 0.108573384601 200.474233095 200.457653137 -0.48229629468 5.719614594841e-005 24.9774922975 2.81829727533e-008
30 0.06 0.0439534764083 0.0441816073795 -0.076046326206 0.0764608848278 200.363558209 200.363558209 -0.4208439863638 6.82956699031e-005 24.9732439751 5.4772269522e-008
31 0.07 0.0836268659757 0.0839104852483 -0.0886402592883 0.088643085469 200.295542636 200.295542636 -0.471584840708 7.53970431773e-005 24.9686599108 7.83274944678e-008
32 0.08 0.089696961718 0.08866670726497 -0.08836670726497 0.08837332741611 200.223219642 200.223219642 -0.40740279681 7.49623933886e-005 24.964213265 6.55437562245e-008
33 0.09 0.118881127844 0.111177699944 -0.125346673283 0.125346673283 200.157565831 200.157565831 -0.377317757 6.68233469374e-005 24.9603025557 -1.48982382099e-008
34 0.1 0.111449519751 -0.1016540826997 0.207.184625938 200.181737819 -0.350648481459 5.581073065958e-005 24.95694266 -2.0958714773e-007
35 0.11 0.0950152220728 0.095191621088 -0.095242375280 0.095242375280 200.050504083 0.3072517169 4.9532778428 -5.465654948554e-007
36 0.12 0.0822260666666 0.0822260666666 0.0822260666666 0.0822260666666 200.050504083 0.3072517169 4.9532778428 -1.092292753e-006
37 0.13 0.0472374813697 0.047243252537 0.0516377417807 0.0516394164652 200.965618066 199.967973859 199.967973859 2.14943935122e-005 24.9472152451 1.65457520238e-006
38 0.14 0.0163744794633 0.0163887812312 -0.032589893892 0.032589893892 200.935673636 199.935873827 -0.195362711869 1.547854874e-005 24.9456382287 -3.372942718e-006
39 0.15 -0.01227668261545 -0.01227668261545 -0.01058542251539 0.01058542251539 200.902289042 199.9067868032 -0.162330838056 1.3102160631e-005 24.9438955674 -3.13878486869e-006
40 0.16 -0.037511363694 -0.0376372767354 0.0115131490168 0.0114457268699 200.877839548 199.88239024 -0.135173126213 1.3636700447e-005 24.9422751801 -3.87568049103e-006
41 0.17 -0.058389551671 -0.0584976375227 0.030138168967 0.0306797298418 200.854894519 199.860993948 199.860993948 -0.11490426928 1.58996956171e-005 24.940983696 -4.18570416183e-006
42 0.18 -0.0724763447437 0.0725773073403 0.0457865675559 0.0457865675559 200.834770251 199.841550096 199.841550096 -0.098571182676 1.85162849477e-005 24.9397858835 -5.0178457692e-006
43 0.19 -0.0795628954026 0.079669116262 0.0560663930422 0.05606574885346 200.815877449 199.823296033 199.823296033 -0.0898159262872 2.0261158988e-005 24.9388890116 -5.3456389222e-006
44 0.2 -0.0795283527022 -0.0795422958842 0.0686738271748 0.0686781453397 200.797395638 199.805440853 199.805440853 -0.0808373358374 2.05958493384e-005 24.9379617877 -5.5049836681e-006
45 0.21 -0.0722820499012 -0.0722762038471 0.060151640698 0.0601685898638 200.77864142 199.787368294 199.787368294 -0.08485899889 1.9238192459e-005 24.9370563824 -5.526375965054e-006
46 0.22 -0.0587370970864 0.05872158469 0.0548701435471 0.054104348742 200.759576968 199.768897919 199.768897919 -0.0842092410242 1.64291288049e-005 24.9361561262 -5.4542171234e-006
47 0.23 -0.040966166395 0.0409386557264 0.042556529361 0.042556529361 200.749299303 199.750269144 199.750269144 -0.0831399240384 1.2546086964e-005 24.9352633666 -5.3156258524e-006
48 0.24 -0.0212233638945 -0.021178432133 0.0280270996468 0.027937099103 200.721329982 199.73194493 199.73194493 -0.082754697925 7.3808414636e-006 24.934379423 -5.1288153072e-006
49 0.25 -0.08017024659882 0.080165578198976 0.0813408532095 0.0813408532095 200.703832089 199.714266905 199.714266905 -0.0813551821221 2.68332547814e-006 24.933580369 -4.7447383646e-006
50 0.26 0.0158804471962 0.01572065451 0.0084158031366 0.0084158031366 200.685639715 199.697457895 199.697457895 -0.08083553120982 -2.68332547814e-006 24.933575261 -4.68111561558e-006
51 0.27 0.005123339451 0.005123339451 0.005123339451 0.005123339451 200.663938272 199.6816672117 199.6816672117 -0.0793322369589 -2.68332547814e-006 24.933575261 -4.68111561558e-006
```

Looking at the `[SEVENPOST_SETUP]` section, it is possible to see that by default the following parameters are set to 'OFF':

- `ACCEL_COMPENSATION`

VI-CarRealTime

- ROLL_COMPENSATION
- PITCH_COMPENSATION

It means that when a 7Post rig event is submitted using such SPC file, the testrig actuators won't compensate the tracked acceleration to match the target longitudinal accelerations and lateral accelerations defined in the 9th and 10th column of the file.

That explains why in the previous results comparison, the 7Post rig analysis results don't show any lateral load transfer when comparing the Tire Normal Forces.

Set these parameters to 'ON', allows the solver to compensate for these accelerations.

Note: for all the details regarding the file parameters, please refer to [SPC File Format](#).

To modify the SPC file content:

- edit **RaceCar_chicane.spc** file with a text editor;
- set ACCEL_COMPENSATION = 'ON'
- set ROLL_COMPENSATION = 'ON'
- set PITCH_COMPENSATION = 'ON'
- save the file with the same name (overwrite the existing one)

```

1  $-----VIGRADE_HEADER
2  [VIGRADE_HEADER]
3  FILE_TYPE      = 'spc'
4  FILE_VERSION   = 2.0
5  FILE_FORMAT    = 'ASCII'
6  $-----UNITS
7  [UNITS]
8  LENGTH         = 'm'
9  ANGLE          = 'radian'
10 FORCE          = 'newton'
11 MASS           = 'kg'
12 TIME           = 'second'
13 $-----SEVENPOST_SETUP
14 [SEVENPOST_SETUP]
15 OPERATING_MODE = 'TRACK_HUBACC'
16 AERODYNAMICS   = 'DATA'
17 STEER_MODE     = 'FILE'
18 ACCEL_COMPENSATION = 'ON'
19 ROLL_COMPENSATION = 'ON'
20 PITCH_COMPENSATION = 'ON'
21 $-----DATA

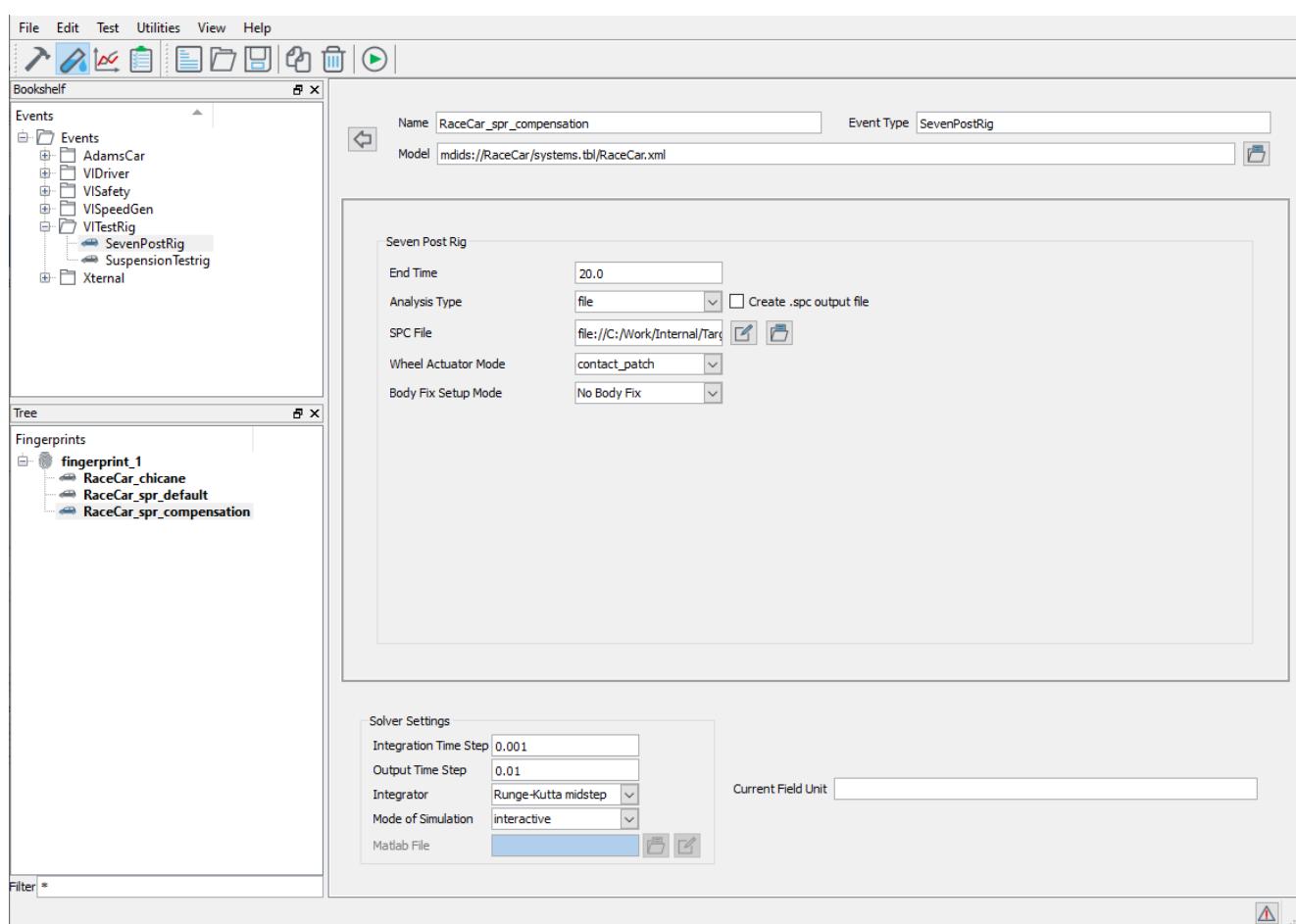
```

Run 7Post Rig Event

Now we will run the same 7Post rig event as the one run before, but this time the event will use the SPC file with compensated accelerations.

Switch back to **Test Mode** and:

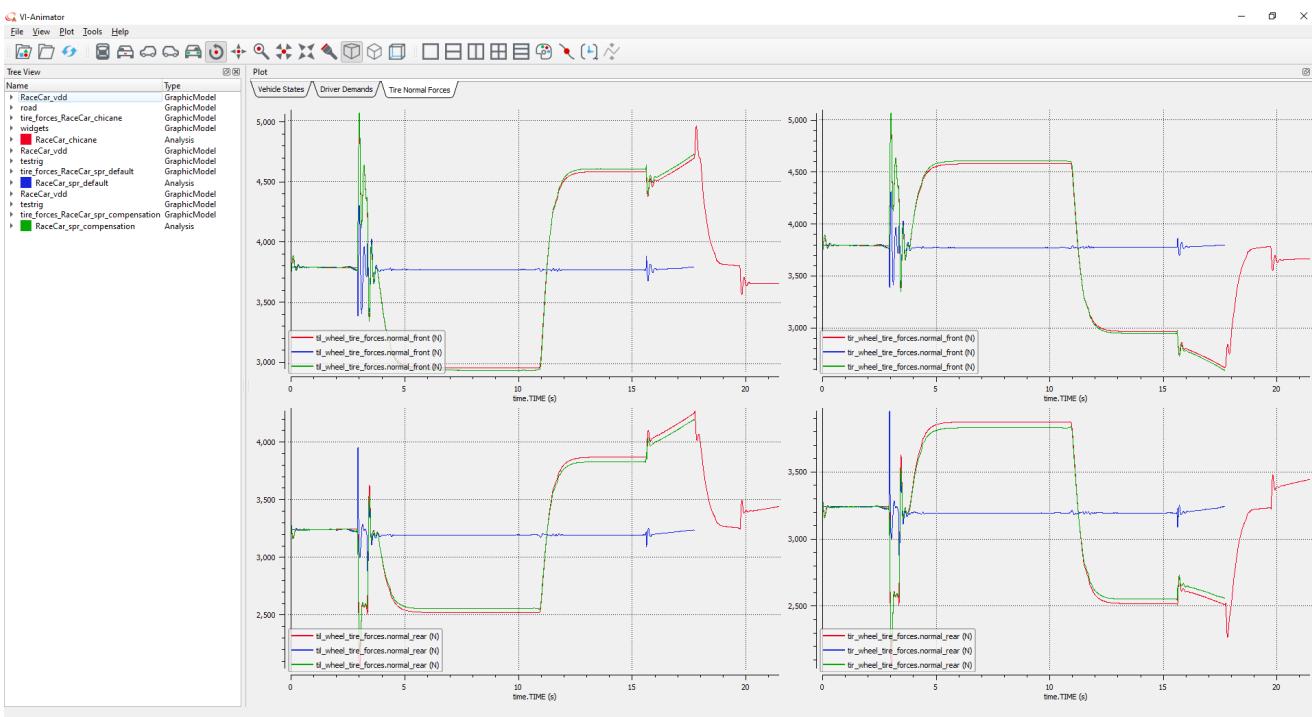
- right click on **RaceCar_spr_default** event and select **Copy Selected Event**;
- rename the new event to **RaceCar_spr_compensation**;
- use exactly the same parameters as the previous event (the only difference will be the SPC file, that now has the acceleration compensation flags set to ON)



Press button in the toolbar to run the event.
Wait until the simulation is completed.

Review Results

Compare the results of the three analyses in VI-Animator.



The 7Post rig analysis with the acceleration compensation set to ON (green curves) shows a very good correlation in terms of Normal Tire Forces with respect to the original FileDriven analysis (red curves).

1.4.2 MATLAB/Simulink

The tutorials in the following section are intended to make users more familiar with the MATLAB/Simulink co-simulation interface, in order to use VI-CarRealTime models together with controllers built using Simulink ([co-simulation](#)). Also, this section shows how to use a set of APIs to interact with VI-CarRealTime's main files, in order to perform a particular experiment aimed at finding the best vehicle and suspension configuration ([optimization tool](#)).

Below is a list of the exercises in this tutorial section:

[Co-Simulation](#)

- [Implementing ABS controller in Simulink](#)
- [Replacing VI-CarRealTime standard components](#)
- [Replacing driveline components](#)
- [Assessing vehicle active systems](#)
- [Implementing a damper skyhook control system](#)
- [Using VI-Driver/MaxPerformance in MatLab/Simulink](#)
- [Using VI-Driver/PressManeuvers in MatLab Simulink](#)
- [Using External Road](#)
- [Connecting External Steering Model](#)

- [Implementing an Electric Vehicle](#)

[VI-CarRealTime Solver Plugin Export from Simulink](#)

[Run Investigation with a Solver Plugin](#)

[API Toolkit](#)

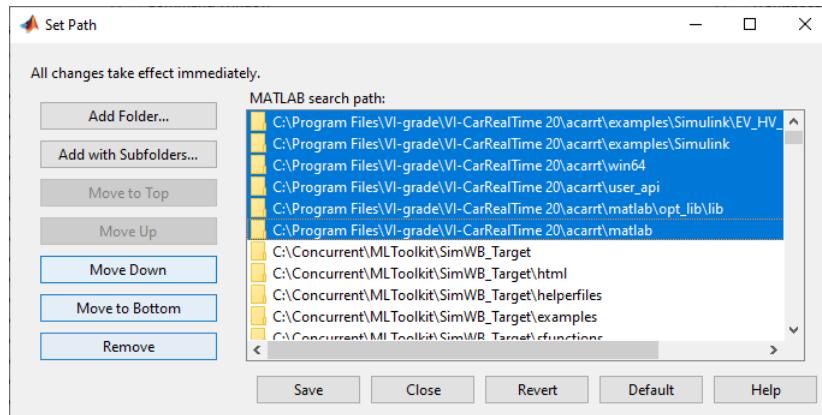
- [Manage a VI-CarRealTime Model](#)
- [Manage a VI-SuspensionGen Model](#)
- [Suspension Optimization](#)

Co-Simulation

In this chapter, the following topics will be covered:

- [Implementing ABS controller in Simulink](#)
- [Replacing VI-CarRealTime standard components](#)
- [Replacing driveline components](#)
- [Assessing vehicle active systems](#)
- [Implementing a damper skyhook control system](#)
- [Using VI-Driver/MaxPerformance in MATLAB/Simulink](#)
- [Using VI-Driver/PressManeuvers in MATLAB/Simulink](#)
- [Using External Road](#)
- [Implementing an Electric Vehicle](#)
- [Implementing an Hybrid Vehicle](#)
- [Connecting External Steering Model](#)

In order to connect VI-CarRealTime to a MATLAB/Simulink interface, the directories containing the VI-CarRealTime solver must be added to the MATLAB search path. To do so, open MATLAB in Windows and enter the "**addpath_vicrt_20**" script (without quotes) into the MATLAB command window. This command adds the necessary VI-CarRealTime directories to the MATLAB search path for the current session (you would need to repeat the aforementioned procedure if MATLAB were closed and reopened). In order to keep the VI-CarRealTime paths saved after running "addpath_vicrt_20", the MATLAB search path list must be saved within the **Set Path** menu from MATLAB toolbar (simply press the **Save** button once your **Set Path** window contains the same highlighted directories shown in the screenshot below).



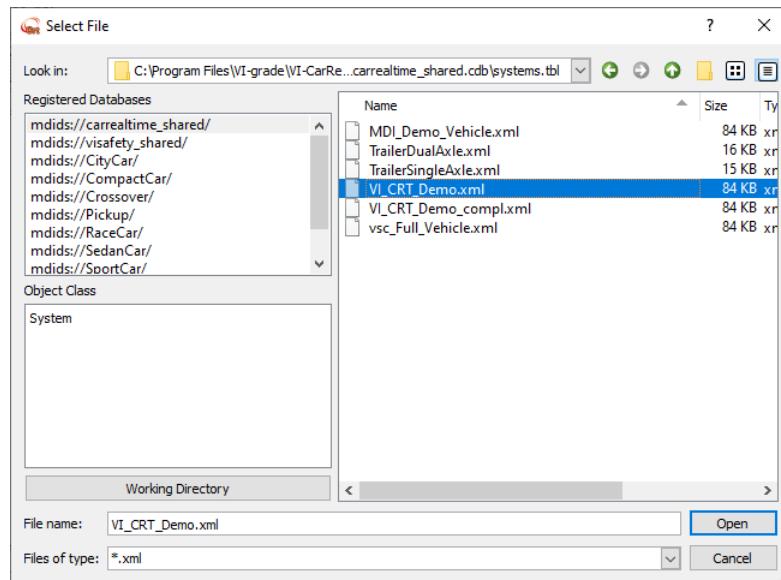
Implementing ABS in Simulink

The following tutorial introduces the use of the VI-CarRealTime MATLAB/Simulink interface.

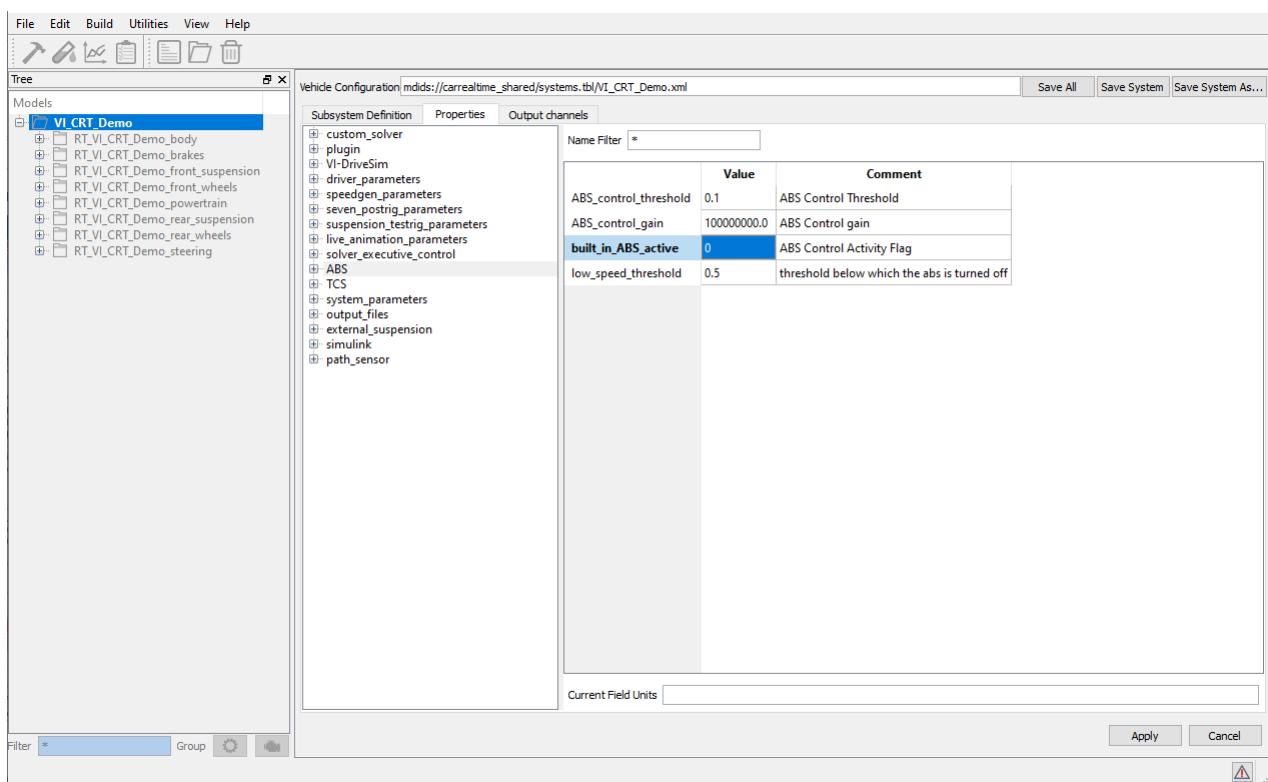
The particular case of a dynamic simulation, including an ABS modeled in MATLAB/Simulink, will be discussed in the following paragraphs.

Start a new VI-CarRealTime session using the **vicrt20** command from a DOS (cmd) shell, or by using the VI-CarRealTime shortcut from the Windows Start Menu.

Starting in **Build mode**, open the model **VI_CRT_Demo.xml** from the shared database.



Click on the **VI_CRT_Demo** system model in the tree view on the left, select the **Properties** tab, browse to the **ABS** section, and set the **built_in_ABS_active** flag to 0.



Click the **Apply** button in the bottom-right of the GUI to apply the aforementioned change.

In this fashion, the built-in VI-CarRealTime ABS model has been deactivated; this has been done because the ABS logic will be managed by Simulink, for the purposes of this tutorial.

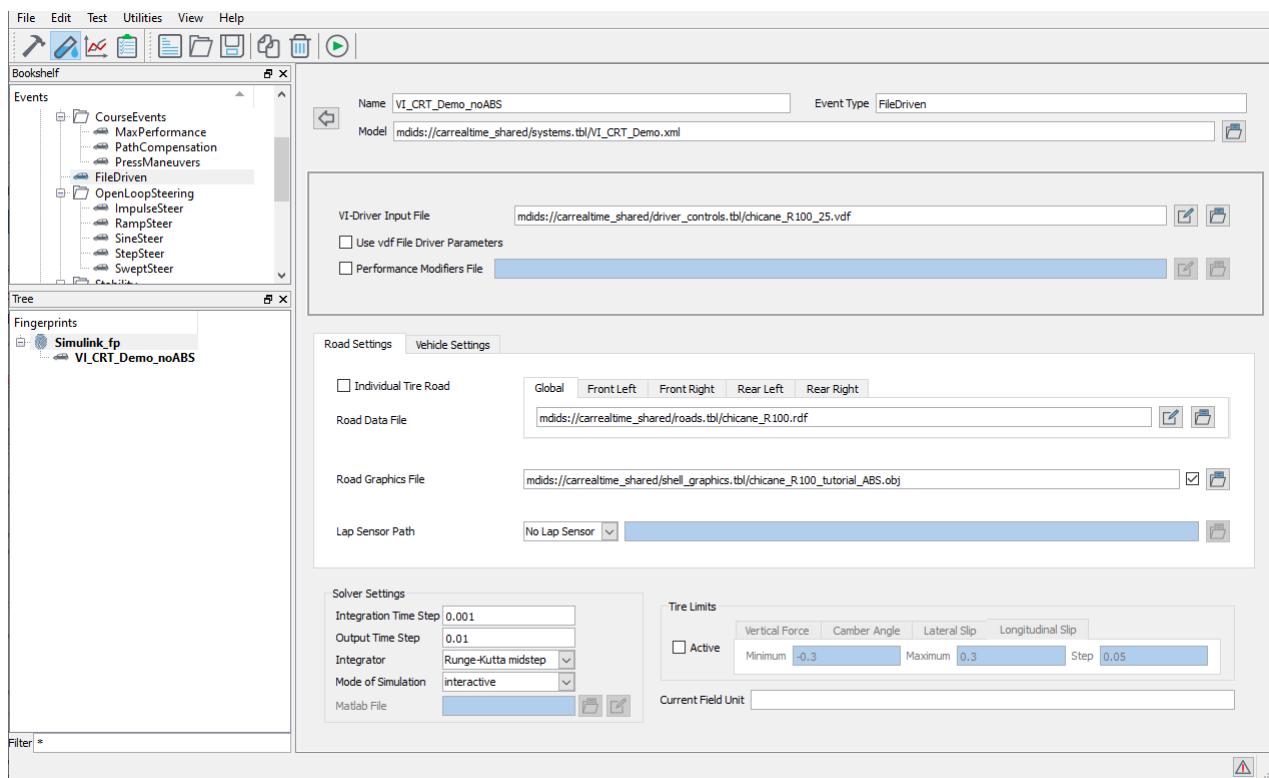
Now, switch to **Test Mode** using the button.

Create a new fingerprint and name it **Simulink_fp**.

Add a new **File Driven** event (`VIDriver -> FileDriven`) to the fingerprint using the **VI_CRT_Demo** model and set the following parameters:

- **chicane_R100_25.vdf** from the `carrealtime_shared` database as the **VI-Driver Input File**;
- **chicane_R100.rdf** from the `carrealtime_shared` database as the **Road Data File**;
- check the **Road Graphics File** checkbox and load the **chicane_R100_ABS_tutorial.obj** file from the `carrealtime_shared` database;
- rename the event **VI_CRT_Demo_noABS** as shown in the picture below.

VI-CarRealTime



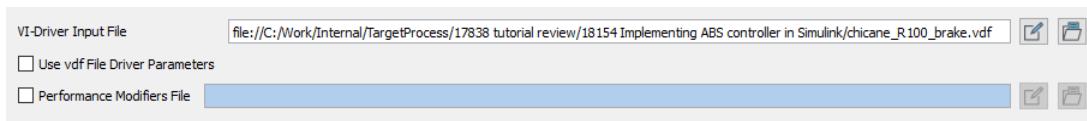
Click the button beside the **VI-Driver Input File** to open VI-EventBuilder and modify the **chicane_R100_25.vdf** vdf event as described below:

- in the **Maneuver_List** block, change the **Maneuver End Time** (abort_time) to **25** seconds;
- in the **Time Stop** block, set the **End Condition** (Ref Value) time to **25** seconds;
- in the **Maneuver_1** block, uncheck the **End of Path** checkbox.
- edit the speed vs. time history in **Speed_Map_1** block (by clicking on the "Edit" button, to the right of "Values"), then fill out the table as shown below:

	x	y
1	0	25
2	50	25
3	70	25
4	80	22
5	100	22
6	350	22
7	400	25
8	450	25
9	500	0
10		

- finally, save the modified vdf file to your working directory, calling it: **chicane_R100_brake.vdf**.

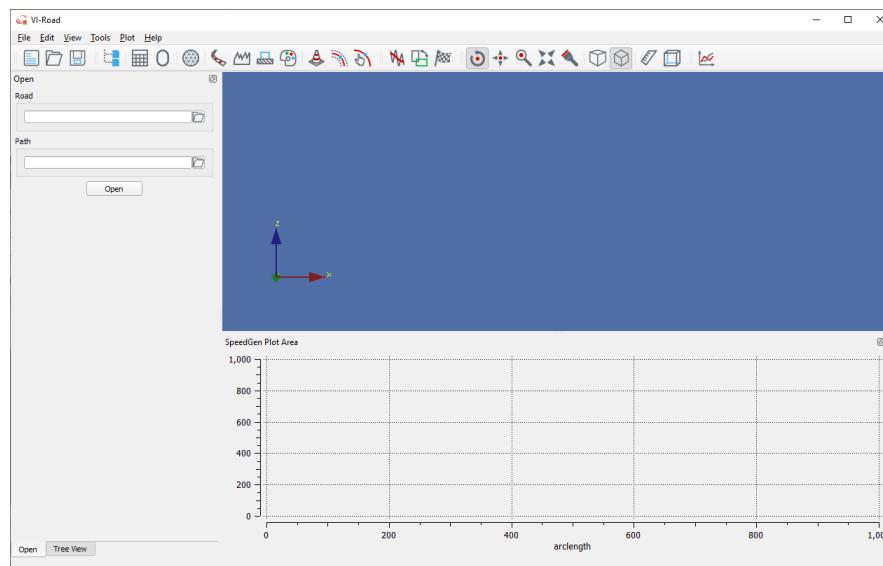
Within the File Driven event that was just set up for this tutorial (e.g. **VI_CRT_Demo_noABS**), point the **VI-Driver Input File** to the event file that was just saved in the working directory, as pictured below.



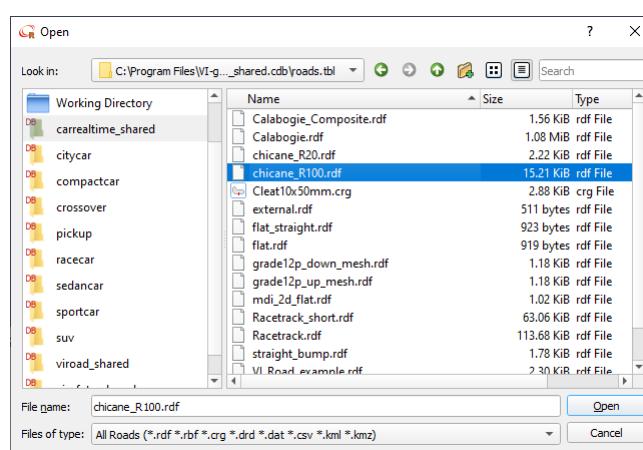
The next step consists in simulating an icy patch on the road, which will be located after the second chicane curve (specifically, on the left side of the road). The "ice" (low grip) road condition will be modeled by decreasing the friction coefficient of the road for the corresponding road section.

In the VI-CarRealTime interface, select **Utilities** and click on **VI-Road**.

VI-Road interface will now show up.

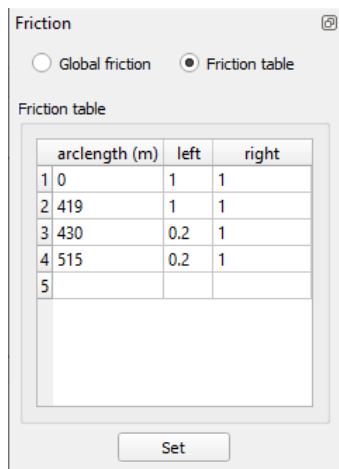


Click on the button closest to the **Road** label (ignore **Path**) and select the **chicane_R100.rdf** file stored in the `carrealtime_shared` database.



Click the **Open** button to load the **Road Data File** into the VI-Road session.

Select the button in the upper VI-Road toolbar, check the **Friction Table** radio button, and fill out the friction table with the particular values shown in the picture below.



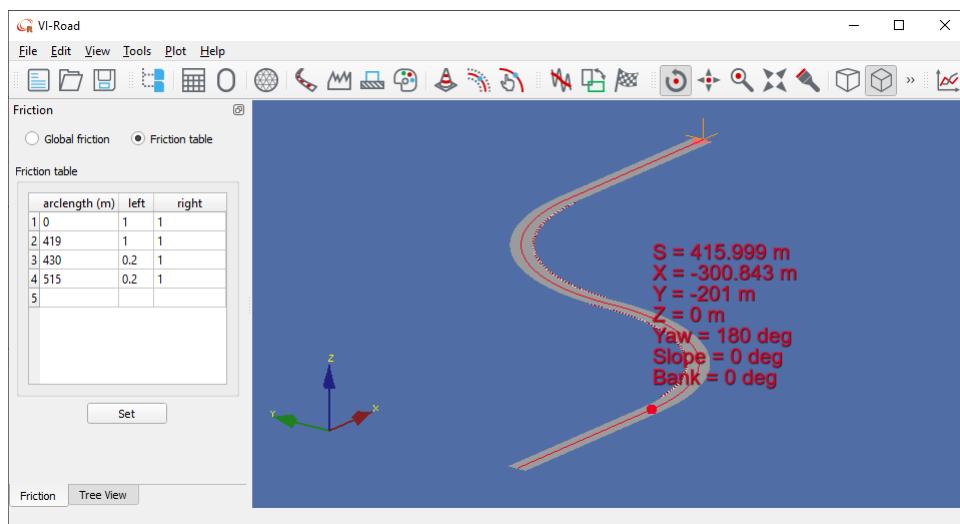
Click the **Set** button to apply the friction changes to the road.

Now, we have a road (chicane) which has, after 419 meters of arc-length, a very low friction coefficient on the left side, which simulates our icy road patch.

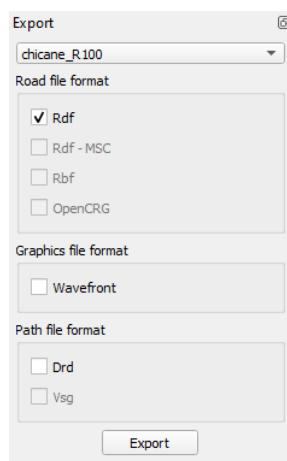
If you want to check where the patch with the lowered friction coefficient starts, you can use the **Show Arclength**



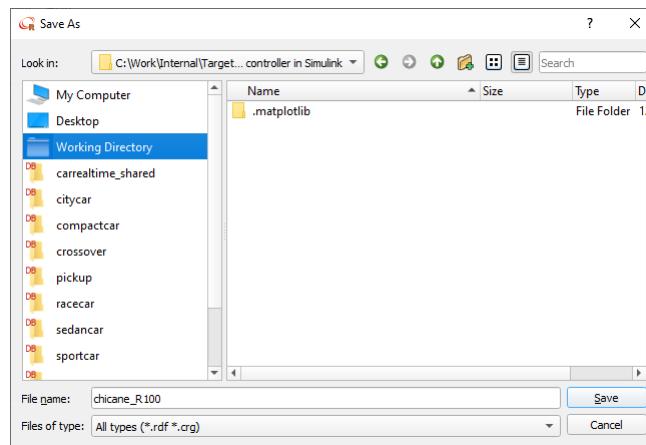
tool, which is in the upper VI-Road toolbar.



Under **File**, select **Export** (or click on the floppy disk icon in the VI-Road toolbar) to save the modified road. Uncheck the **Wavefront** toggle button, because only the edited rdf file itself is needed for the purposes of this tutorial (the road graphics file is already stored in the `carrealtime_shared` database).



Click the **Export** button and save the modified Road Data File in your working directory, as shown below:

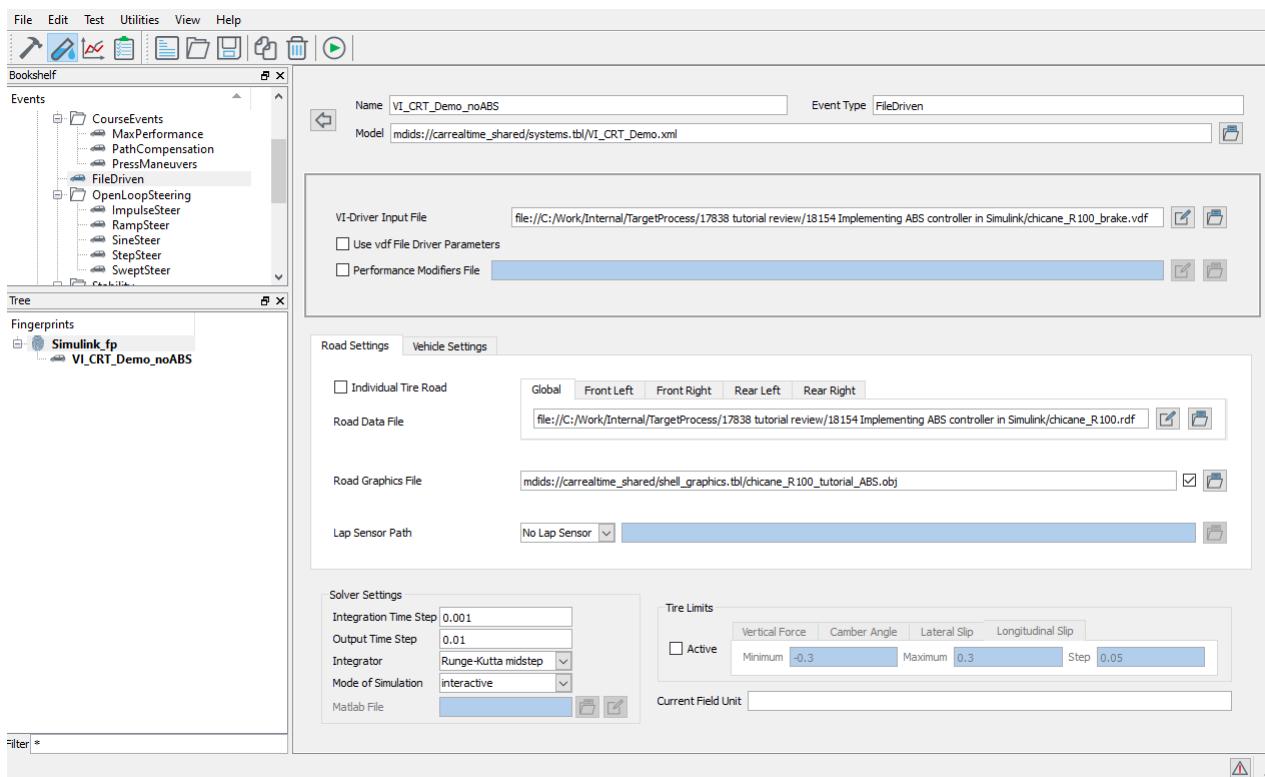


Note: When saving the rdf file, verify that the VI-Road working directory is the same as the VI-CarRealTime one.

Close VI-Road and switch back to the VI-CarRealTime interface.

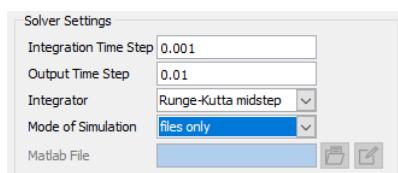
In the **VI_CRT_Demo_noABS** event, point to the rdf file that was just saved as the **Road Data File**, and the **chicane_R100_ABS_tutorial.obj** file stored in the carreatime_shared database as the **Road Graphics File**.

VI-CarRealTime



Save the **Simulink_fp** fingerprint, which now contains all edits and tweaks to the **VI_CRT_Demo_noABS** event.

Run the event in **files only** mode by pressing the button in VI-CarRealTime so that the **svm_send.xml** file will be created, which will be used later for MATLAB and Simulink simulations.

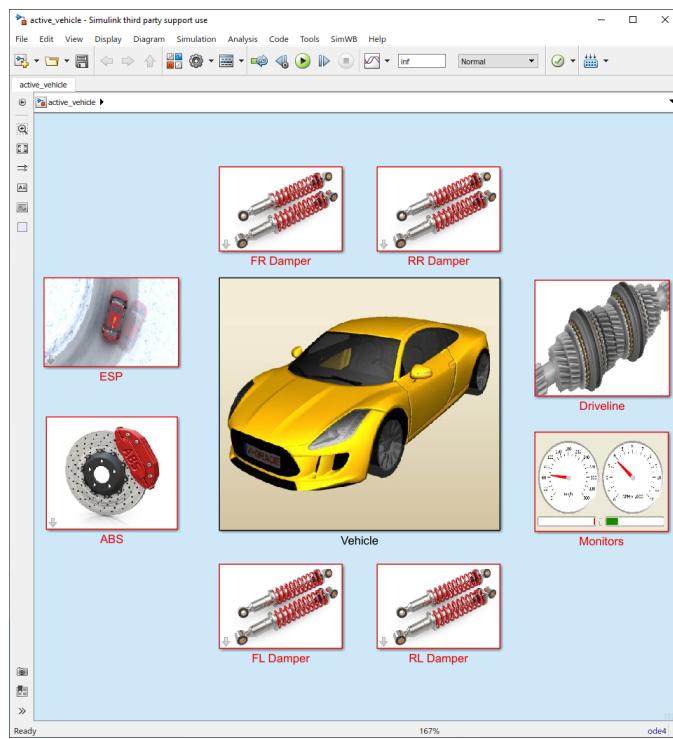


The message in the resulting DOS (cmd) shell informs you that the **VI_CRT_Demo_noABS_send_svm.xml** file has been created in your working directory.

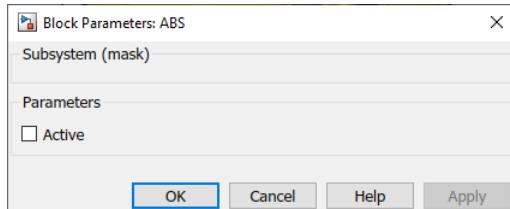
Start **MATLAB**, and therein, set the same working directory as the VI-CarRealTime one.

In MATLAB Command Window, type and enter the command **addpath_vicrt_20** to add all the necessary paths for the co-simulation.

In MATLAB once again, open the following Simulink model: **\$VI-CarRealTimeInstallationdir\acarr\examples\Simulink\active_vehicle.mdl**



To run the first simulation (without the ABS system), we will deactivate the Simulink ABS controller by unchecking the checkbox in the Simulink ABS block (it should already be set to non-active, just double-check it). The menu shown below is summoned by double clicking the ABS mask within the Simulink window.



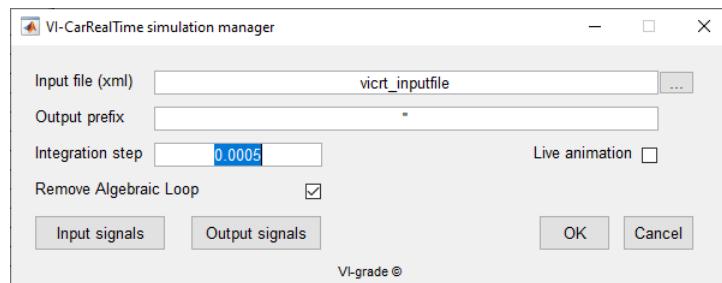
Enter the following command in your MATLAB Command Window:

```
vicrt_inputfile = 'VI_CRT_Demo_noABS_send_svm.xml'
```

This command sets a variable which references the send_svm file that stores all the VI-CarRealTime model information; this variable is used in the VI-CarRealTime Simulink block.

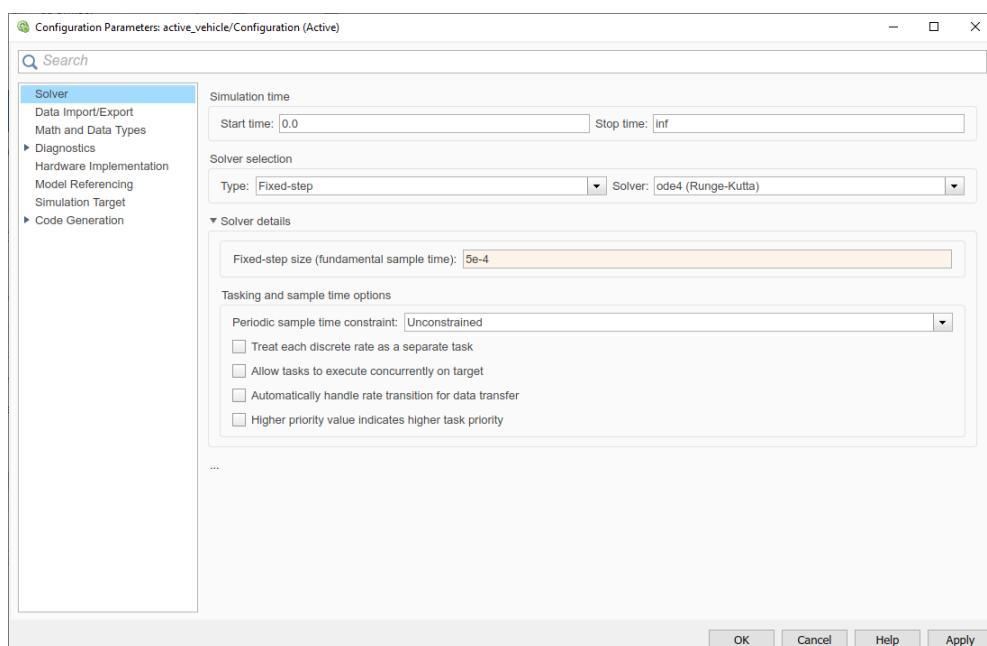
Browse to the VI-CarRealTime Simulation Manager panel (*Active Vehicle -> Vehicle -> VI-CarRealTime* subsystem) and set the Integration step to **0.0005**. This particular menu is summoned by double clicking the Vehicle mask within the Simulink window, and then by double clicking on the Vehicle mask once again in the vehicle submenu.

VI-CarRealTime



Because we've changed the VI-CarRealTime solver Integration Time Step, the Simulink Integration Step must also be set accordingly.

To do so, from the Simulink menu, select **Simulation > Model Configuration Parameters**, and within the **Model Configuration Parameters**, select the **Solver** panel and set **5e-4** as the **Fixed-step size** value.



From your MATLAB environment, start the Simulink simulation with the button in the upper Simulink window toolbar.

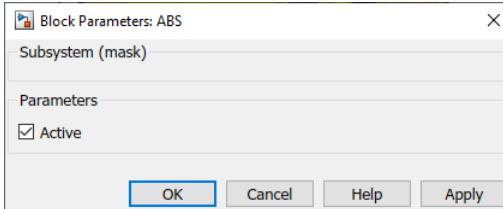
When the Simulink computation starts the appropriate xml file is used to retrieve the model parameters.

By double clicking on the **Monitors** mask in Simulink, it is possible to inspect the relevant plots of vehicle states. Below is a snapshot of the **Brake Pressure Gains** (accessed by double clicking on the **ABS Pressure Gains** mask); the values of these channels are modified by the ABS system, when active. In this particular test case, since the last simulation was run with inactive ABS, they are at a constant value of 1.

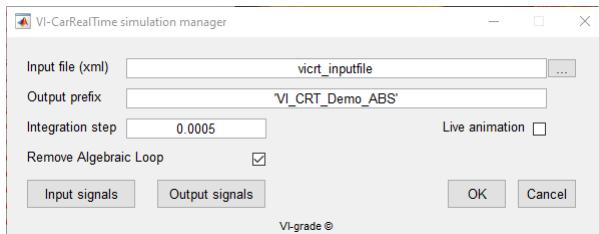


The next step consists in running the same event, using the same model as before, however this time, with the ABS system activated. We can use the same exact `send_svm.xml` file that was just used, but now we have to activate the ABS system model in Simulink.

To do this, go to the **Active Vehicle ABS** subsystem, and turn on the ABS controller by checking the **Active** checkbox in the **Block Parameters: ABS** menu.

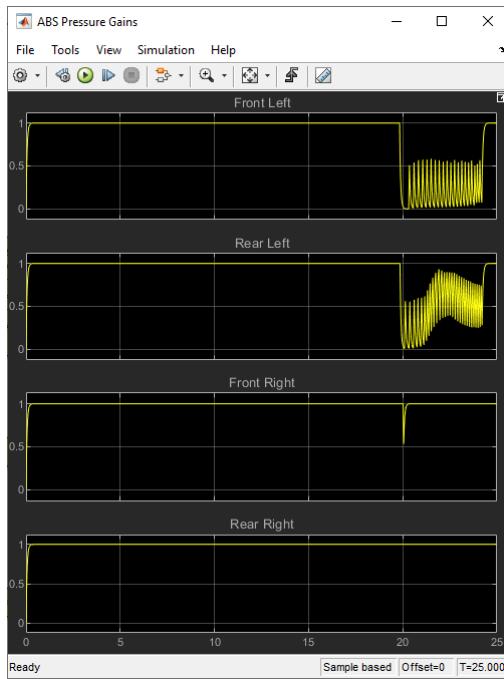


Open the VI-CarRealTime Simulation Manager menu window, which was earlier used to set the Integration step, and enter the following name in the **Output prefix** field: '**'VI_CRT_Demo_ABS'**' (Make sure '**'VI_CRT_Demo_ABS'**' is between apostrophes, otherwise you will get an error in MATLAB when running the simulation).



Start the new co-simulation once again by pressing the  button in the Simulink toolbar.

At the end of computation, the ABS Pressure gains plots should look like the following:



In this case, the ABS system was indeed active, which you can clearly see from the resulting plots above.

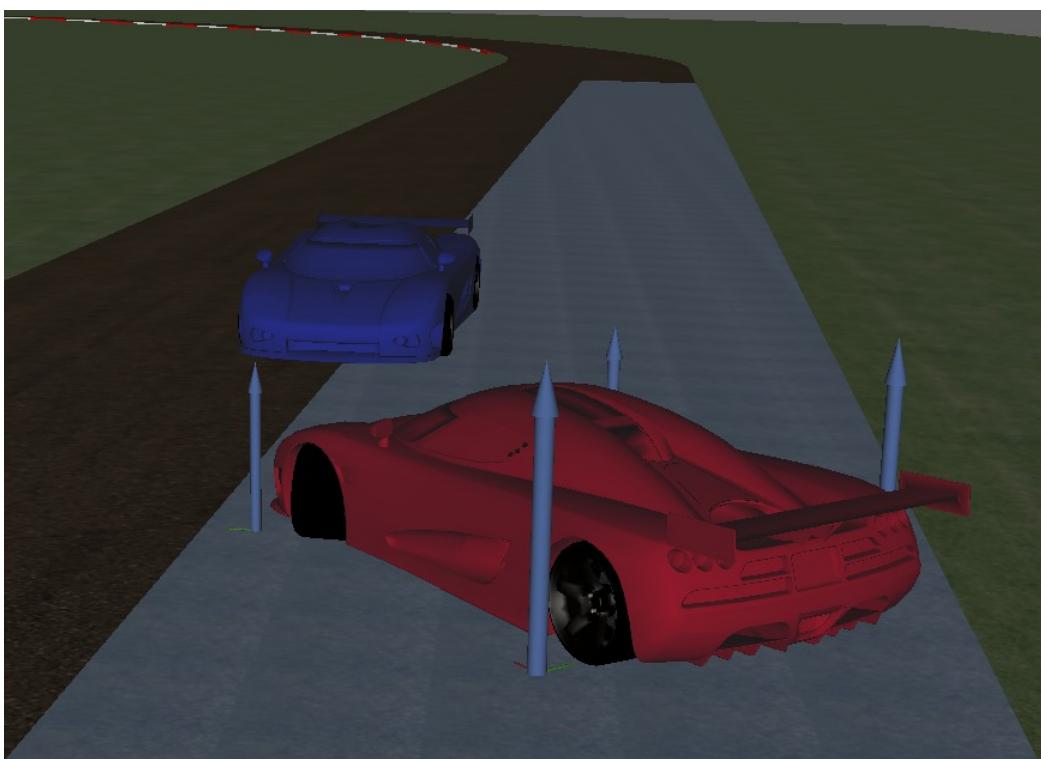
Now, go back to the VI-CarRealTime GUI.

In [Review mode](#) select the **VI_CRT_Demo_noABS** analysis and launch VI-Animator to compare results.

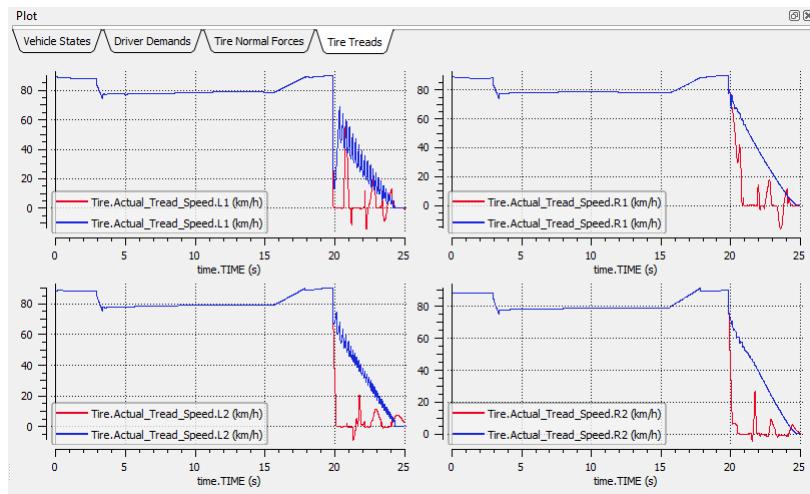
Note: in the VI-CarRealTime Review Mode panel, the **VI_CRT_Demo_ABS** analysis is not present, because the event file was not created in the VI-CarRealTime GUI, but instead was run directly from Simulink by setting an **Output prefix** name. Nonetheless, the res files have been created for both analyses, so in VI-Animator you can also load the **VI_CRT_Demo_ABS.res** to review its animation and plots. In order to add the **VI_CRT_Demo_ABS.res** file into VI-Animator alongside the already present **VI_CRT_Demo_noABS** for comparison purposes, follow the steps described below:

- Right click on the **VI_CRT_Demo_RT GraphicModel** in the left side VI-Animator tree view and select **clone**
- Rename this clone of the **VI_CRT_Demo_RT** GraphicModel to **VI_CRT_Demo_RT_ABS**
- Go to **File->Load Analysis** and open **VI_CRT_Demo_ABS.res** from the working directory
- Right click on **VI_CRT_Demo_RT_ABS** GraphicModel, select **Set Analysis** and click on **VI_CRT_Demo_RT_ABS**.

We can see by viewing both Graphic Models concurrently, that with ABS the vehicle maintains control whereas without ABS the vehicle loses control.



Add a new page, rename it **Tire Treads** and create plots for tire treads longitudinal equivalent velocity vs. time. The dependant variables for the four plots below are named **Tire.Actual_Tread_Speed**, followed by the location of the specific tire (e.g. R1 for right, front) in question.



These plots above show the effectiveness of the ABS system for the test case studied.

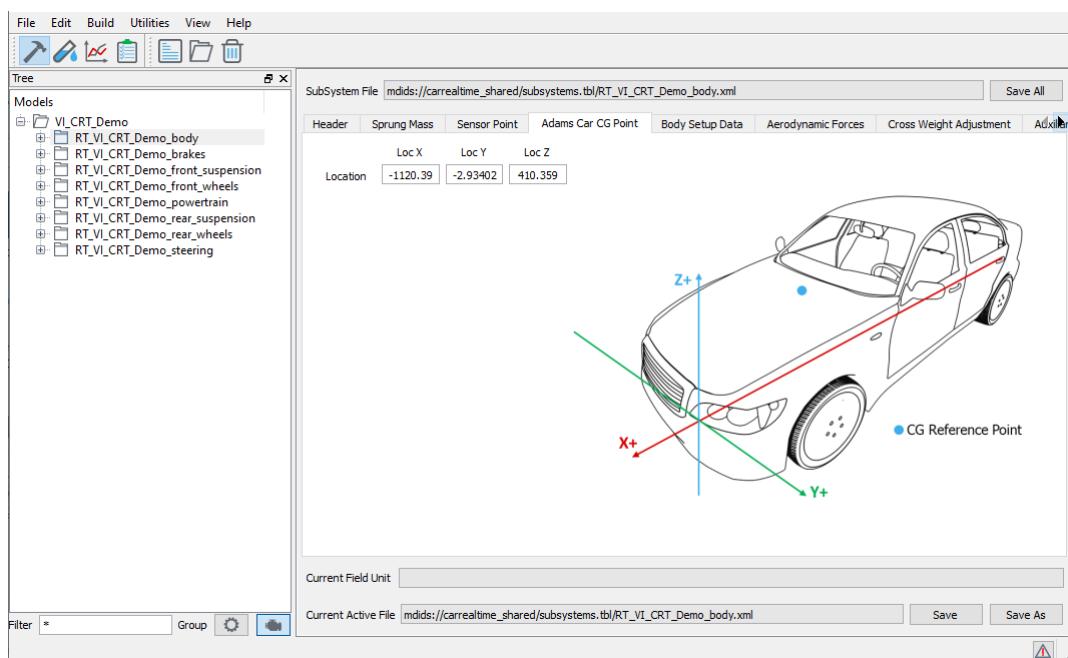
Replacing Standard Components

In this tutorial, you will learn how to use the MATLAB/Simulink interface to replace standard vehicle components, such as dampers, in VI-CarRealTime.

Start a new VI-CarRealTime session using the **vicrt20** command from a DOS (cmd) shell, or use the VI-CarRealTime shortcut from the Windows Start Menu.

VI-CarRealTime

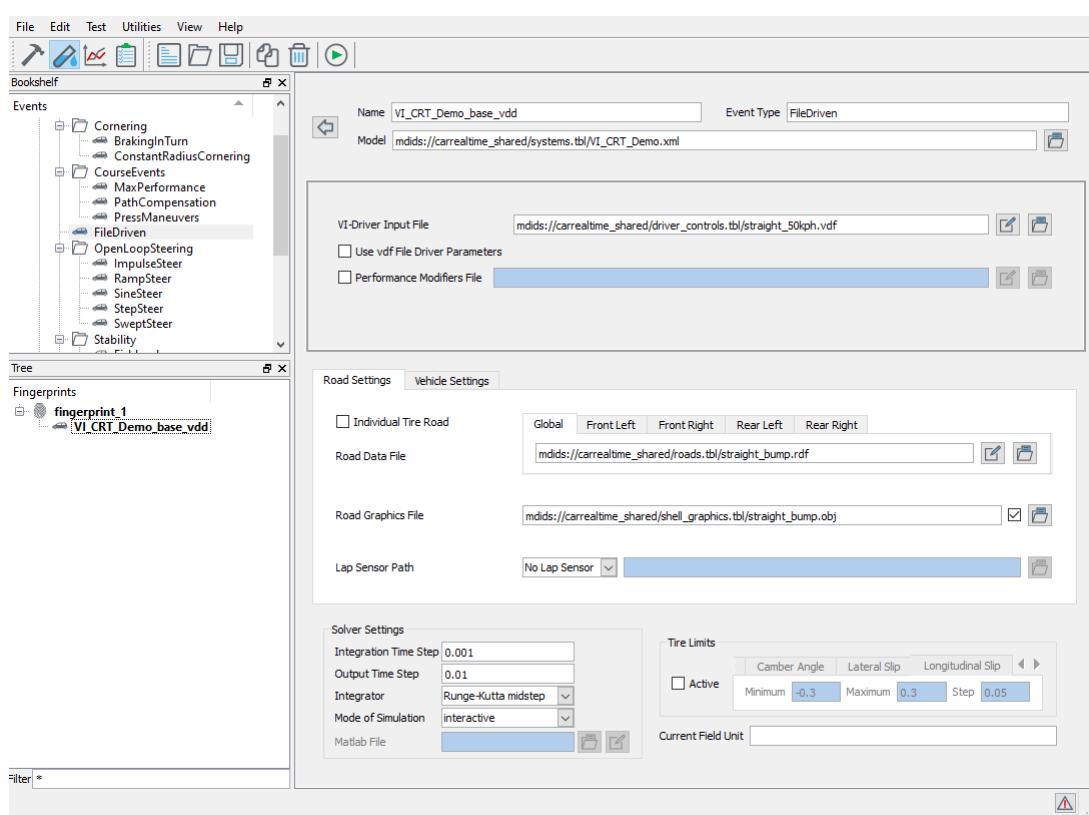
In **Build mode**, open the model **VI_CRT_Demo.xml** from the *carrealtime_shared* database. Make sure that in the **RT_VI_CRT_Demo_body** subsystem, under the **Adams Car CG Point** tab, that your parameters match the ones shown in the picture below.



In **Test Mode**, create a new fingerprint and add a new **File Driven** event to it.

Rename the **File Driven** event to "**VI_CRT_Demo_base_vdd**", and set its parameters as follows:

- VI-Driver Input File: **mdids://carrealtime_shared/driver_controls.tbl/straight_50kph.vdf**
- Road Data File: **mdids://carrealtime_shared/roads.tbl/straight_bump.rdf**
- Road Graphics File: **mdids://carrealtime_shared/shell_graphics.tbl/straight_bump.obj**



Run the event with the button.

```
VI.PTW (VI-CarRealTime Python Task Window)
-----
Static equilibrium has been achieved.
Elapsed Simulation Time: 3.155 (sec).
Elapsed Clock Time: 0.486 (sec).

Performing dynamic simulation...

Simulation Progress (percent complete):
0 50 100
-----
=====
= VI-CarRealTime =
= VEHICLE SIMULATION STATISTICS =
=====

--- Simulation Statistics ---
>> LAP TIME      = 5.000000 sec
>> LAT ACCEL MAX = 0.002207 G
>> LAT ACCEL MIN = -0.001454 G
>> LON ACCEL MAX = 1.522937 G
>> LON ACCEL MIN = -1.981084 G
>> LON SPEED MAX = 50.012419 km/h
>> LON SPEED MIN = 49.015799 km/h
>> LON SPEED AVG = 49.607767 km/h
>> STEERING MAX = 0.000000 rad
>> STEERING MIN = 0.000000 rad

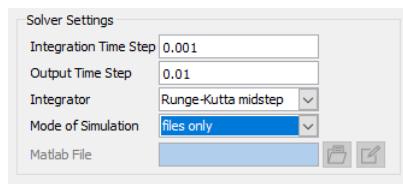
Writing output to file. Please Wait ...

Elapsed Simulation Time: 5.000 (sec).
Elapsed Clock Time: 0.873 (sec).
Computational Efficiency: 5.727 (sim. sec)/(sec).
```

Copy the base event and rename it: "VI_CRT_Demo_non_linear_damper_vdd" (no quotes).

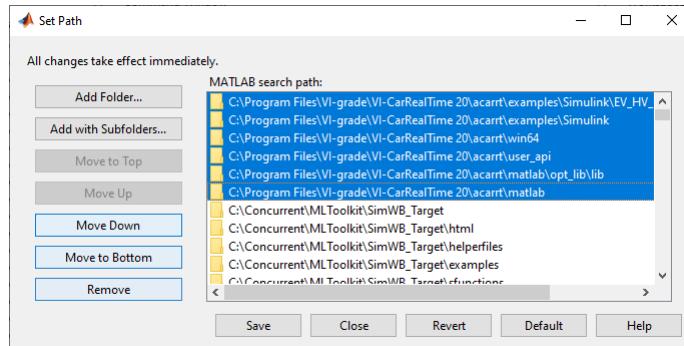
Run this newly created event in **files only** mode (under the **Mode of Simulation** drop down menu in VI-CarRealTime) in order to generate the send_svm file for the co-simulation analysis.

VI-CarRealTime



Start MATLAB, run the **"addpath_vicrt_20"** script (no quotes) from your MATLAB command window to add the VI-CarRealTime libraries to the MATLAB search path, and then save this configuration using the **Set Path** window from MATLAB's toolbar.

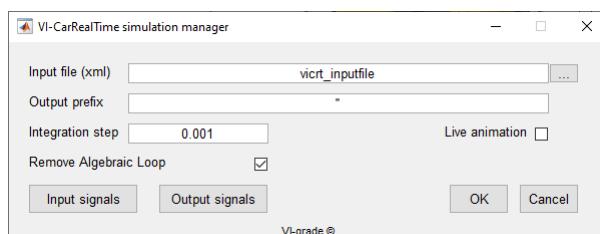
Note: If you've already completed the previous "Implementing ABS in Simulink" exercise, you may disregard the paragraph above.



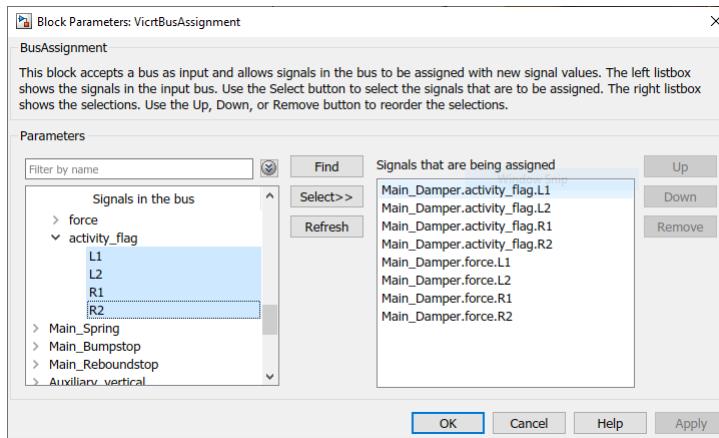
Open the Simulink model file named: **\$VI-CarRealTimeInstallationdir\acarr\examples\Simulink\damper.mdl**

The particular Simulink model that just opened implements vehicle dampers using a spline lookup table and a linear damper. Since user inputs to a vehicle model in VI-CarRealTime are generally additive, we will need to deactivate VI-CarRealTime damper properties before proceeding with the exercise.

To deactivate the relevant VI-CarRealTime damper properties, double-click on the vehicle damper.mdl subsystem mask (in other words, double-click on the car icon in the Simulink window). The following window shows up:

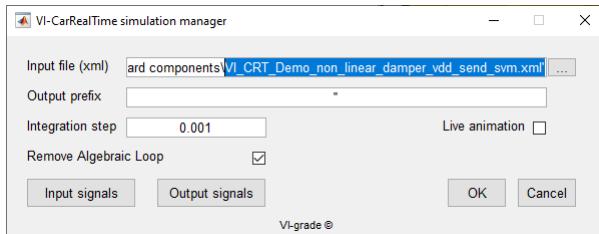


Click on the **Input signals** button to open the **VICrtBusAssignment** menu. On the right side you can see the list of signals that are already assigned. For practice, you could remove the **activity_flag** signals, and then in the **Signals in the bus** list, locate **Main_Damper** and expand its contents, click on and expand the **activity_flag** section, and then select the **L1,L2,R1,R2** inputs together. Next, press the **Select>>** button to add these to the mask inputs, as shown in the picture below:



Press the **OK** button to apply the changes.

Back in the VI-CarRealTime Simulation Manager window, click on the **...** button, browse the working directory and select the **VI_CRT_Demo_non_linear_damper_vdd_send_svm.xml** send file, which is needed for the upcoming co-simulation.



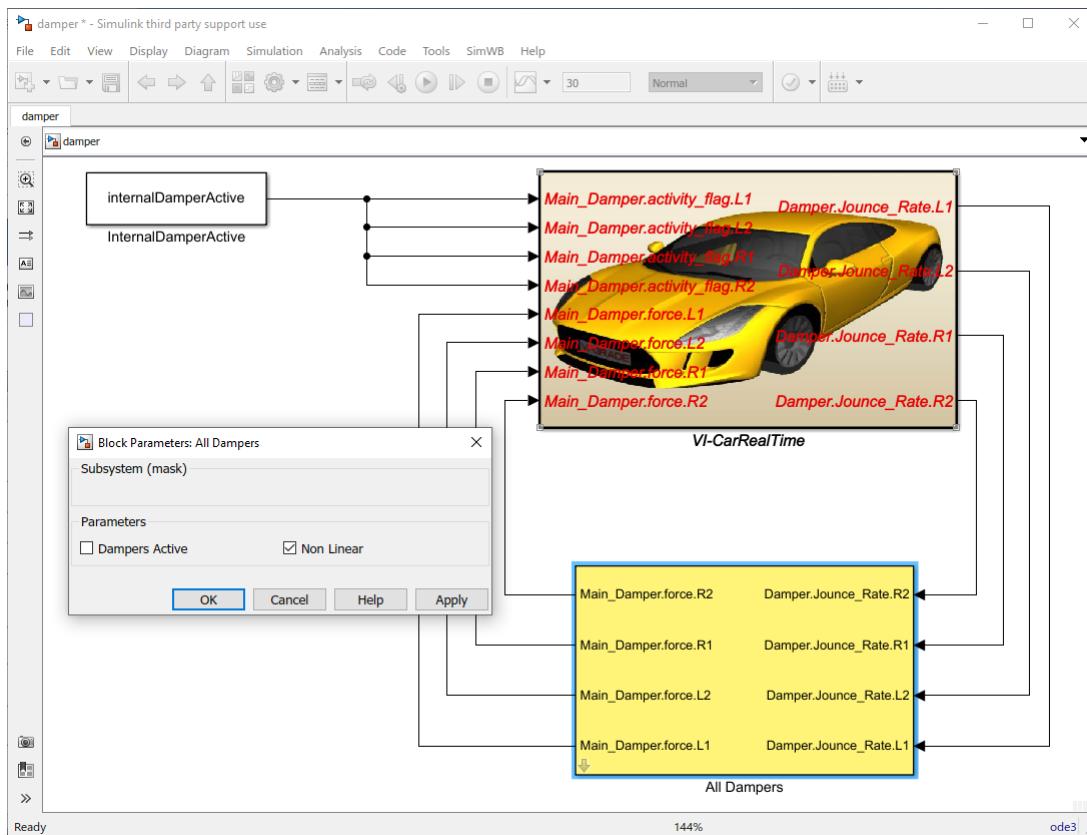
Click on the **OK** button.

Now, in the Simulink window, you'll see the inputs you just added as shown beneath:

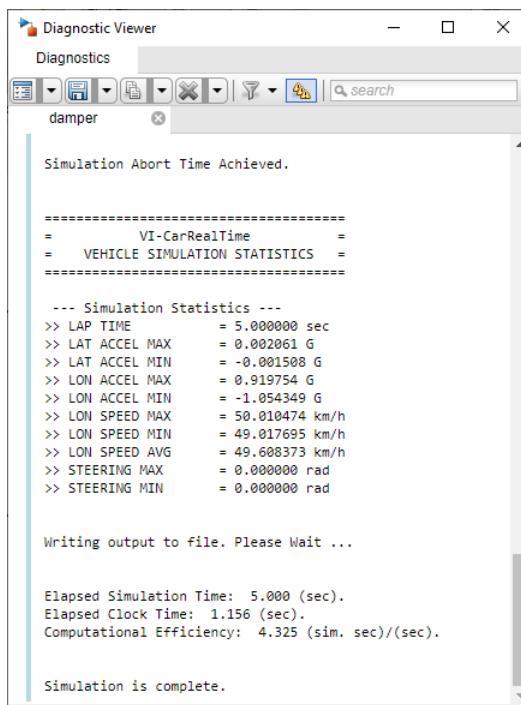


Double clicking on the **All Dampers** block, un-check the **Dampers Active** parameter and press **OK**. This will set the **internalDamperActive** constant equal to **0**, so that during the co-simulation analysis, the internal VI-CarRealTime damper configuration settings will not be used. Instead, the damper force will be managed only by external Simulink damper models.

VI-CarRealTime



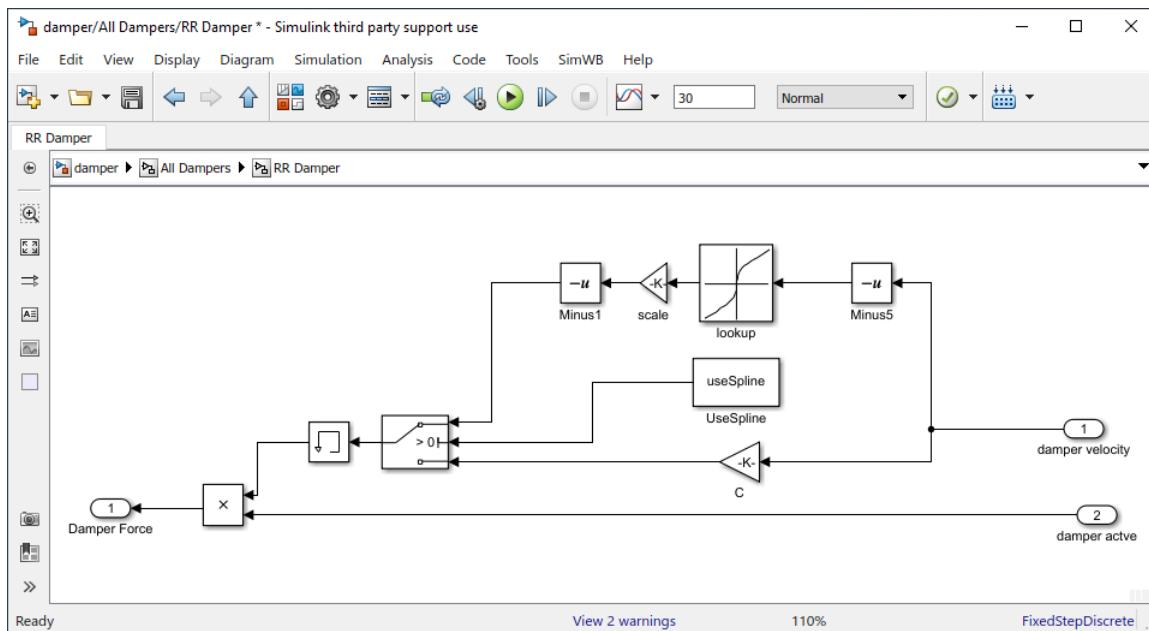
The co-simulation is now ready to run. Launch it by pressing the button in Simulink's toolbar, and wait until the analysis is complete.



When the simulation is complete, a result file will be created in your working directory.

Next, we will modify the Simulink damper characteristics in all four dampers to make the four dampers linear. To do this, double click on the **All Dampers** block and un-check the **Non Linear** parameter and press **OK**.

Looking under the mask, and then opening the **RR Damper** subsystem, you can see that this will move the switch to receive the linear damper behavior.



In **VI-CarRealTime's Test Mode**, copy the **VI_CRT_Demo_base_vdd** event and rename: "**VI_CRT_Demo_linear_damper_vdd**" (no quotes).

Run the event that you just created in files only mode to create the send file for Simulink.

```
VI.PTW (VI-CarRealTime Python Task Window)
>>> run ()
Running Simulation in files only mode ....

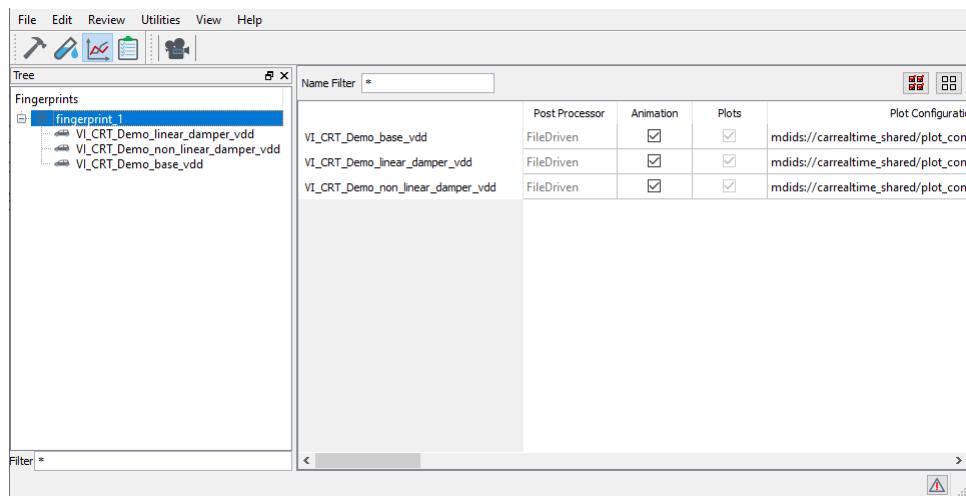
File VI_CRT_Demo_linear_damper_vdd_send_svm.xml was successfully created.
```

Switch back to your active Simulink window, double-click the vehicle mask to open the VI-CarRealTime Simulation Manager window, and set the freshly created **VI_CRT_Demo_linear_damper_vdd_send_svm.xml** send file as

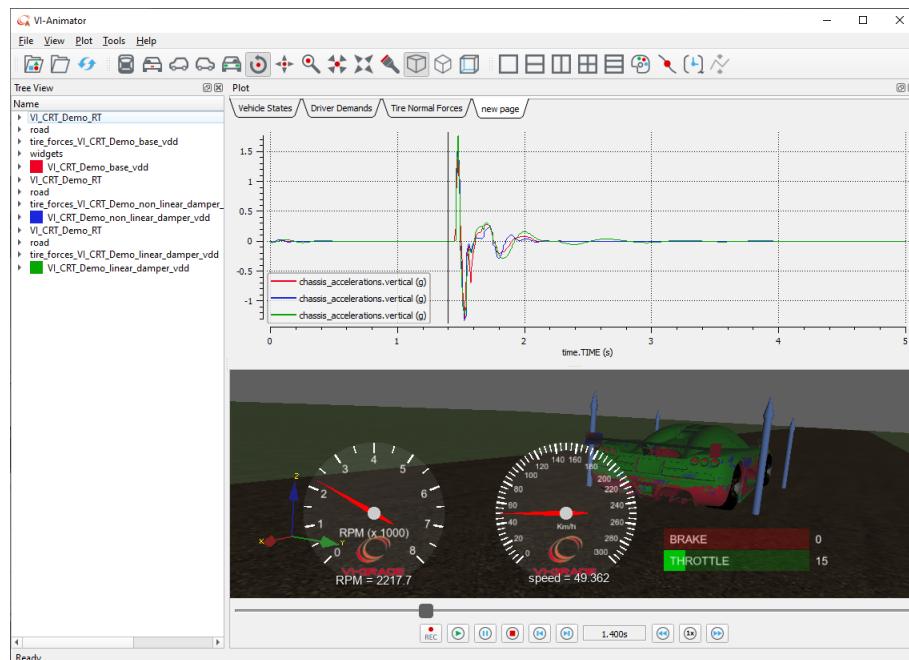
the **Input File**. Press the **OK** button, and then run the Simulink model with the button. This particular model contains the external (Simulink) linear dampers.

In **VI-CarRealTime**, save the fingerprint you've been working with for this exercise. Switch to **Review Mode**, select the three events, and then run postprocessing in VI-Animator by clicking the button.

VI-CarRealTime



Create a new tab in the plot window to plot **chassis vertical accelerations** for all three events.

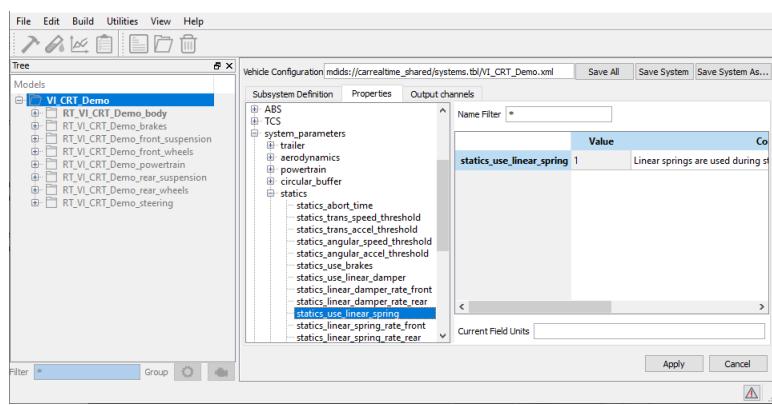


Additionally, you could plot other relevant channels, such as **damper forces vs time**.

Note: It is important to notice that Simulink inputs are only fed into the VI-CarRealTime model during dynamic analyses. This means that for a particular type of force, the need to remove pre-existing internal components (e.g. dampers) would make a particular component apply a force equal to zero during static analyses; this behavior could cause convergence problems.

A typical case is the replacement of a VI-CarRealTime spring model with an analogous Simulink component; the preload installation method would be applied during static analyses (if set), but the presence of a zero force spring property file would likely cause a static analysis to fail. To resolve this problem, however, it is possible to activate **special internal linear springs**, which are only to be used during static analyses.

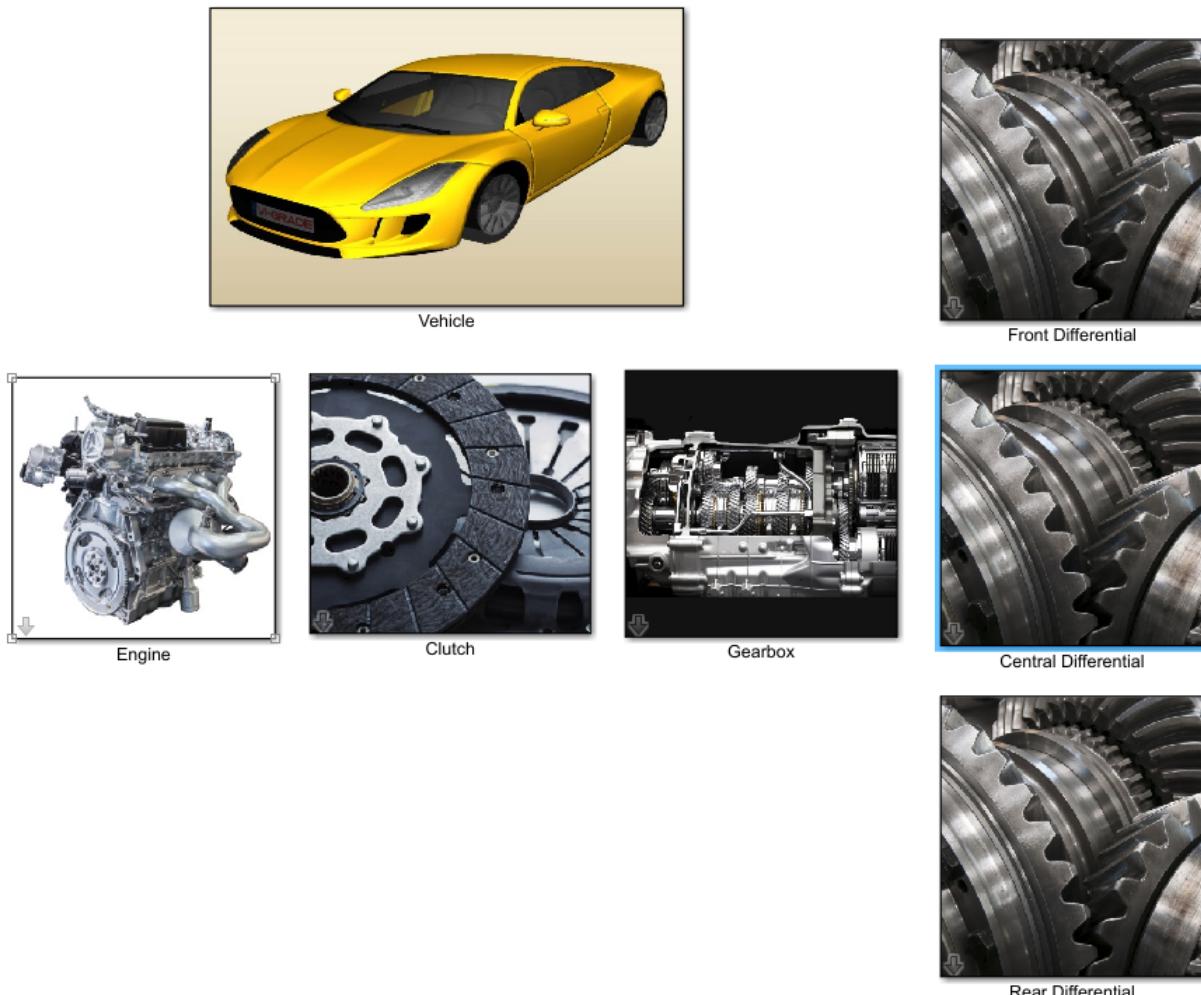
To set up an appropriate property file for springs, to be used within static analyses, simply set the parameter value in **statics_use_linear_springs** to 1 (under the **statics** parameter tree) and then set the appropriate spring rates (see the picture below, which clarifies the aforementioned procedure).



Replacing Driveline Components

In this tutorial, you will learn how to use the MATLAB/Simulink interface to replace any driveline components, from the differentials to the engine, in VI-CarRealTime.

We will refer to the example Simulink model provided with the VI-CarRealTime installation; the simulink model is called **driveline.mdl**, and it is placed in **\$VI-CarRealTime\Installationdir\acarrt\examples\Simulink**



VI-CarRealTime

In this simulink model the driveline of an all wheel drive vehicle is completely replaced, and we will focus mainly on the channels that need to be written externally.

The main concept behind the replacement of a driveline component, especially for the offline simulation, is that VI-Driver needs certain informations to properly estimate the available torque from the engine. To do this, the rpm of the engine must be calculated properly.

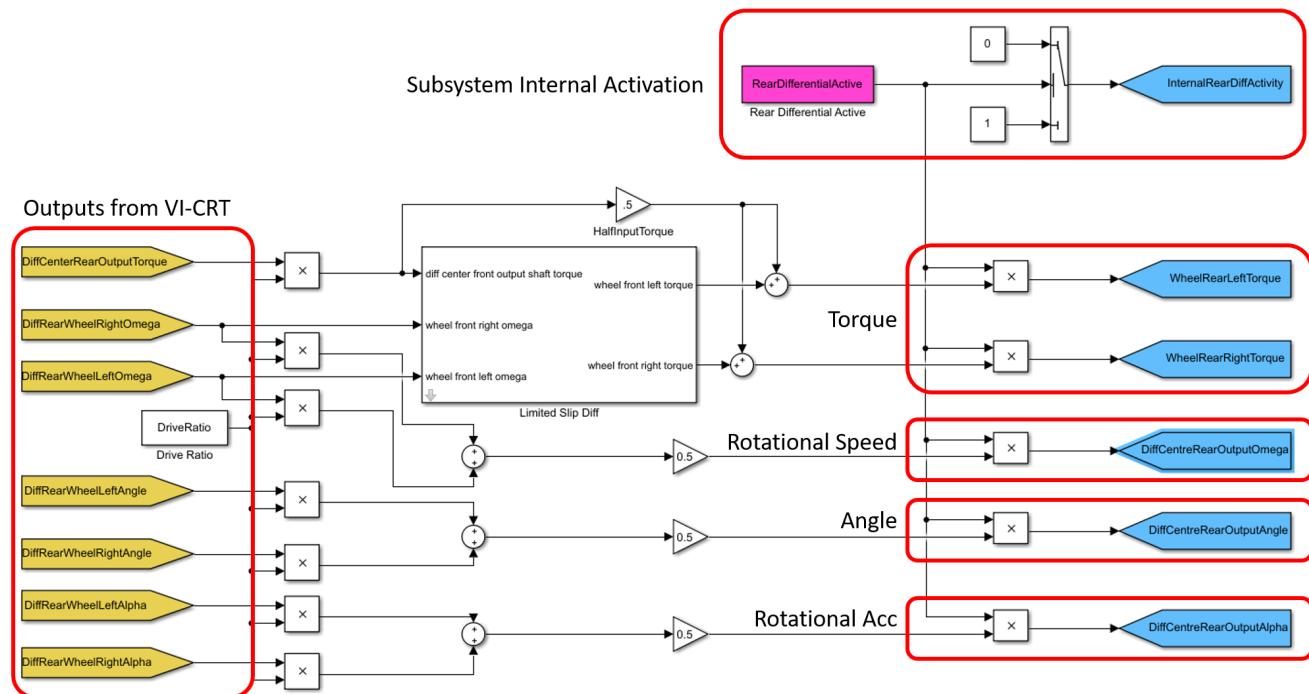
VI-CarRealTime calculates the rpm of the engine starting from the wheel rotational speed; then, considering all the ratios in between, it calculates the different shafts rotational speed of each element between the wheels and engine, such as:

- Rear/front differential
- Central Differential
- Gearbox
- Clutch

and finally the engine rpm. If this chain is broken when replacing a component, then the calculation of the rpms is no longer possible, and wrong results will be given out from the simulation.

Front/Rear Differential

By opening the Front or Rear Differential mask in the simulink model, we can have a look at the input/output channels used to replace the component.



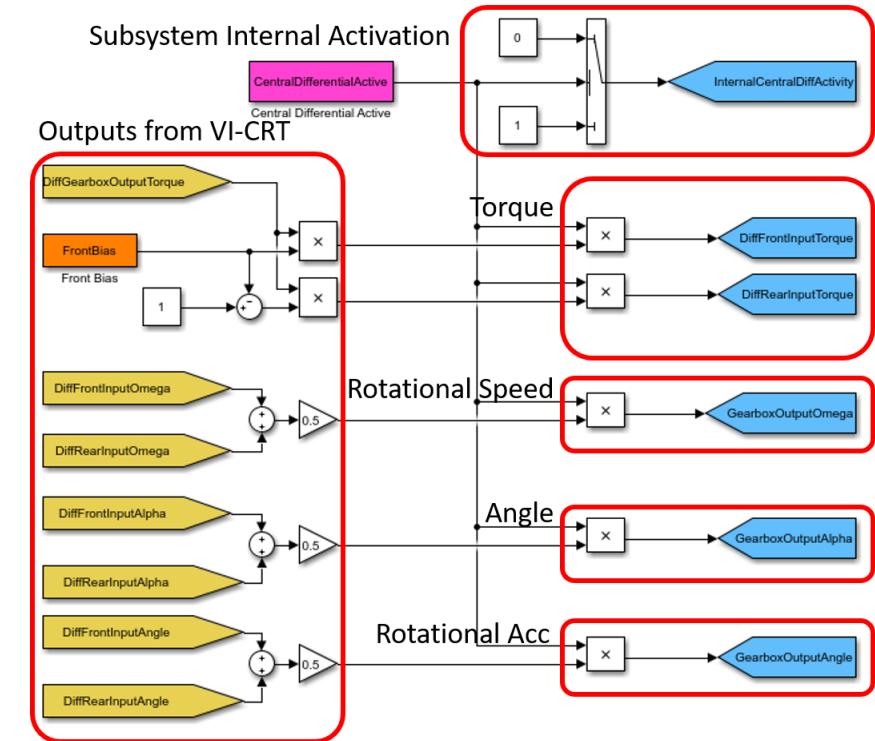
The output channels from VI-CarRealTime are the ones needed to calculate the relatives inputs; looking at the rear differential (the front is exactly the same, but with the front differential channels), these are the input channels that need to be calculated and sent to the VI-CarRealTime mex. Some of them are mandatory, others can be neglected:

- Mandatory inputs channels:
 - **InternalRearDiffActivity**, linked to **Subsystem_Activation.Internal_RearDifferential_Activity**: to activate/deactivate (= 1 or = 0) the internal rear differential
 - **WheelRearLeftTorque**, linked to **Differential_Rear.wheel_rear_left_torque**: torque sent to the left wheel
 - **WheelRearRightTorque**, linked to **Differential_Rear.wheel_rear_right_torque**: torque sent to the right wheel

- **DiffCentreRearOutputOmega**, linked to **Differential_Rear.diff_central_rear_outputshaft_omega**: rotational speed of the element above the replaced one, in this case the rotational speed of the center differential. Note that even if the vehicle model is not an all wheel drive, this is the channel that needs to be written. Even for a front/rear wheel drive vehicle model, VI-CarRealTime internally uses a center differential with all ratios set to 1, no inertia, and torque distribution 100% on the front or rear, depending on the driveline setting.
- Not mandatory inputs channels
 - **DiffCentreRearOutputAlpha**, linked to **Differential_Rear.diff_central_rear_outputshaft_alpha**: angular acceleration of the element above the replaced one, in this case angular acceleration of the center differential. This is used to calculate the inertia contribution.
 - **DiffCentreRearOutputAngle**, linked to **Differential_Rear.diff_central_rear_outputshaft_angle**: absolute value of the angle of the element above the replaced one, in this case the angle of the center differential.

Central Differential

By opening the Central Differential mask in the simulink model, we can have a look at the input/output channels used to replace the component.



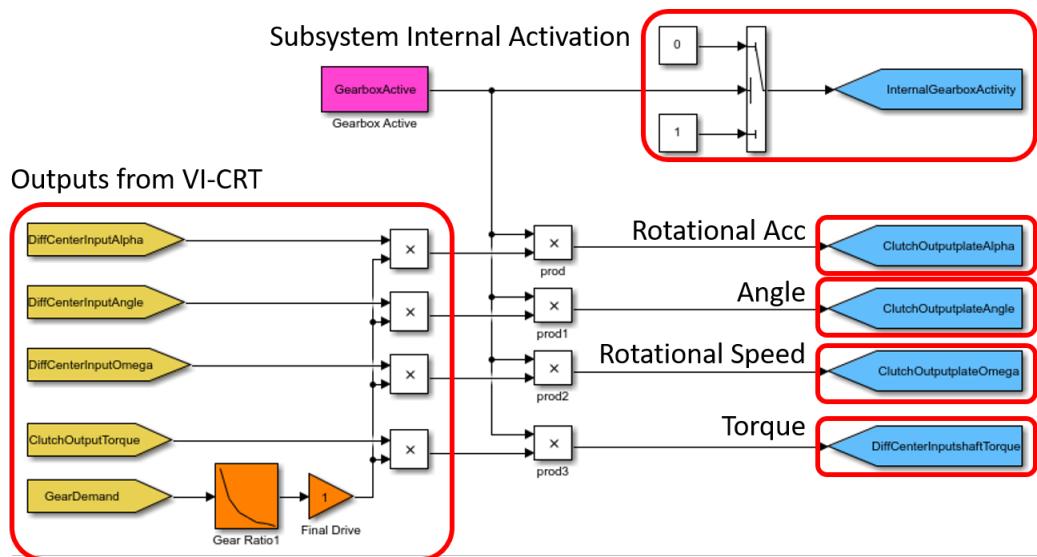
The output channels from VI-CarRealTime are the ones needed to calculate the relatives inputs; looking at the central differential, these are the input channels that need to be calculated and sent to the VI-CarRealTime mex. Some of them are mandatory, others can be neglected:

- Mandatory inputs channels:
 - **InternalCentralDiffActivity**, linked to **Subsystem_Activation.Internal_CentralDifferential_Activity**: to activate/deactivate (= 1 or = 0) the internal central differential
 - **DiffFrontInputTorque**, linked to **Differential_Central.diff_front_inputshaft_torque**: torque sent to the front axle
 - **DiffRearInputTorque**, linked to **Differential_Central.diff_rear_inputshaft_torque**: torque sent to the rear axle

- **GearboxOutputOmega**, linked to **Differential_Central.gearbox_outputshaft_omega**: rotational speed of the element above the replaced one, in this case the rotational speed of the gearbox.
- Not mandatory inputs channels
- **GearboxOutputAlpha**, linked to **Differential_Central.gearbox_outputshaft_alpha**: angular acceleration of the element above the replaced one, in this case angular acceleration of the gearbox. This is used to calculate the inertia contribution.
- **GearboxOutputAngle**, linked to **Differential_Central.gearbox_outputshaft_angle**: absolute value of the angle of the element above the replaced one, in this case the angle of the gearbox.

Gearbox

By opening the Gearbox mask in the simulink model, we can have a look at the input/output channels used to replace the component.

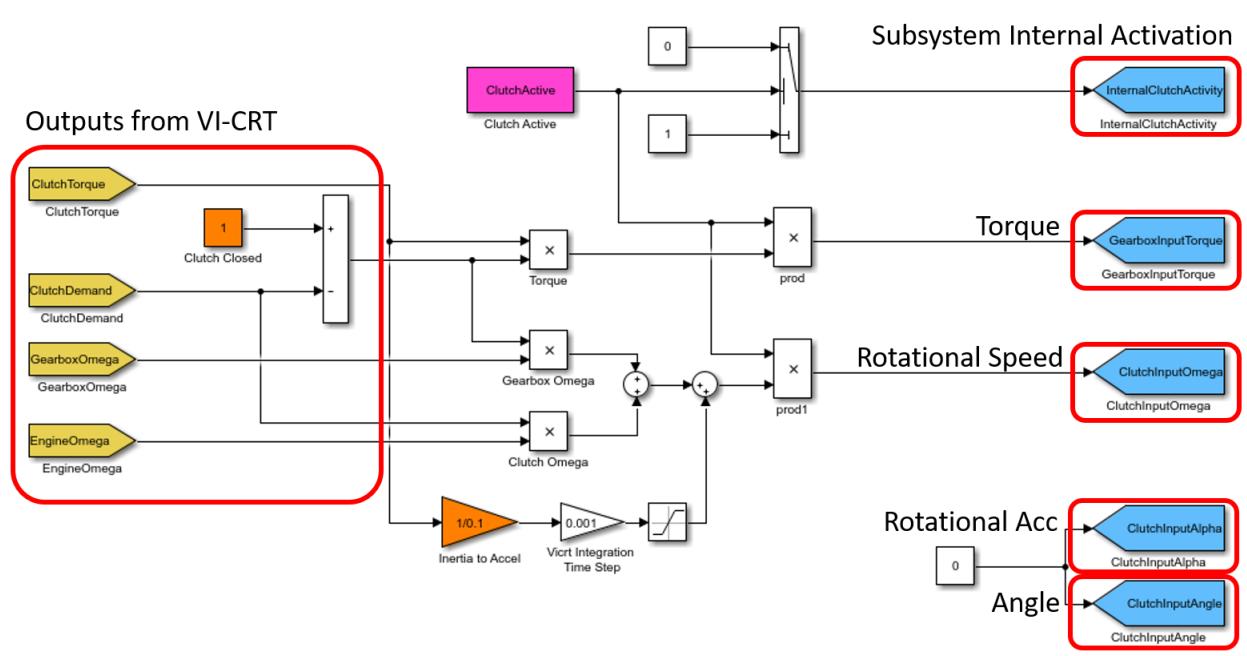


The output channels from VI-CarRealTime are the ones needed to calculate the relatives inputs; looking at the gearbox, these are the input channels that need to be calculated and sent to the VI-CarRealTime mex. Some of them are mandatory, others can be neglected:

- Mandatory inputs channels:
 - **InternalGearboxDiffActivity**, linked to **Subsystem_Activation.Internal_Gearbox_Activity**: to activate/deactivate (= 1 or = 0) the internal gearbox
 - **DiffCenterInputshaftTorque**, linked to **GearBox.diff_central_inputshaft_torque**: torque sent to the central differential. Note that even if the vehicle model is not an all wheel drive, this is the channel that needs to be written. Even for a front/rear wheel drive vehicle model, VI-CarRealTime internally uses a center differential with all ratios set to 1, no inertia, and torque distribution 100% on the front or rear, depending on the driveline setting.
 - **ClutchOutputplateOmega**, linked to **GearBox.clutch_outputplate_omega**: rotational speed of the element above the replaced one, in this case the rotational speed of the clutch.
- Not mandatory inputs channels
 - **ClutchOutputplateAlpha**, linked to **GearBox.clutch_outputplate_alpha**: angular acceleration of the element above the replaced one, in this case angular acceleration of the clutch. This is used to calculate the inertia contribution.
 - **ClutchOutputplateAngle**, linked to **GearBox.clutch_outputplate_angle**: absolute value of the angle of the element above the replaced one, in this case the angle of the clutch.

Clutch

By opening the Clutch mask in the simulink model, we can have a look at the input/output channels used to replace the component.

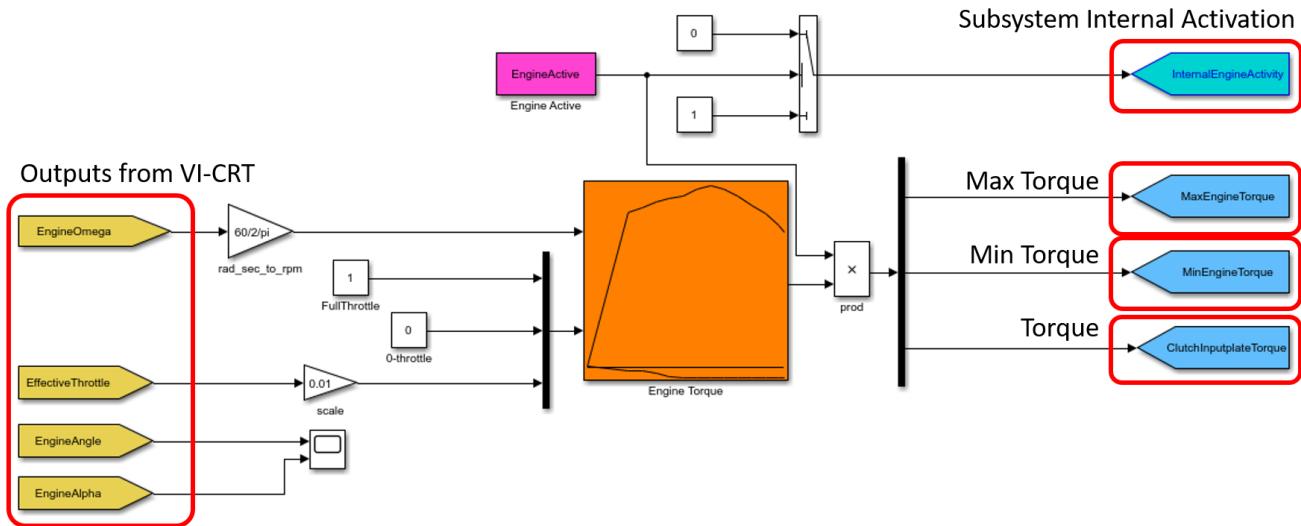


The output channels from VI-CarRealTime are the ones needed to calculate the relatives inputs; looking at the clutch, these are the input channels that need to be calculated and sent to the VI-CarRealTime mex. Some of them are mandatory, others can be neglected:

- Mandatory inputs channels:
 - **InternalClutchActivity**, linked to **Subsystem_Activation.Internal_Clutch_Activity**: to activate/deactivate (= 1 or = 0) the internal clutch
 - **GearboxInputTorque**, linked to **Clutch.gearbox_inputshaft_torque**: torque sent to the gearbox.
 - **ClutchInputOmega**, linked to **Clutch.inputplate_omega**: rotational speed of the shaft entering the clutch.
- Not mandatory inputs channels
 - **ClutchInputAlpha**, linked to **Clutch.inputplate_alpha**: in this case set to 0.
 - **ClutchInputAngle**, linked to **Clutch.inputplate_angle**: in this case set to 0.

Engine

By opening the Engine mask in the simulink model, we can have a look at the input/output channels used to replace the component.



The output channels from VI-CarRealTime are the ones needed to calculate the relatives inputs; looking at the engine, these are the input channels that need to be calculated and sent to the VI-CarRealTime mex. All of these channels are mandatory in order to have the powertrain working properly:

- **InternalEngineActivity**, linked to **Subsystem_Activation.Internal_Engine_Activity**: to activate/deactivate (= 1 or = 0) the internal engine
- **MaxEngineTorque**, linked to **Engine.engine_max_trq**: max torque of the engine
- **MinEngineTorque**, linked to **Engine.engine_min_trq**: min torque of the engine
- **ClutchInputplateTorque**, linked to **Engine.clutch_inputplate_torque**: torque sent to the clutch.

In case of an electric motor, the input channels will be the same as above, but with `Generic_Engine_to_*` prefix, depending where the motor is placed in the driveline.

In general, when using an external driveline, the vehicle model in VI-CarRealTime should be created with the actual driveline layout that will be then replaced with the external model. This will allow to select the correct input channels from VI-CarRealTime Simulation Manager in Simulink, after the send.xml file is selected.

Assessing Vehicle Active Systems

The following tutorial demonstrates the simulation of active components in VI-CarRealTime using MATLAB/Simulink.

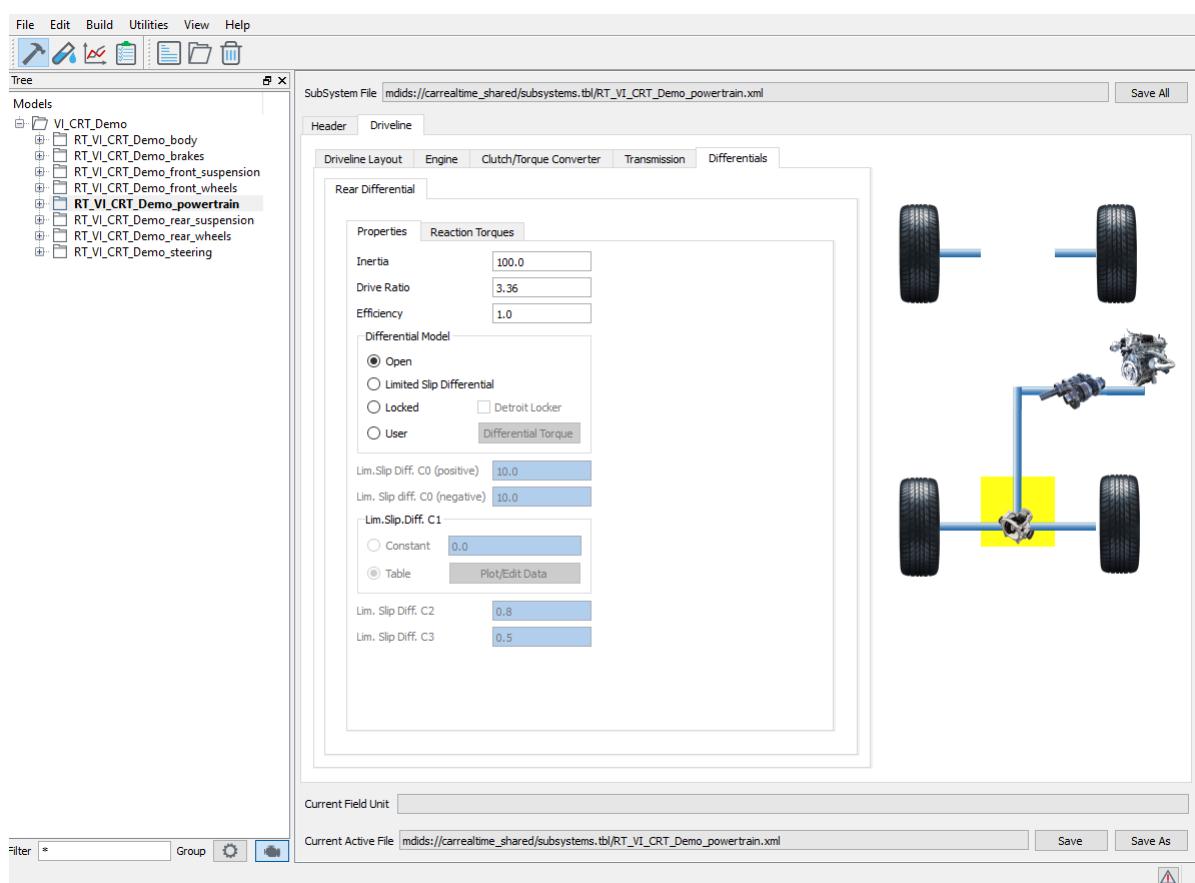
A comparison of vehicle performance will be completed for the following components configurations:

1. A passive vehicle
2. An active vehicle (ESP+semi-active dampers + engine + clutch + gearbox + limited slip differential)

In order to study the influence of active systems on vehicle dynamics, the usual **VI_CRT_Demo** vehicle will be used as test case.

Start a new VI-CarRealTime session and load the **VI_CRT_Demo** vehicle model from the `carrealtime_shared` database, and then follow the steps below:

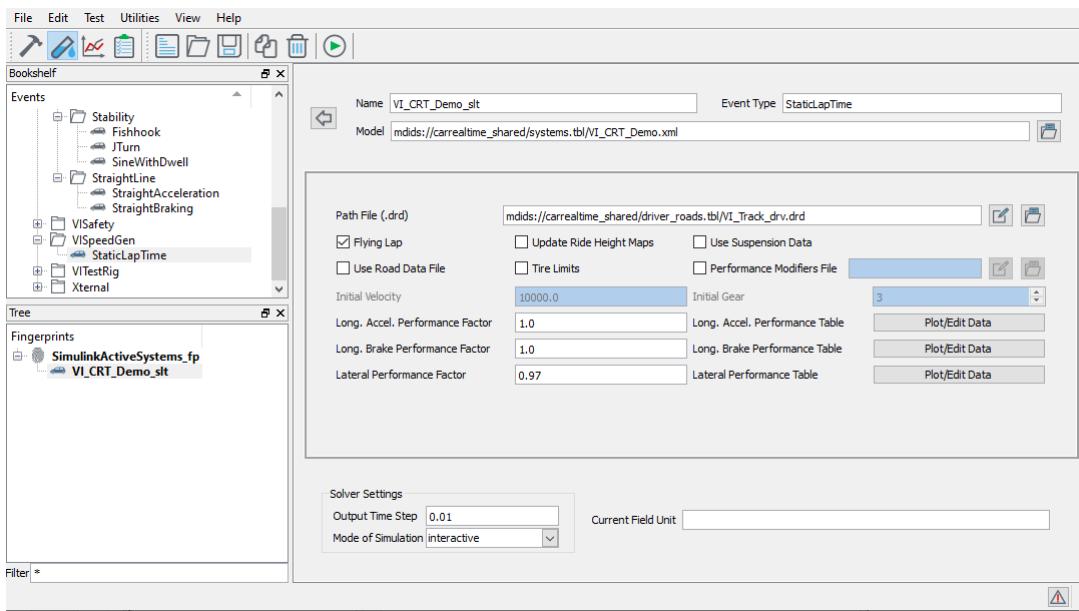
- in **Build Mode**, browse to powertrain subsystem and select **Differentials** panel and select **Open as Differential Model**.



Note: because the aim consists of comparing the performance of the passive vehicle (without controllers) with an active vehicle having a limited slip differential (LSD), the vehicle differential model is set to Open. Open differential in VI-CarRealTime model can be useful in case of a simple differential model in Simulink that only moves a certain amount of the torque from a wheel to another: in order to completely substitute the internal VI-CarRealTime differential is necessary to calculate the complete amount of torque to provide to the single wheel and the correct rotational velocity of the driveline before the differential.

- Switch to **Test mode** using the  button
- Create a new fingerprint, and name it **SimulinkActiveSystems_fp**
- Add a new **StaticLapTime** (under the VI-SpeedGen section, within the Events tree) event named **VI_CRT_Demo_VITrack_spg** to the fingerprint using the **VI_CRT_Demo** model
- Set the event parameters as shown below:

VI-CarRealTime



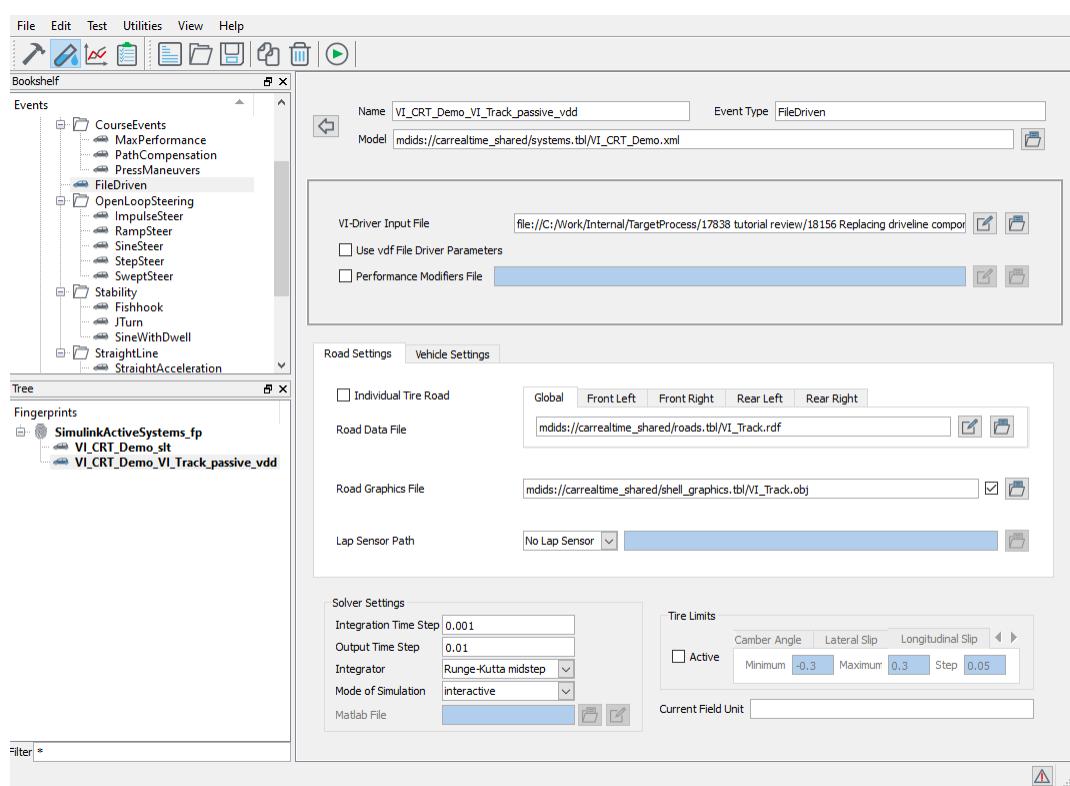
Run the event that you just created by selecting it in the **Test Mode** Fingerprints tree view, and by clicking on the button in the toolbar.

```
VI.PTW (VI-CarRealTime Python Task Window)
Reading Model from 'VI_CRT_Demo_slt.xsg' file...
FLAT road model has been initialized...
initializing VI-TIRE Pacejka 96/02 model... OK

Calculating Speed Profile...
Speed Profile generated. Dump to file VI_CRT_Demo_slt.res
Calculated Lap Time = 55.367822 s
.... Creating VI-Driver event file ....
File VI_CRT_Demo_slt_vdd_event.vdf successfully created.
```

Now, create a **FileDriven** event using the **VI_CRT_Demo** model, and then point the VI-Driver Input File field to the vdf file (**VI_CRT_Demo_VITrack_spg_vdd_event.vdf**) that was just created in your working directory during the previous simulation.

Rename the event **VI_CRT_Demo_VI_Track_passive_vdd** and fill the event editor parameters as shown below:



Run the event with the button.

```

VI.PTW (VI-CarRealTime Python Task Window)
Elapsed Simulation Time: 4.355 (sec).
Elapsed Clock Time: 0.707 (sec).

Performing dynamic simulation...

Simulation Progress (percent complete):
0          50          100
-----
-- INFO --
-- INFO -- <<< VI-Driver MANEUVER INFO >>>
-- INFO -- Maneuver 1 (MANEUVER_1) end of path reached ...
-- INFO -- (maneuver time = 55.79s / simulation time = 55.79s)
-- INFO --
-- INFO --
-- INFO --
-- INFO -- <<< VI-Driver MANEUVER INFO >>>
-- INFO -- == LAST MANEUVER COMPLETED ==
-- INFO -- == DRIVER HAS BEEN DEACTIVATED ==
-- INFO --
-- INFO --

A Driving-Machine end condition was triggered: simulation
terminated prior to simulation abort time.

=====
= VI-CarRealTime =
= VEHICLE SIMULATION STATISTICS =
=====

--- Simulation Statistics ---
>> LAP TIME      = 55.791000 sec
>> LAT ACCEL MAX = 1.167784 G
>> LAT ACCEL MIN = -1.170056 G
>> LON ACCEL MAX = 0.909966 G
>> LON ACCEL MIN = -1.753669 G
>> LON SPEED MAX = 210.425780 km/h
>> LON SPEED MIN = 62.299666 km/h
>> LON SPEED AVG = 109.752349 km/h
>> STEERING MAX  = 4.253720 rad
>> STEERING MIN  = -2.370151 rad

Writing output to file. Please Wait ...

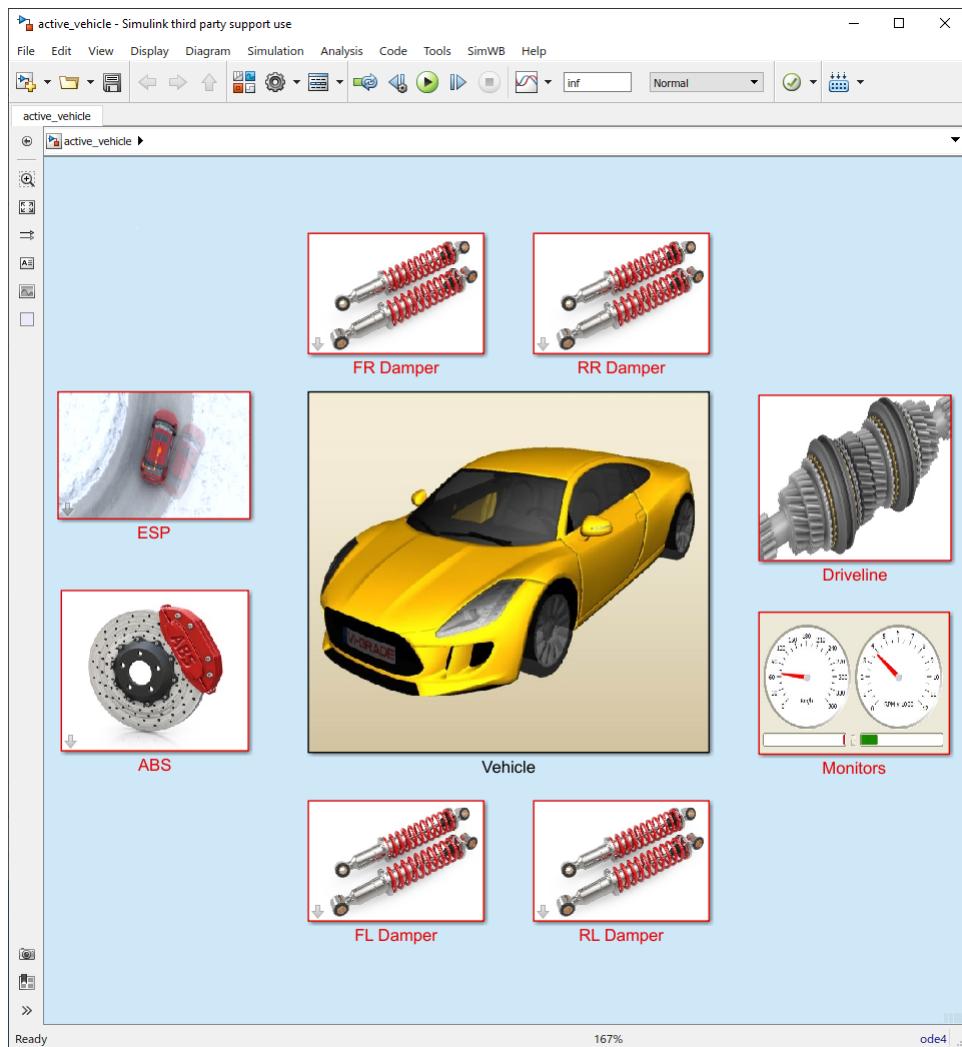
Elapsed Simulation Time: 55.791 (sec).
Elapsed Clock Time: 10.438 (sec).
Computational Efficiency: 5.345 (sim. sec)/(sec).

Simulation is complete.
-- WARNING -- During analysis some spline extrapolations occur.

```

Now, start **MATLAB**, and change its working directory so that it matches VI-CarRealTime's working directory. Run the "**addpath_vicrt_20**" script from your MATLAB command window to add the VI-CarRealTime libraries to your MATLAB search path (if they have not already been added in earlier exercises).

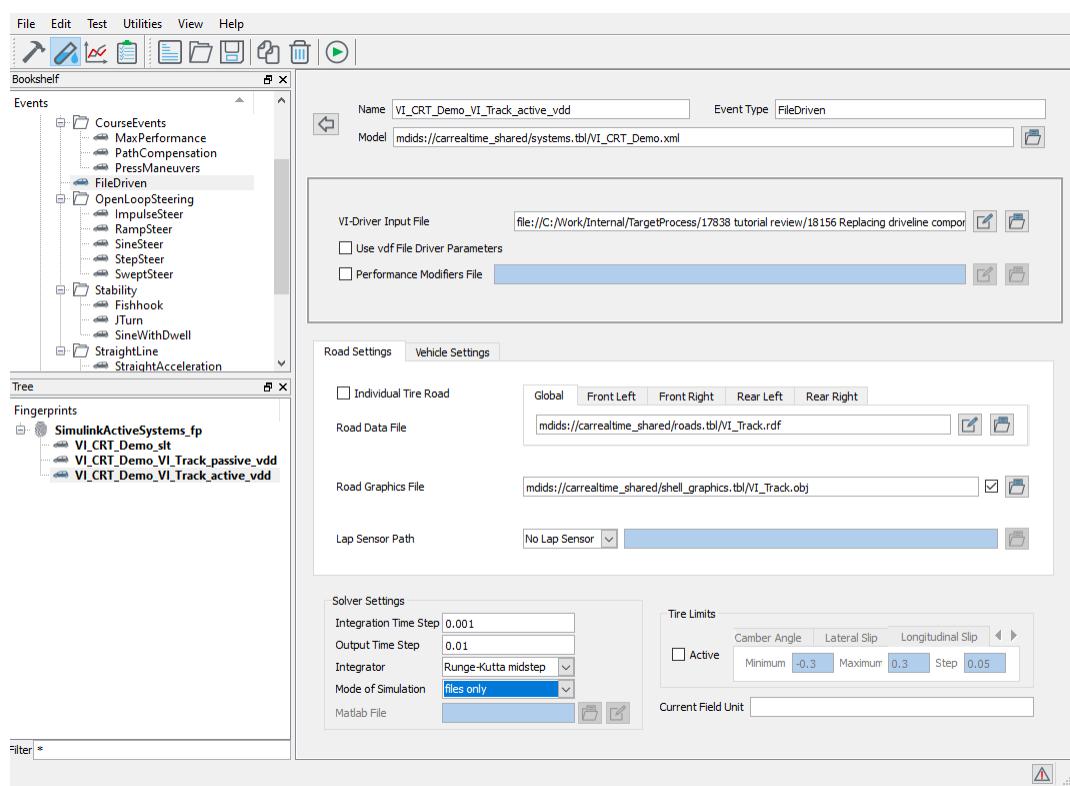
Load the **\$VI-CarRealTime\Installationdir\acarrt\examples\Simulink\active_vehicle.mdl** model.



The **\$VI-CarRealTime\Installationdir\acarrt\examples\Simulink\active_vehicle.mdl** model contains several Simulink subsystems, namely:

- ESP
- ABS
- Semi-Active dampers
- Driveline (including engine, clutch, gearbox and limited slip differential as separate blocks)

Switch back to the VI-CarRealTime window, and also switch to **Test mode**. Copy the existing **VI_CRT_Demo_VI_Track_passive_vdd** FileDriven event, and rename it **VI_CRT_Demo_VI_Track_active_vdd**.



Run the analysis in **files only** mode in order to generate the necessary send file for the upcoming co-simulation.

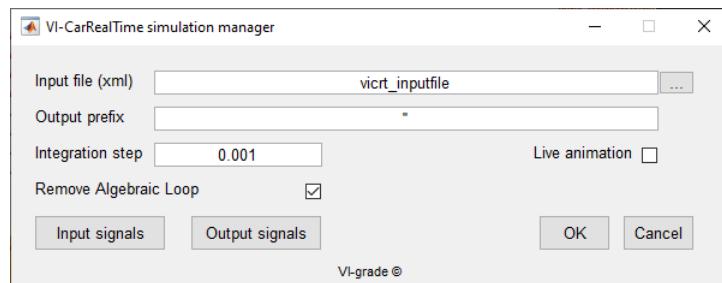
Running this simulation will create the send file in your working directory, which includes the options needed to launch **VI-Animator** from a remote application (MATLAB, in this particular case).

```
VI.PTW (VI-CarRealTime Python Task Window)
>>> run ()
Running Simulation in files only mode ....
File VI_CRT_Demo_VI_Track_active_vdd_send_svm.xml was successfully created.
```

In MATLAB, enter the following into your Command Window:

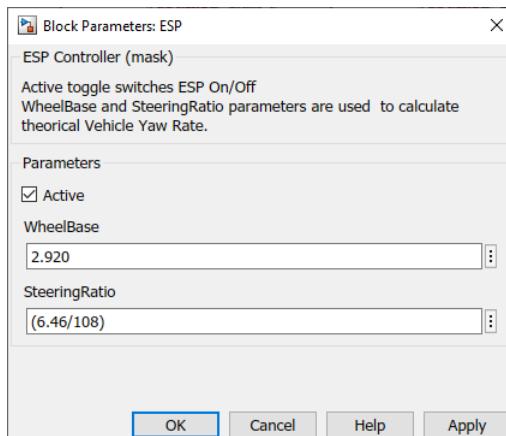
```
Command Window
fx >> vicrt_inputfile = 'VI_CRT_Demo_VI_Track_active_vdd_send_svm.xml';
```

The above command is used in the VI-CarRealTime Simulation Manager to retrieve the `send_svm` file that will be needed for our co-simulation analysis. Now, the variable shown in the **Input File (xml)** field in the picture below points to the `VI_CRT_Demo_VI_Track_active_vdd_send_svm.xml` file in your working directory.

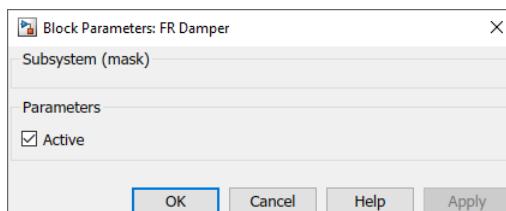


In the Active Vehicle Simulink model, activate the following components by checking the **Active** checkbox within each respective subsystem mask (in other words, double-click the icon corresponding to each component listed below to summon the component's respective Block Parameters menu):

- ESP:

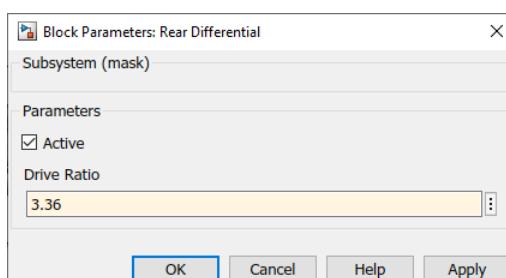


- Dampers (FL, FR, RL, RR):



Note: Since we are using external (Simulink) dampers, we must tell VI-CarRealTime's solver not to use the internal damper models. In order to do this, within Simulink, we should add the damper activity inputs signals into the **VI-CarRealTime** vehicle mask, and set them all to a constant value of zero. In this case, activity inputs are already present in **active_vehicle.mdl** model and their values are related to the **Active** checkbox inserted in the mask of the damper subsystems.

- Rear Differential (found by double-clicking on the Driveline icon):

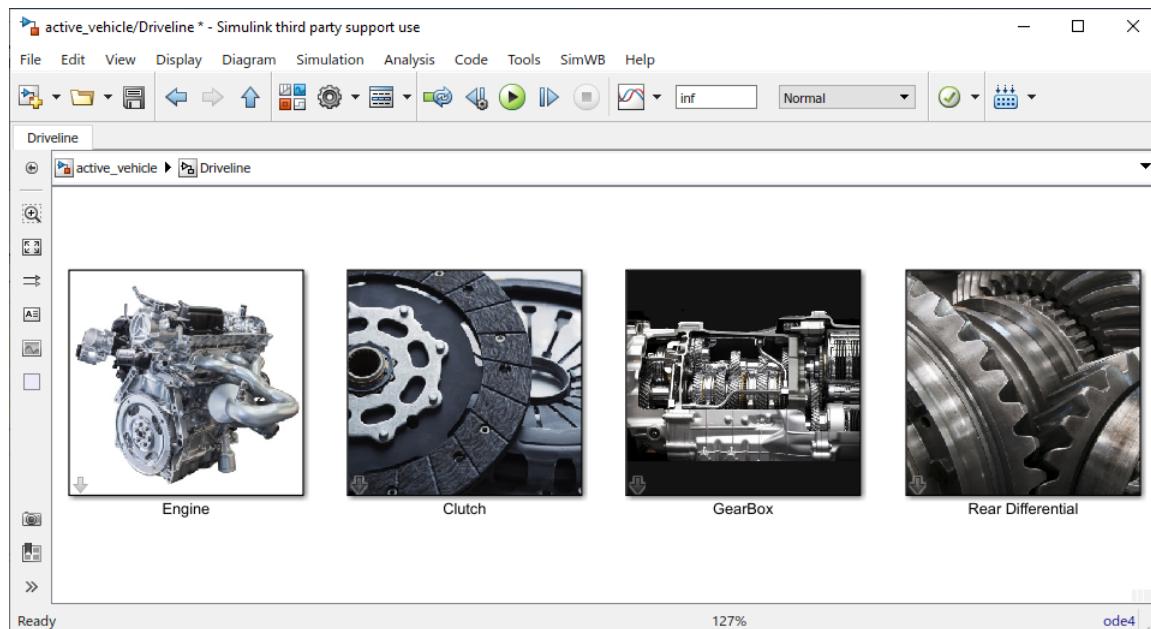


Note: Since we are going to compare results of passive and active vehicle and Static Lap Time is computed using VI-CarRealTime vehicle data, we have to set, in Simulink differential, the same **Drive Ratio** of VI-CarRealTime model.

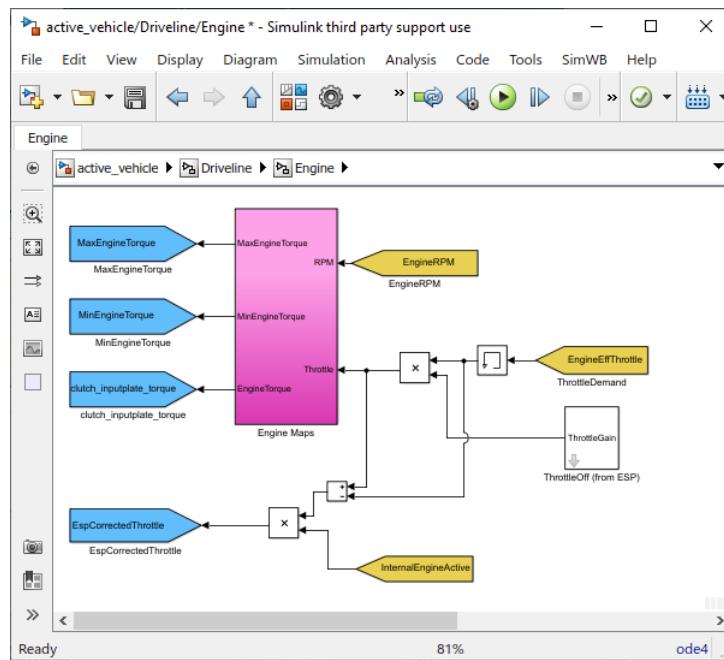
- Live Animation:



Even though within this tutorial, only the Rear Differential Driveline component is used, the other Driveline components can be manipulated/edited by double-clicking their respective Driveline subsystem mask (icon).



As a practical example of other editable Driveline components, the implementation of the Engine block is shown below; notice there is a connection block between the ESP and engine components. In the case of strong understeering conditions, the throttle input to the engine can be released (turned off), in order to avoid losing control of the vehicle.

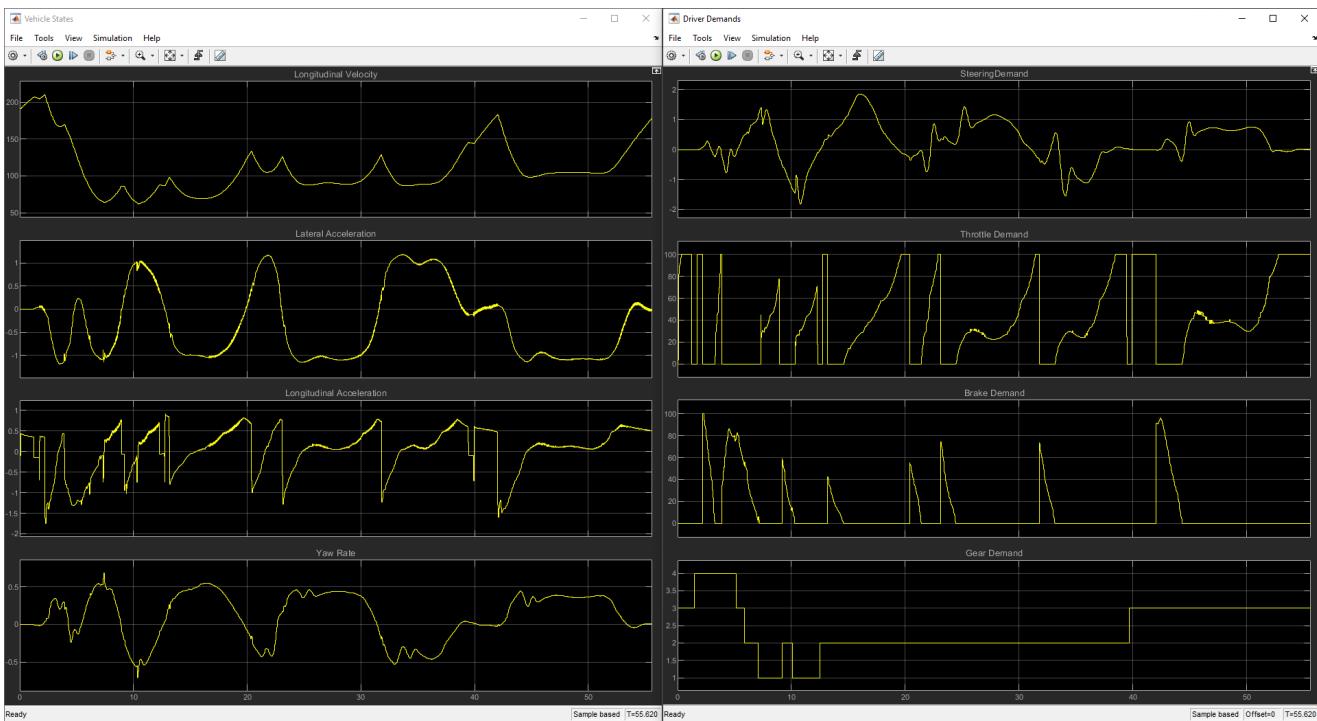


Note: some of the subsystems contain algebraic loops that are solved through Memory blocks that provide IC to the model. In order to obtain good results, IC have to be set to reasonable values for the analysis.

Now that everything is properly set up (according to the many steps above), start the simulation by

pressing the button, and VI-Animator will run, showing the resulting animation, while Simulink's analysis is concurrently in progress.

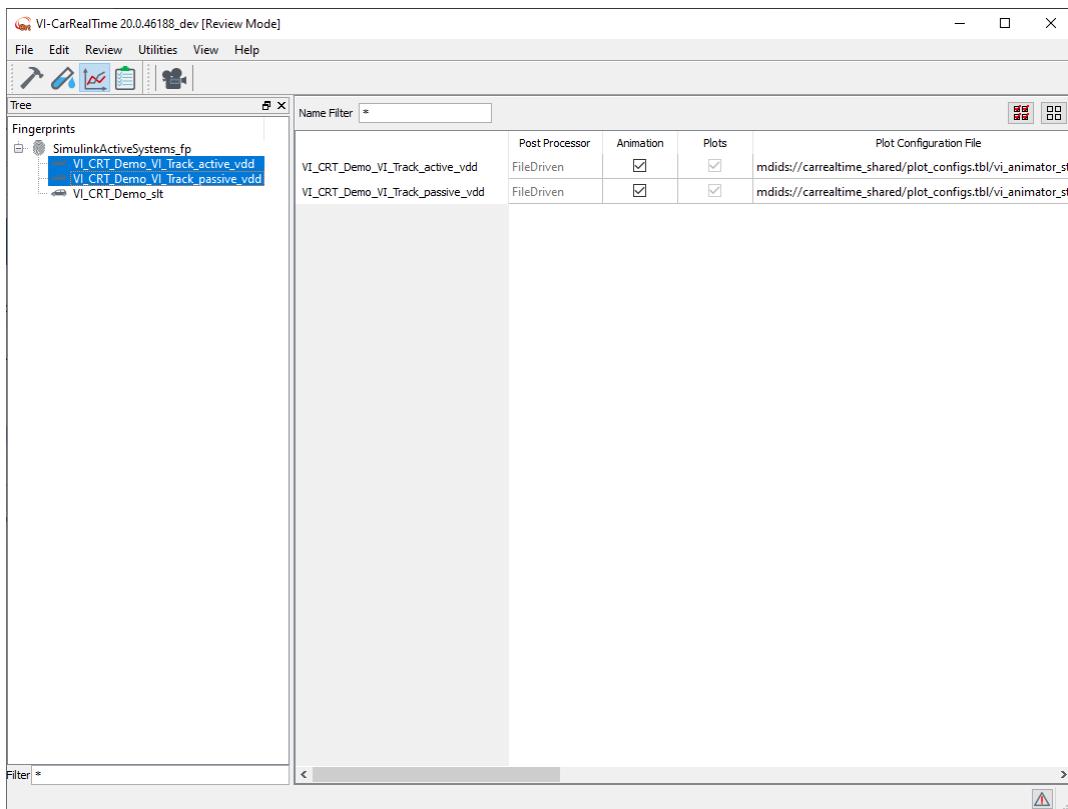
Double-click on the **Monitors** mask (icon) from the Simulink window to access the **Vehicle States** and **Driver Demands** scopes, which should correspondingly match the ones pictured in the following screenshots.



Now we can switch back to **VI-CarRealTime** and compare results of the active and passive cases, the former having been generated using MATLAB/Simulink.



In **VI-CarRealTime's Review Mode**, select the fingerprint and postprocess results using the button, which will summon VI-Animator.



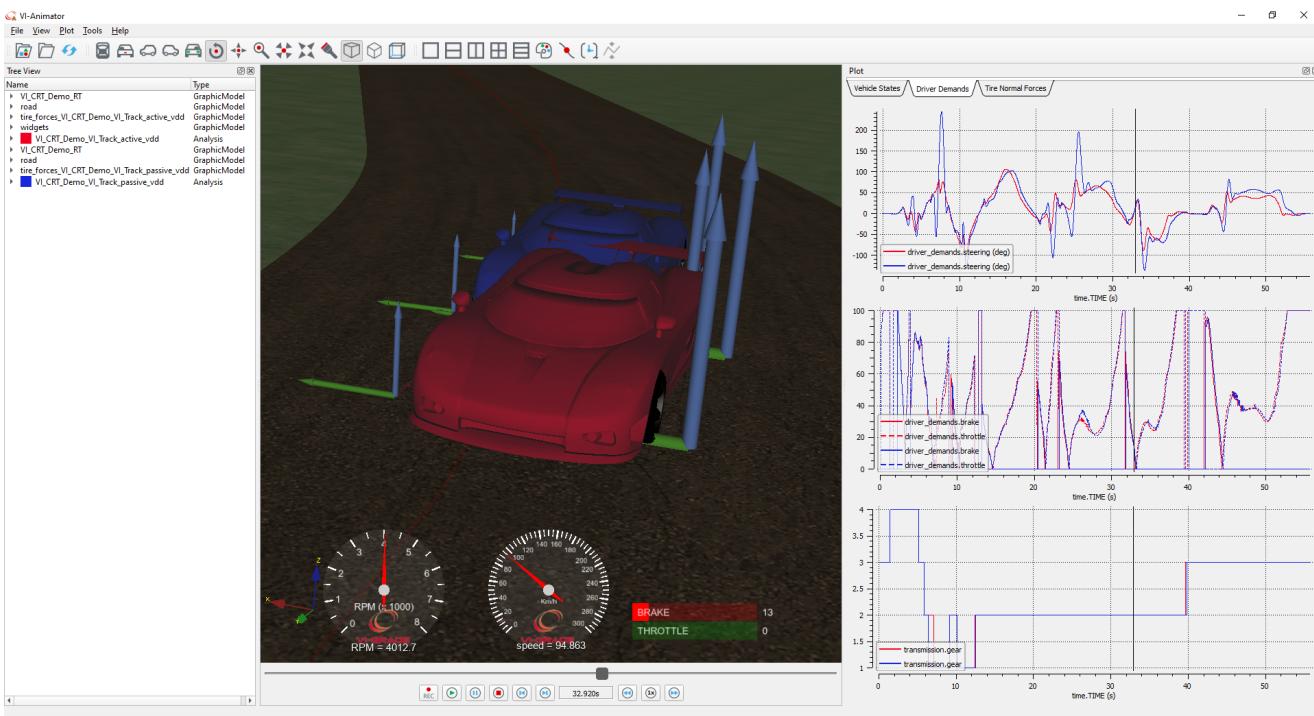
Note: Save this VI-CarRealTime session at this point, since it will be the starting point for the next tutorial ([Skyhook Control System](#)).

The VI-Animator window that just finished loading results shows a comparison between the passive (blue) and the active (red) models.

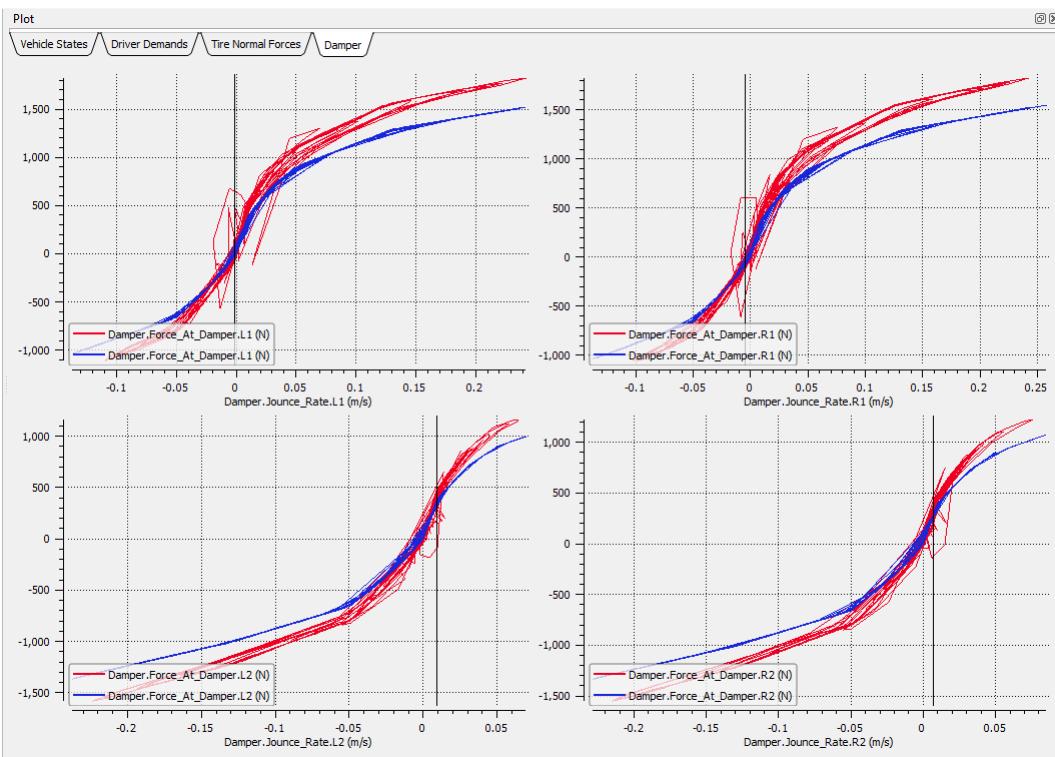
In certain sections of the track, the active vehicle is slower than the passive one; this phenomenon occurs because active devices prevent the driver from getting too close to the stability limit (whereas the passive vehicle does not).

The screenshot below shows a direct comparison between the two steering signals; they are very similar, but the small differences that do exist are due to the Simulink active system used in our active simulation (LSD, EPS, etc.).

VI-CarRealTime



The next screenshot shows the comparison of the damping curves for the two vehicle simulations:

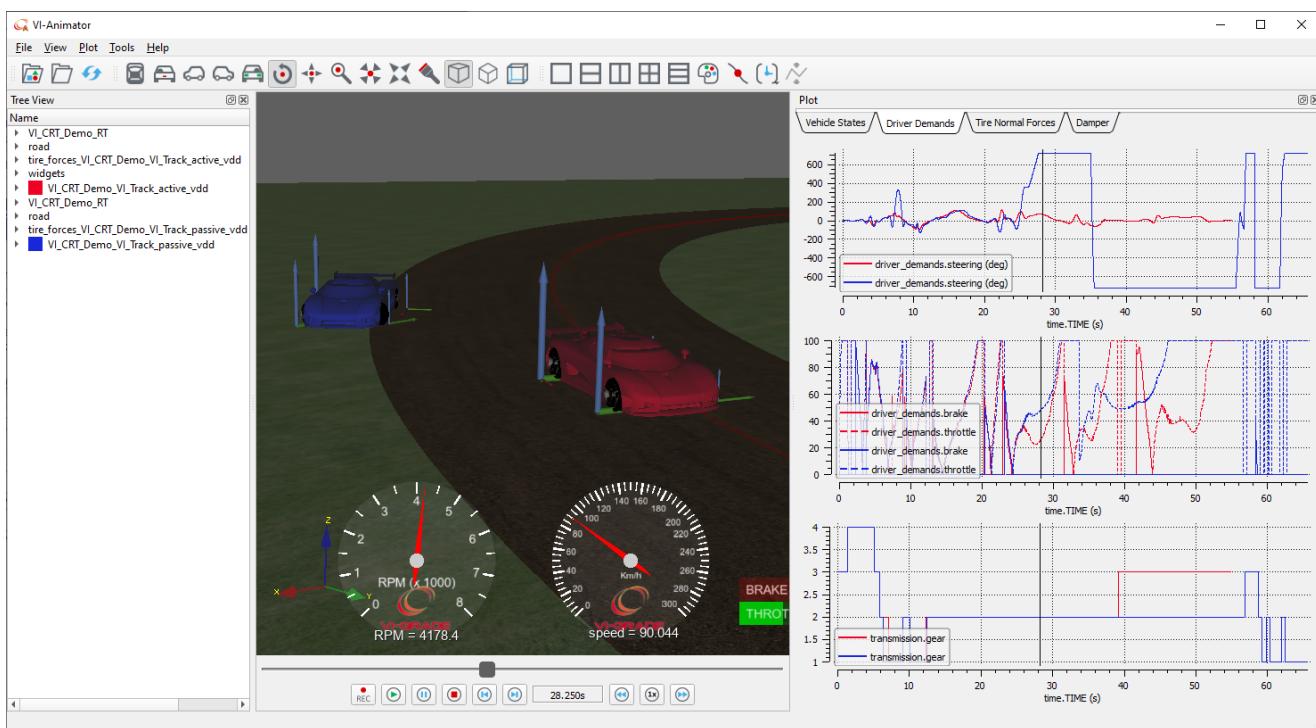


It can also be observed that while the passive vehicle damper forces essentially act on a single spline curve, the effect of the active dampers display a more "cloudy" characteristic, specially for the front dampers.

It is now interesting to repeat the process described in this tutorial section, but this time using a **Lateral Performance Factor of 1.00** for the **StaticLapTime** event.

Doing so would illustrate (in VI-Animator) that while the passive vehicle model (blue) loses control on the 5th turn, the active model (red) stays inside the track.

This means that the stability limit has been exceeded by passive vehicle, and no action could possibly be taken by the driver to avoid losing control of the car.



Skyhook Control System

The purpose of this tutorial is to replace VI-CarRealTime's non-linear dampers with a new type of damper called Skyhook dampers, which are built into Simulink.

Note: You can only perform this tutorial once you have already completed the previous [Assessing Vehicle Active Systems](#) tutorial section.

As the name suggests, a skyhook damper is a damper connected to some inertial reference "in the sky".

The goal of this particular damper system is to minimize the vehicle chassis's vertical velocity and acceleration, thereby substantially improving passenger comfort within the vehicle.

Essentially, this system adds more damping to the sprung mass and subtracts damping from the unsprung mass.

In this tutorial, the following logic is used to implement skyhook control:

- the skyhook control system knows the vertical velocity of the chassis (v_1), and the relative vertical velocity of the dampers (v_{12}) for each wheel;
- the system computes the dot product between the two previous velocities, and based on the result, computes the value of F_{sky} based on one of the follow equations:

$$\text{IF } v_1 \cdot v_{12} > 0 \text{ THEN } F_{sky} = c_{sky} * v_1$$

$$\text{IF } v_1 \cdot v_{12} < 0 \text{ THEN } F_{sky} = 0$$

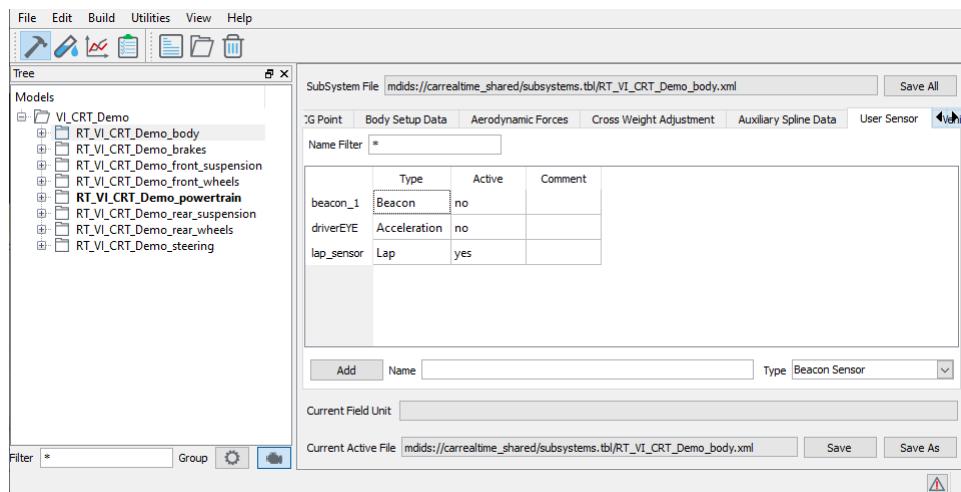
VI-CarRealTime

In your VI-CarRealTime installation directory, under the sub-folder `acarrt/examples/Simulink`, you find a model called `active_vehicle_skyhook`, which already has the skyhook system implemented.

In this tutorial section, you will learn how to build this skyhook system step-by-step, so that if you can later understand how to edit and manage all the skyhook damper subsystems, and also delete any relevant blocks inside these subsystems. It is recommended to make a copy of the skyhook damper subsystems file, in order to non-destructively perform any necessary edits to it.

Open the VI-CarRealTime session saved from the previous ([Assessing Vehicle Active Systems](#)) tutorial section.

In **Build Mode**, select the `Demo_body` subsystem under the `VI_CRT_Demo` model tree and navigate to the `User Sensor` tab.



Using the **Add** button, add four **Velocity Sensors**, named `vel_FL`, `vel_FR`, `vel_RL`, and `vel_RR`, respectively. These sensors represent a measure of the chassis' vertical velocities. Fill out their respective parameters as follows:

- **vel_FL:**

- I marker on `front_chassis` body, having coordinates (0,880,600);
- J marker on the `ground`, coordinates (0,880,600);
- Ref marker on the `ground`, having coordinates (0,0,0).

- **vel_FR:**

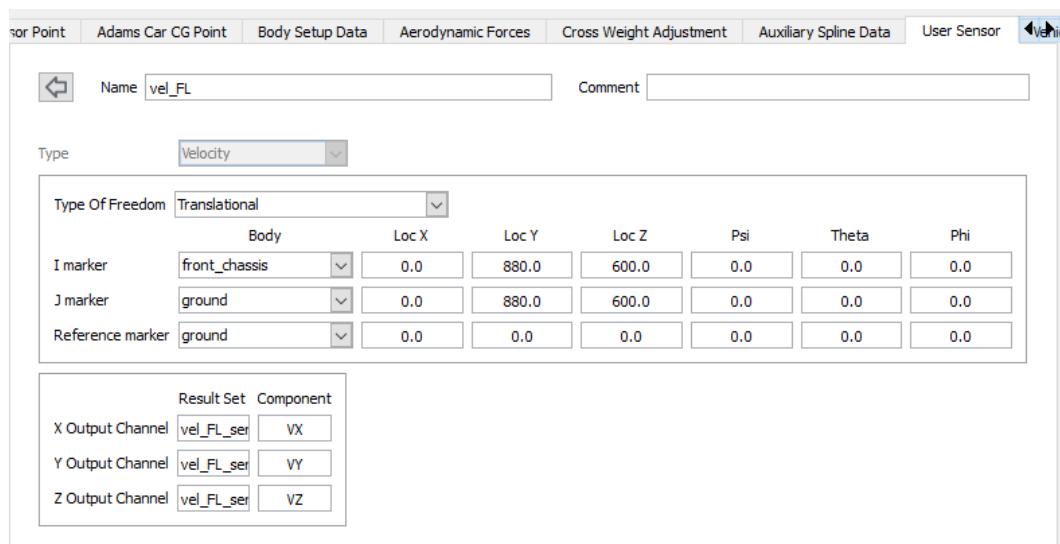
- I marker on `front_chassis` body, having coordinates (0,-880,600);
- J marker on the `ground`, coordinates (0,-880,600);
- Ref marker on the `ground`, having coordinates (0,0,0).

- **vel_RL:**

- I marker on `rear_chassis` body, having coordinates (-2920,870,600);
- J marker on the `ground`, coordinates (-2920,870,600);
- Ref marker on the `ground`, having coordinates (0,0,0).

- **vel_RR:**

- I marker on `rear_chassis` body, having coordinates (-2920,-870,600);
- J marker on the `ground`, coordinates (-2920,-870,600);
- Ref marker on the `ground`, having coordinates (0,0,0).



Switch to **Test mode** using the button. Right-click on the `VI_CRT_Demo_VI_Track_active_vdd` event and select **Copy Selected Event**.

Rename the new event `VI_CRT_Demo_VI_Track_active_skyhook` and run this event (in the **files only** Mode of Simulation) by clicking on the

```
VI.PTW (VI-CarRealTime Python Task Window)
>> run ()
Running Simulation in files only mode ....
File VI_CRT_Demo_VI_Track_active_skyhook_send_svm.xml was successfully created.
```

Now, open MATLAB and change its working directory so that it matches the VI-CarRealTime working directory.

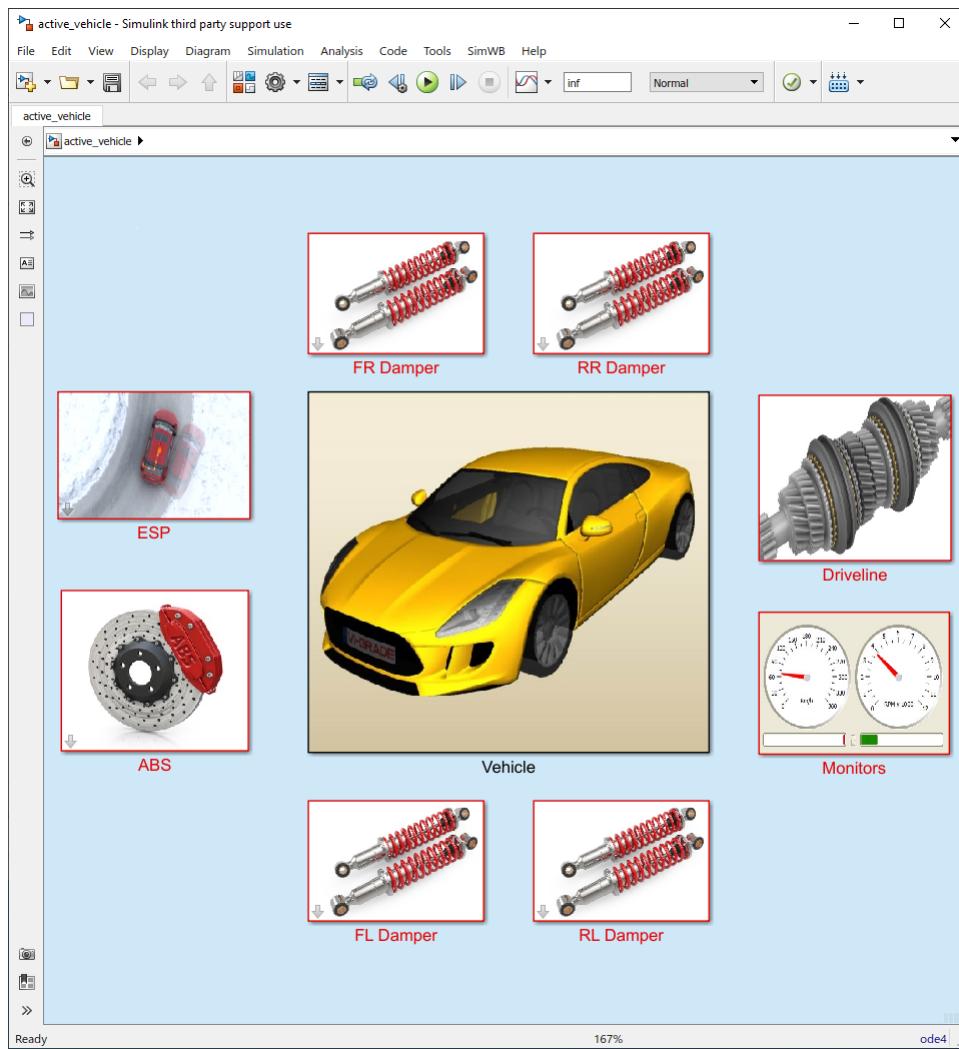
Run the "**addpath_vicrt_20**" (no quotes) command from the MATLAB Command Window to add the VI-CarRealTime libraries to MATLAB search path.

Type the following command in the Command Window to set the `send_svm.xml` file for the co-simulation.

```
Command Window
fx >> vicrt_inputfile = 'VI_CRT_Demo_VI_Track_active_skyhook_send_svm.xml';
```

Open the Simulink model `$VI-CarRealTimeInstallationdir\acarrt\examples\Simulink\active_vehicle.mdl` in MATLAB.

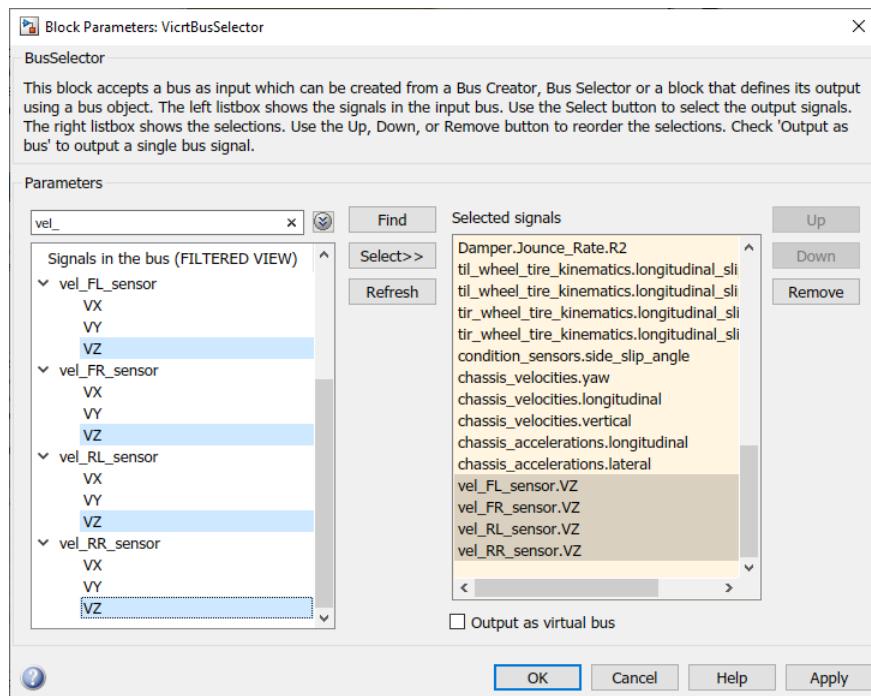
Note: the complete Skyhook Simulink model is provided in the same folder (`active_vehicle_skyhook.mdl`).



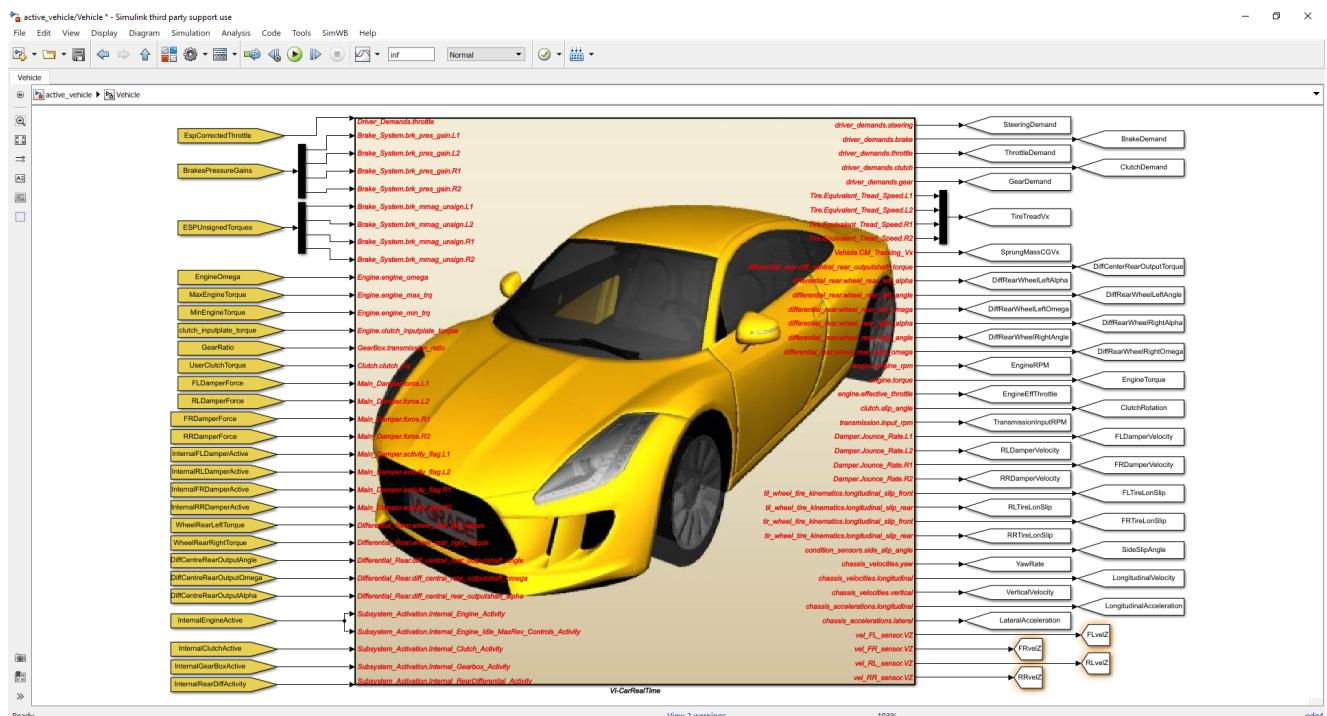
The next step is to edit the subsystems in the **active_vehicle.mdl** file as needed, to then be able to implement the skyhook control.

Using common Simulink blocks stored in the **Library Browser**, make changes to the **active_vehicle.mdl** subsystems as described in the following steps:

- Open the **Vehicle** subsystem from the main window, and then double-click the vehicle mask (icon).
- Click on the **Output signals** button in the VicrtBusSelector window and add the Z components of the four velocity sensors created earlier, as shown in the following screenshot. **Note:** If you don't see the required velocity sensors signals in the list of output signals (the list is called *Signals in the bus*), click on the **Refresh** button to reload all input and output signals.

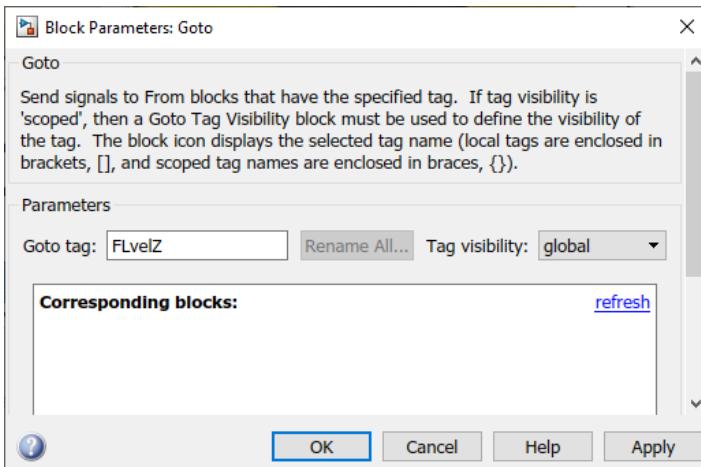


- Click on the **OK** button, both in the **VicrtBusSelector** window and in the **VI-CarRealTime Simulation Manager** menu, to apply changes.
- Now, once you see the output signals list including the four VZ values for your velocity sensors, add four **GOTO** blocks as shown below. The purpose of these blocks is to link the sensor outputs with the related **FROM** type blocks (these will be created in the next steps).

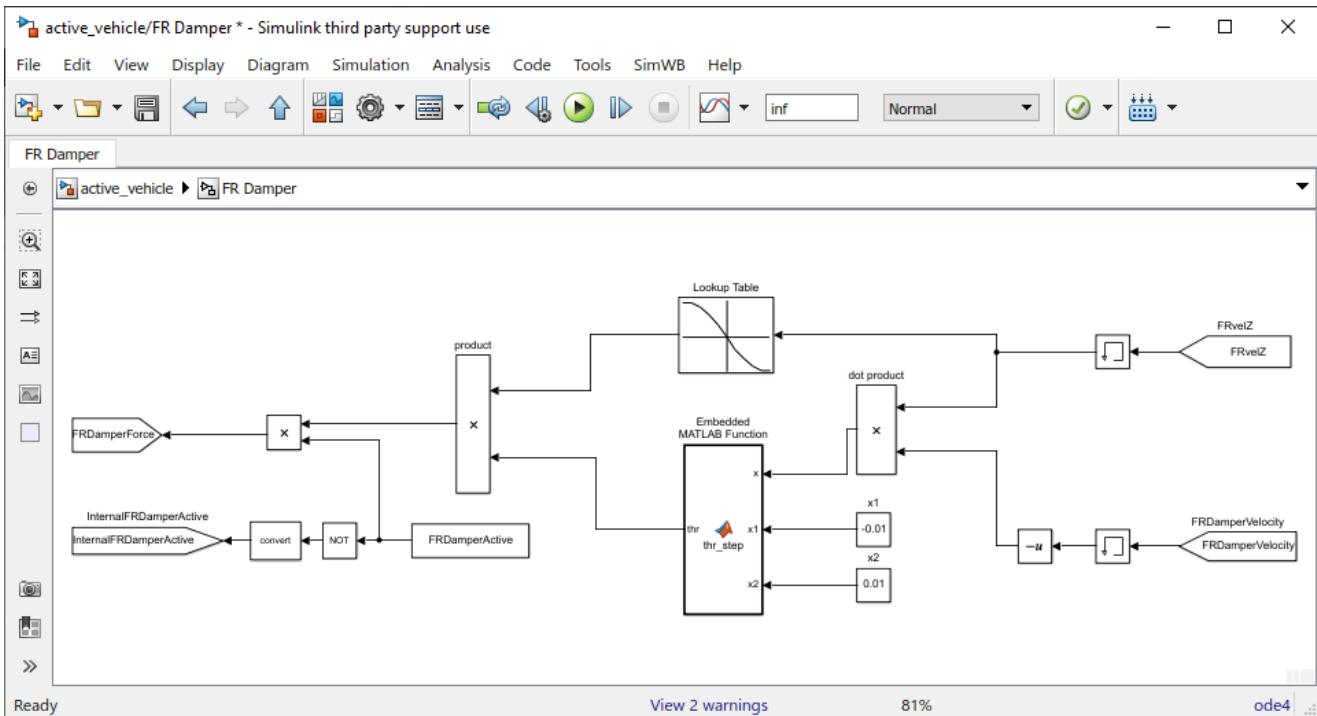


Note: Don't forget to set the tag visibility for each **GOTO** block to *global* (as shown in the menu in the following screenshot, which can be accessed by double clicking on the GOTO block for each velocity sensor), in order to make each block visible to all of the Simulink model's subsystems.

VI-CarRealTime

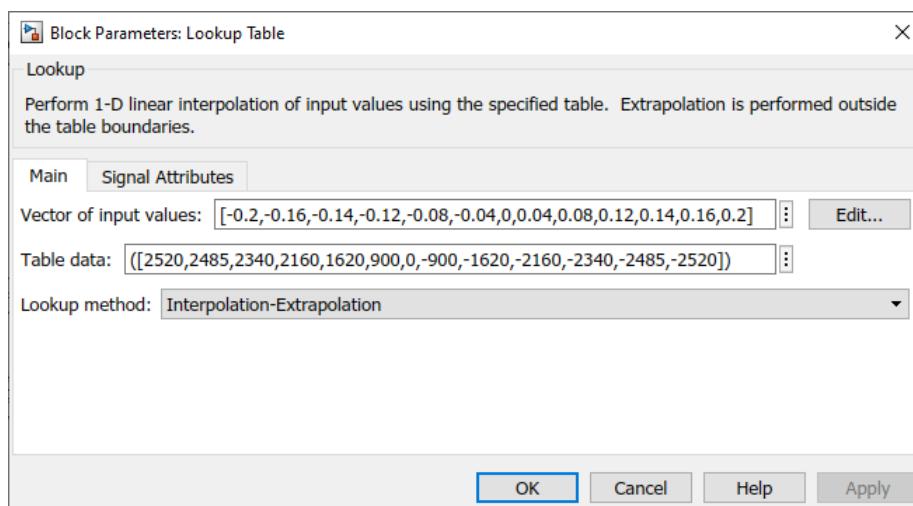


Switch to the **FR Damper** subsystem (whose mask can be found within the main Simulink model page) and change the blocks as displayed in the figure below (all of the blocks shown below can be found in the Simulink toolbar, within the **Library Browser**).



FRvelZ and *FRDamperVelocity* are **From** type blocks. They refer to the stored **Goto** blocks that were previously created. *FRDamperForce*, on the other hand, is a **Goto** block that refers to the **From** block with the same name (stored inside the **Vehicle** subsystem).

The *Lookup Tables* block stores the skyhook damper curve, whose values are as follows:



The *Embedded MATLAB Function* block stores the code (pictured below) which models a necessary step function. This function is used to reproduce the **IF** condition for the skyhook control; it is essentially a means of obtaining a continuous control system.

```

1  function thr = thr_step(x,x1,x2)
2  y1=0;
3  y2=1;
4  n=length(x);
5  thr=zeros(n,1);
6  if (x2<x1)
7      tmp=x2;
8      x2=x1;
9      x1=tmp;
10     tmp=y2;
11     y2=y1;
12     y1=tmp;
13 end
14 for i=1:n
15     if x(i)<=x1
16         thr(i)=y1;
17     elseif x(i)>x2
18         thr(i)=y2;
19     else
20         a=y2-y1;
21         delta=(x(i)-x1)/(x2-x1);
22         thr(i)=y1+a*delta^2*(3.0-2.0*delta);
23     end
24 end

```

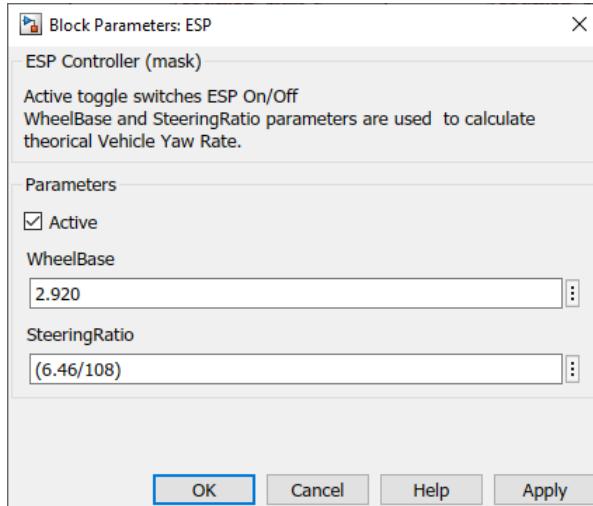
Repeat the same steps explained above for the other three damper subsystems (FL, RR, and RL Dampers).

Note: To do this quickly, you may simply copy and paste the FR Damper subsystem blocks into the three other respective subsystems, and then match the corresponding name values (variables) in each respective Goto and From block.

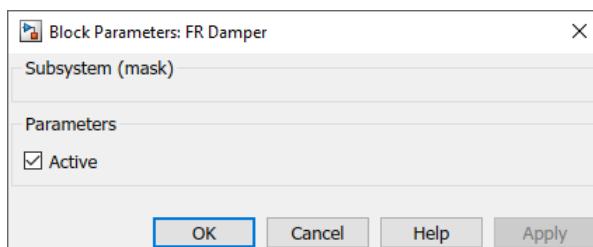
Since we are using external (Simulink) skyhook dampers, we must tell VI-CarRealTime's solver not to use the internal dampers model. In order to do this, within Simulink, we should add the damper activity inputs in the vehicle mask, as shown in [Replacing Standard Components](#) tutorial. In this case, activity inputs are already present in active_vehicle.mdl model and their values are related to the **Active** checkbox inserted in the mask of the damper subsystems.

In the Active Vehicle Simulink model, activate the following components by checking the **Active** checkbox within each respective subsystem mask (in other words, double-click the icon corresponding to each component listed below to summon the component's respective Block Parameters menu):

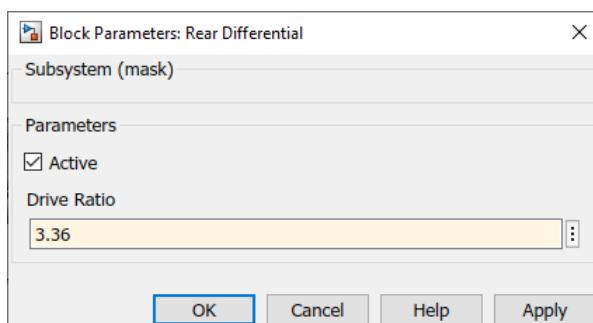
- ESP



- Skyhook Dampers (FL, FR, RL, RR)



- Rear Differential (found by double-clicking on the Driveline icon)

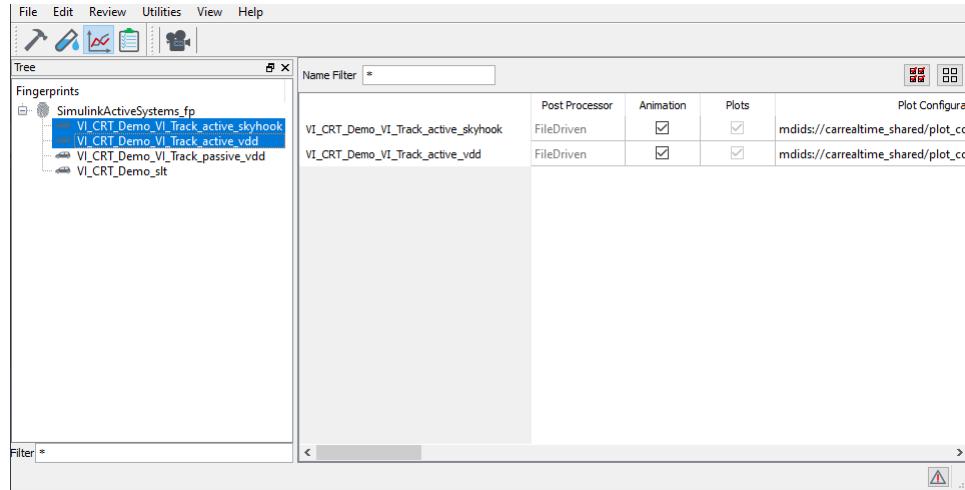


Note: Since we are going to compare results of passive and active vehicle and Static Lap Time is computed using VI-CarRealTime vehicle data, we have to set, in Simulink differential, the same **Drive Ratio** of VI-CarRealTime model.

Start the simulation by clicking on the button and wait until it is complete. Please be aware that this particular simulation may take slightly longer than any of the ones completed in earlier tutorial sections.

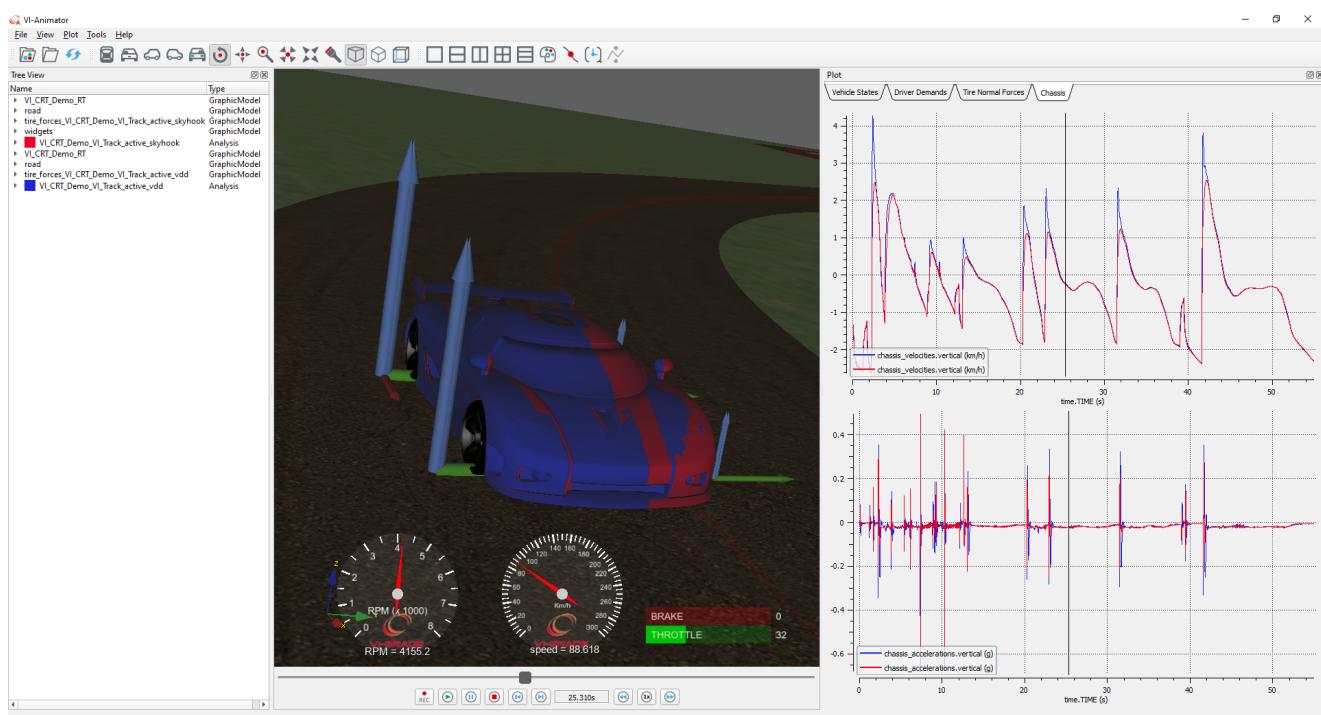
Now, we must switch back to VI-CarRealTime and compare the results related to the active vehicle versus the skyhook vehicle.

In **VI-CarRealTime's Review Mode** select the two events as shown below, and post-process the results using the VI-Animator (click on the button, with the events shown below selected).



In the VI-Animator interface, add a new plot after *Tire Normal Forces* table as rename it as **Chassis**.

Modify the page layout to 2x1 (). Plot *chassis_velocities.vertical* vs. *time* in the first plot, and *chassis_accelerations.vertical* vs. *time* in the second plot.



The damper skyhook configuration (red curve) reduces the the chassis' vertical velocity and acceleration, improving overall passengers comfort, as can be confirmed when looking at the plots created in VI-Animator.

Note: When taking into account the sprung mass vertical velocity and neglecting the unsprung mass velocity, the skyhook system configuration can however lead to worse performance, when compared with the same vehicle using standard dampers.

Using MaxPerformance in Simulink

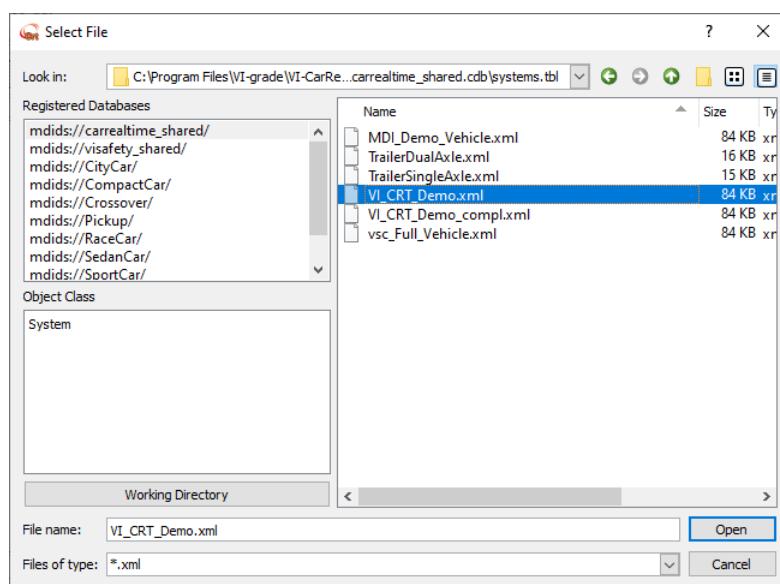
The tutorial explains how to set up and run a MaxPerformance event in MATLAB/Simulink, in four major steps:

1. [Create Event in VI-CarRealTime](#)
2. [Instrument Simulink Model](#)
3. [Create and Customize Callbacks](#)
4. [Run MaxPerformance](#)

Create Event in VI-CarRealTime

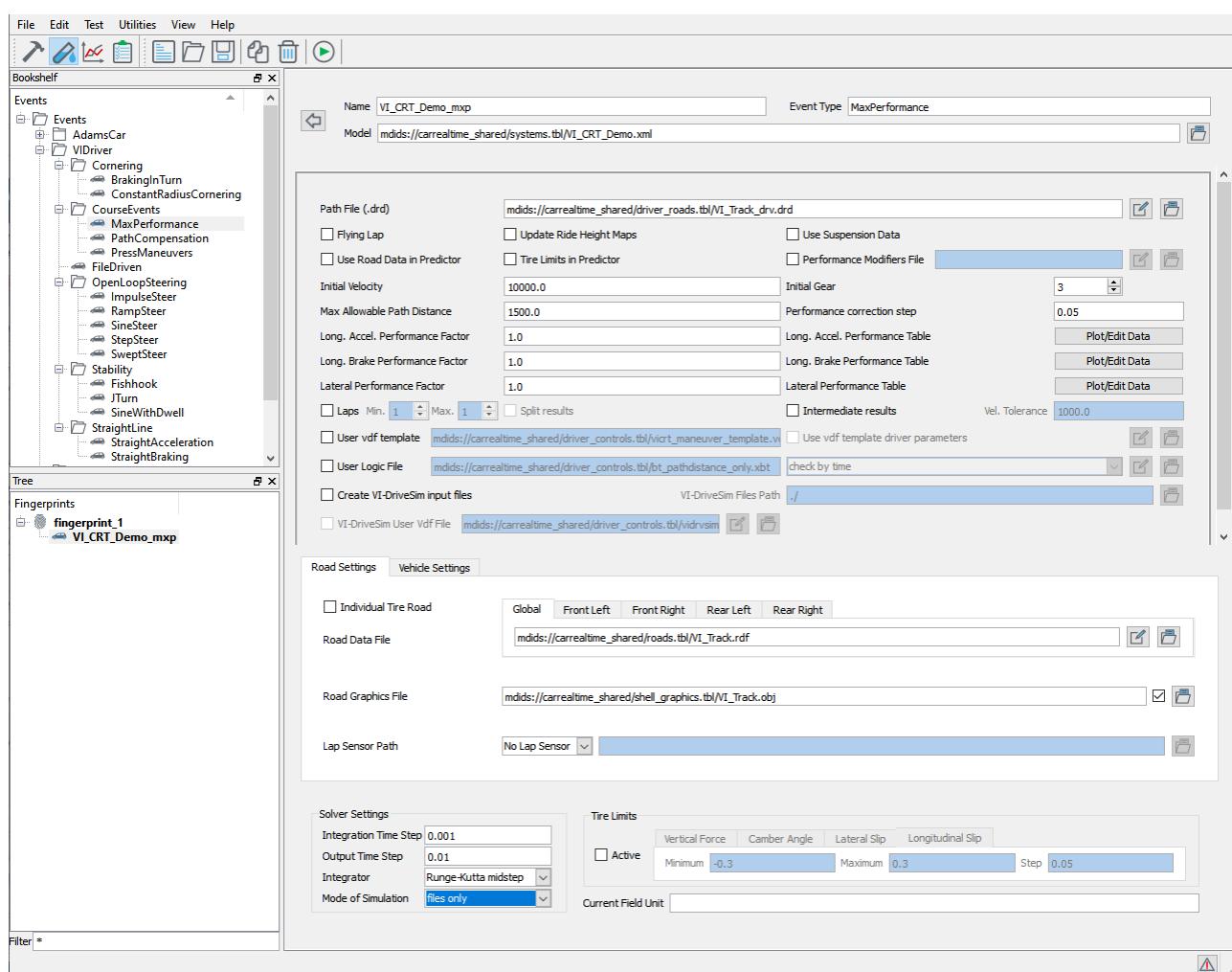
The first part consists in setting up and creating the event in VI-CarRealTime.

Starting in **Build mode**, open the model **VI_CRT_Demo.xml** from the shared database.



Now, switch to **Test Mode** using the button and create a **MaxPerformance** event (**CourseEvents** subsection in the Tree View) under a new fingerprint, and set the Mode of Simulation to "**Files only**".

After matching your event parameters with those found in the screenshot below, run the simulation to generate the `*send_svm.xml` file.



Instrument Simulink Model

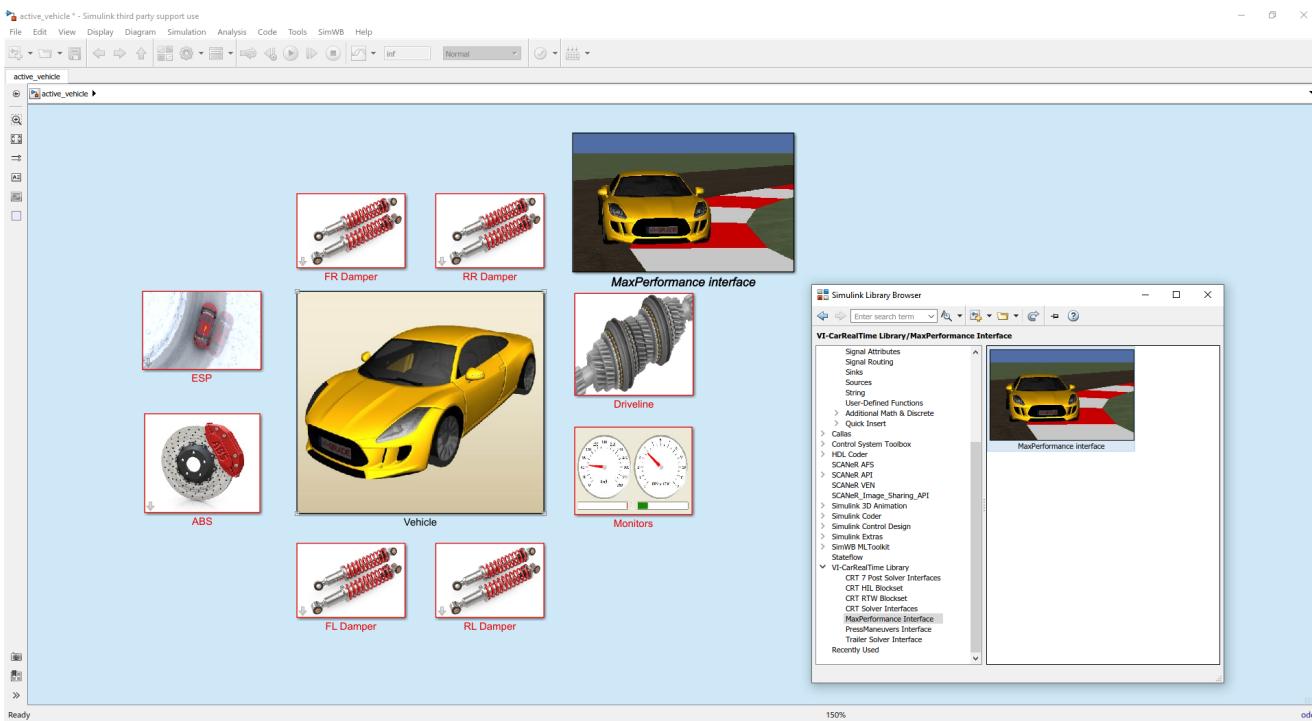
Start a MATLAB session in the same working directory as the one previously used for VI-CarRealTime.

Run the **addpath_vicrt_20** script from your MATLAB command window to add the VI-CarRealTime libraries to the MATLAB search path.

A generic VI-CarRealTime Simulink model can be used to run a MaxPerformance co-simulation; it is sufficient to add a particular instrumentation block anywhere in the Simulink model page, to allow the relevant optimization routines to interact with the underlying VI-CarRealTime solver.

Open **active_vehicle.mdl** and add the *MaxPerformance* interface to the Simulink model: to do it just drag the **MaxPerformance interface** block from the VICRT Library Browser and drop it anywhere into the active Simulink page (as shown below).

VI-CarRealTime

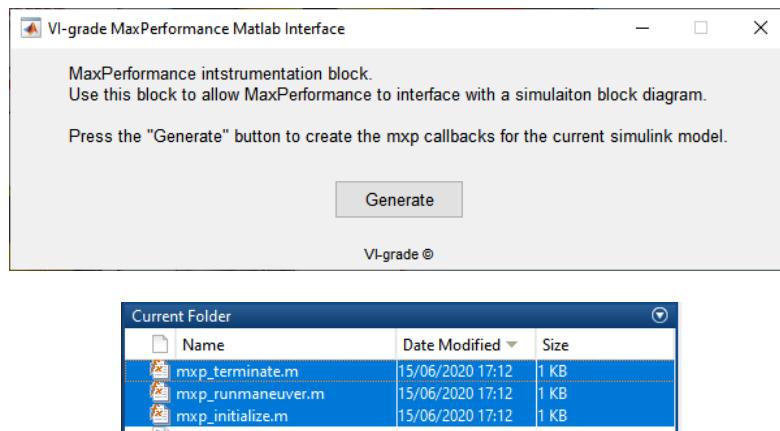


Rename the model **active_vehicle_mxp** and save it in your working directory.

Create and Customize Callbacks

Press the **Generate** button, which is located in a menu summoned by double-clicking the "MaxPerformance interface" mask (icon).

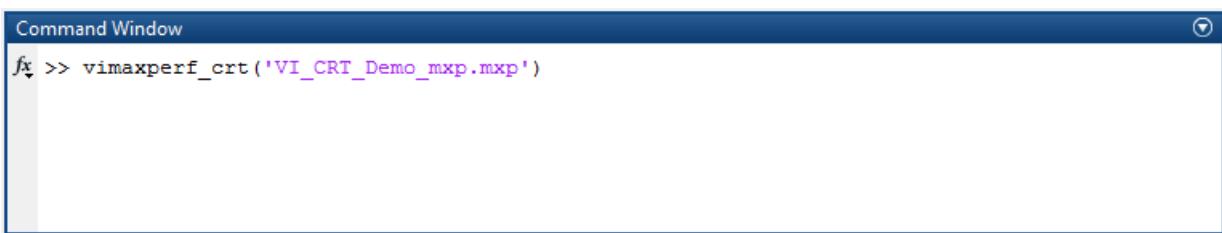
This action will generate the three interface type [Matlab functions](#) into your working directory and will customize them to allow the addition of your own MATLAB initialization/termination code, if need be.



Note: Simulink model must be **saved** before submitting the MaxPerformance simulation. In fact, *mxp_runmaneuver* callback calls the *load_system* function which loads a new instance of the Simulink model.

Run MaxPerformance

To submit the MaxPerformance event, from the MATLAB console, run the **vimaxperf_crt** function with the "VI_CRT_Demo_mxp.mxp" file as input, exactly as shown below.



You can also provide additional arguments passing the output prefix for the simulation results file and/or the input history file name if you want to restart the MaxPerformance event from a previous state configuration:

```
vimaxperf_crt ({simulation_filename , simulation_outputprefix }, history_filename )
```

- *simulation_filename*
name of the input .mfp file
- *simulation_outputprefix*
name of the output prefix for the results file (optional)
- *history_filename*
name of the input .xsh file (optional)

Using PressManeuvers in Simulink

The following tutorial shows how to set up and run a *PressManeuver* event in MATLAB/Simulink.

The tutorial is composed by the following sections:

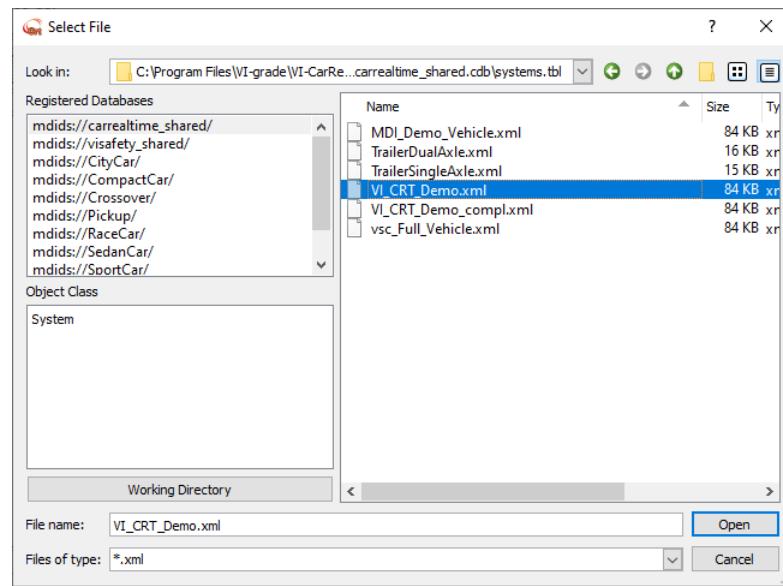
1. [Create Event in VI-CarRealTime](#)
2. [Instrument Simulink Model](#)
3. [Create and Customize Callbacks](#)
4. [Run PressManeuvers](#)

Create Event in VI-CarRealTime

The first part consists in setting up and creating the event in VI-CarRealTime.

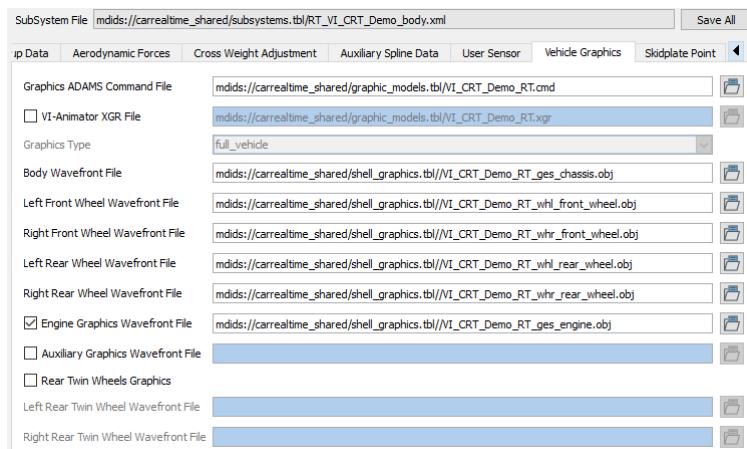
Starting in **Build mode**, open the model **VI_CRT_Demo.xml** from the shared database.

VI-CarRealTime

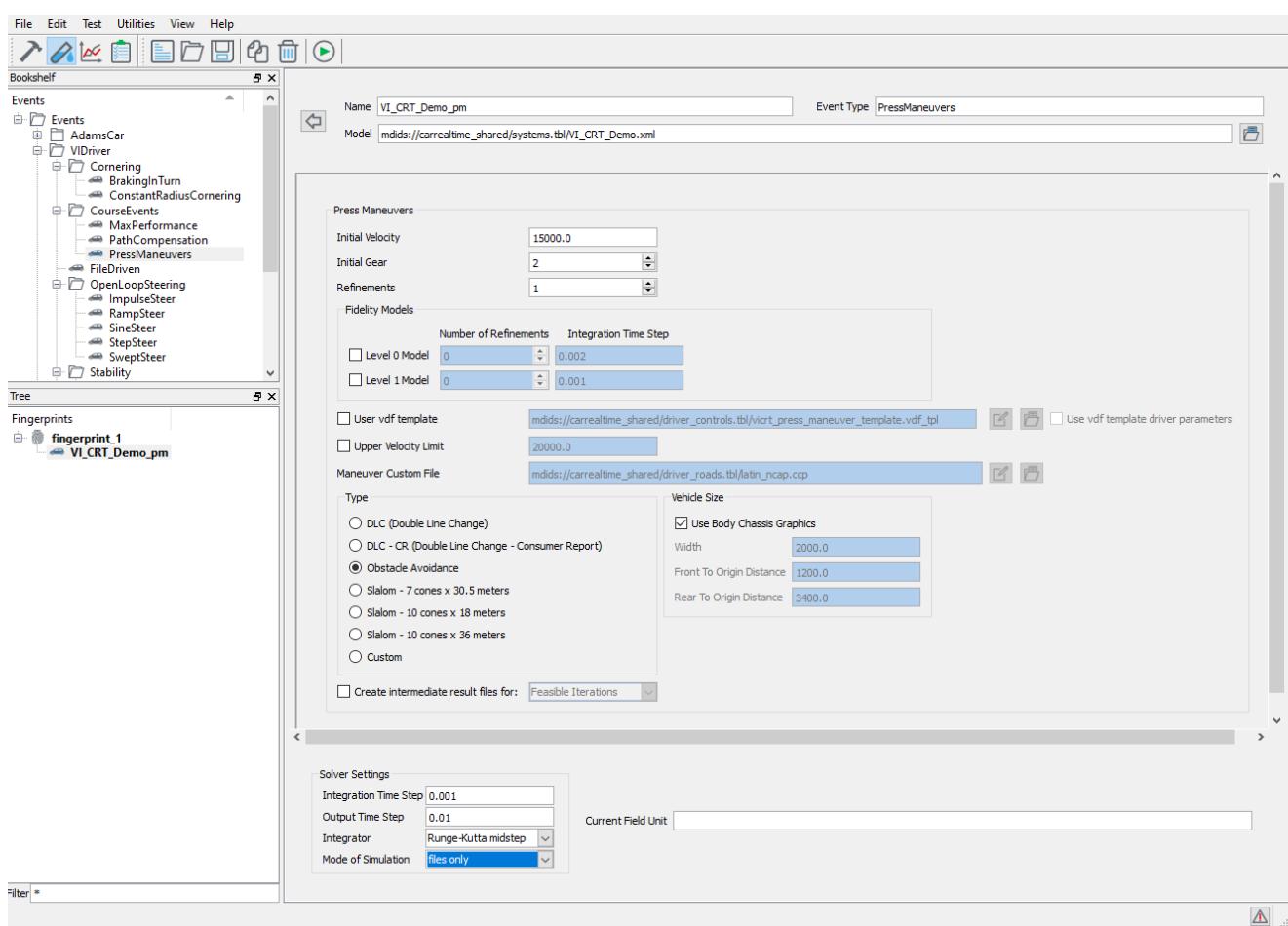


Since XGR file is not supported for automatically retrieving the vehicle size based on the geoemtry, we need to switch to OBJ files:

- Browse to **Body Subsystem**
- select **Vehicle Graphics** panel
- remove the check from **VI-Animator XGR File**.



Now, switch to **Test Mode** using the button and create a **PressManeuver** event (**CourseEvents** subsection in the Tree View) under a new fingerprint, and set the Mode of Simulation to "Files only".



The aim is to generate the `*send_svm.xml` file to be used in Simulink.

```
VI.PTW (VI-CarRealTime Python Task Window)
>>> run ()
##### PRESS MANEUVER INFO #####
Vehicle width      : 2.176 meters
Front to origin distance: 1.003384448 meters
Rear to origin distance : 3.9436155520000002 meters
Running Simulation in files only mode ....
File VI_CRT_Demo_pm_send_svm.xml was successfully created.
```

Instrument Simulink Model

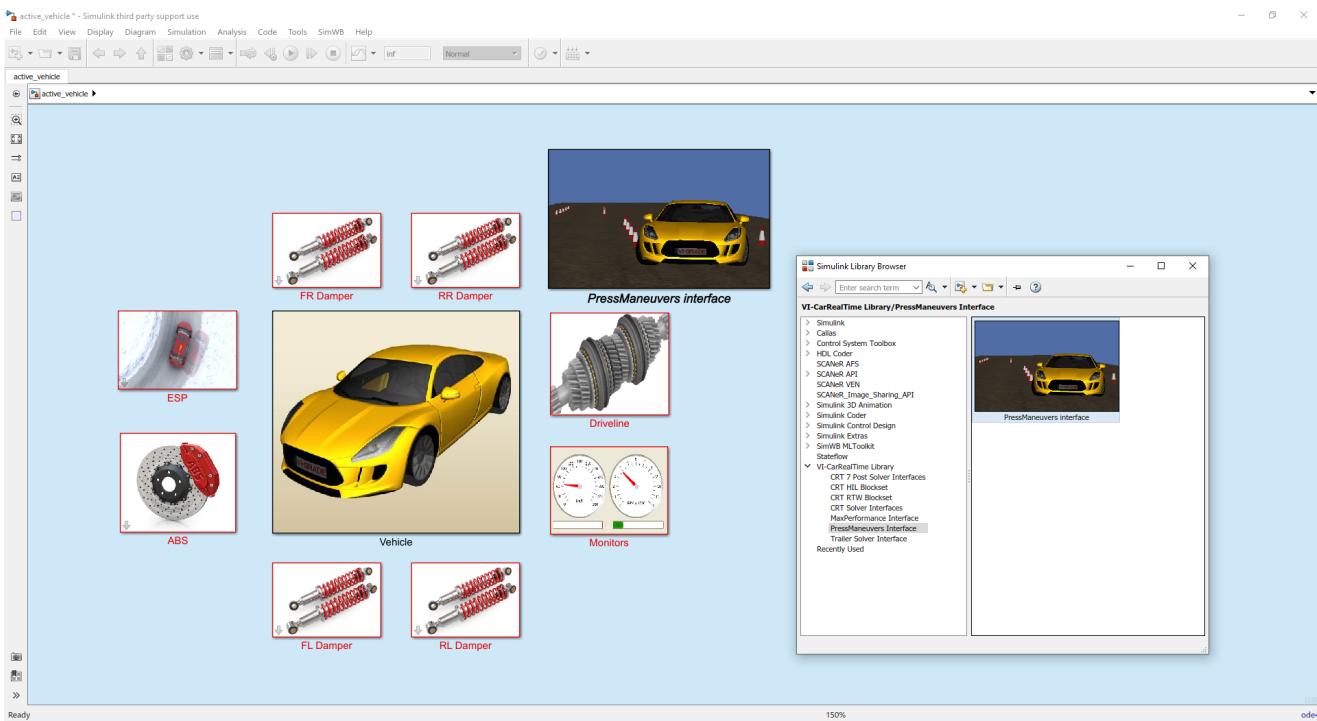
Start a MATLAB session in the same working directory as the one previously used for VI-CarRealTime.

Run the `addpath_vicrt_20` script from your MATLAB command window to add the VI-CarRealTime libraries to the MATLAB search path.

A generic VI-CarRealTime Simulink model can be used to run a PressManeuver co-simulation; it is sufficient to add the corresponding instrumentation block anywhere into the model, allowing the smart DOE routines to interact with the underlying VI-CarRealTime solver.

Open `active_vehicle.mdl` and add the `PressManeuvers` interface to the Simulink model: to do it just drag the **PressManeuvers interface** block from the VICRT Library Browser and drop it anywhere into the active Simulink page (as shown below).

VI-CarRealTime

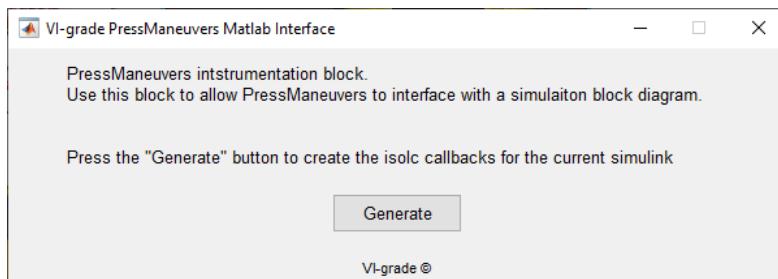


Rename the model **active_vehicle_pm** and save it in your working directory.

Create and Customize Callbacks

Press the **Generate** button, which is located in a menu summoned by double-clicking the "PressManeuvers interface" mask (icon).

This action will generate the three interface type [Matlab functions](#) into your working directory and will customize them to allow the addition of your own MATLAB initialization/termination code, if need be.

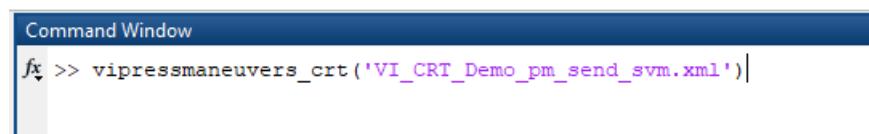


Current Folder		
	Name	Date Modified
isolc_terminate.m	16/06/2020 17:00	1 KB
isolc_runmaneuver.m	16/06/2020 17:00	1 KB
isolc_initialize.m	16/06/2020 17:00	1 KB
active_vehicle_pm.slx	16/06/2020 16:59	192 KB
VI_CRT_Demo_pm_send_svm.xml	16/06/2020 16:17	1.03 MB
VI_CRT_Demo_pm.bat	16/06/2020 16:17	1 KB

Note: Simulink model must be **saved** before submitting the PressManeuver simulation. In fact, *isolc_runmaneuver* callback calls the *load_system* function which loads a new instance of the Simulink model.

Run PressManeuvers

To submit the PressManeuvers event, from the MATLAB console, run the **vipressmaneuvers_crt** function with the "VI_CRT_Demo_pm_send_svm.xml" send file as input, exactly as shown below.



It is also possible to set the *output prefix* for the simulation results file as the second item of an input string array:

```
vipressmaneuvers_crt {{simulation_filename , simulation_outputprefix }}
```

- *simulation_filename*
name of the input *send_svm.xml* file.
- *simulation_outputprefix*
name of the output prefix for the results file (optional).

Using External Road

The aim of this tutorial is to instruct the user on how to use an external road in Simulink.

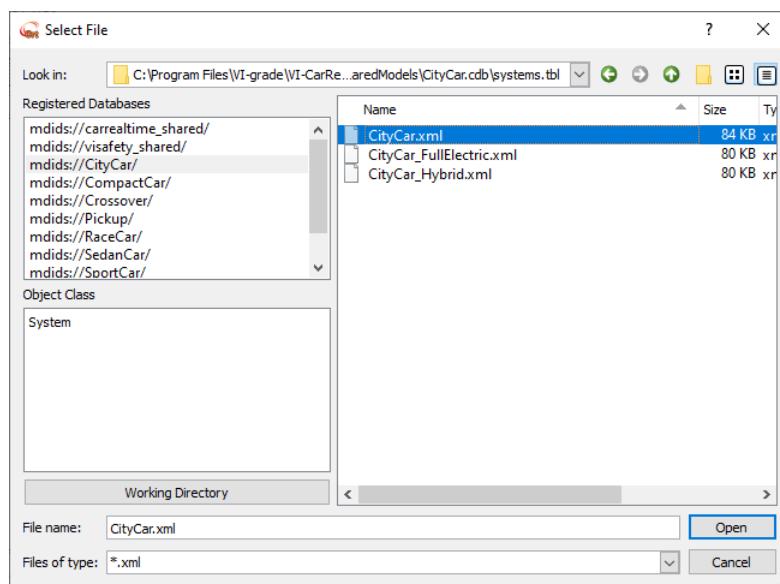
To characterize an external road, VI-CarRealTime solver must be provided with contact path information for all the tires: in particular X,Y,Z global contact patch coordinates, Nx,Ny,Nz cosine direction and friction coefficient must be supplied.

For the details about the inputs required please refer to [Road External](#) input topic.

Start a new VI-CarRealTime session using the **vicrt20** command from a DOS (cmd) shell, or by using the VI-CarRealTime shortcut from the Windows Start Menu.

Starting in **Build mode**, add example model databases by selecting **Build -> Register Example Databases**.

Open the model **CityCar.xml** from CityCar database.



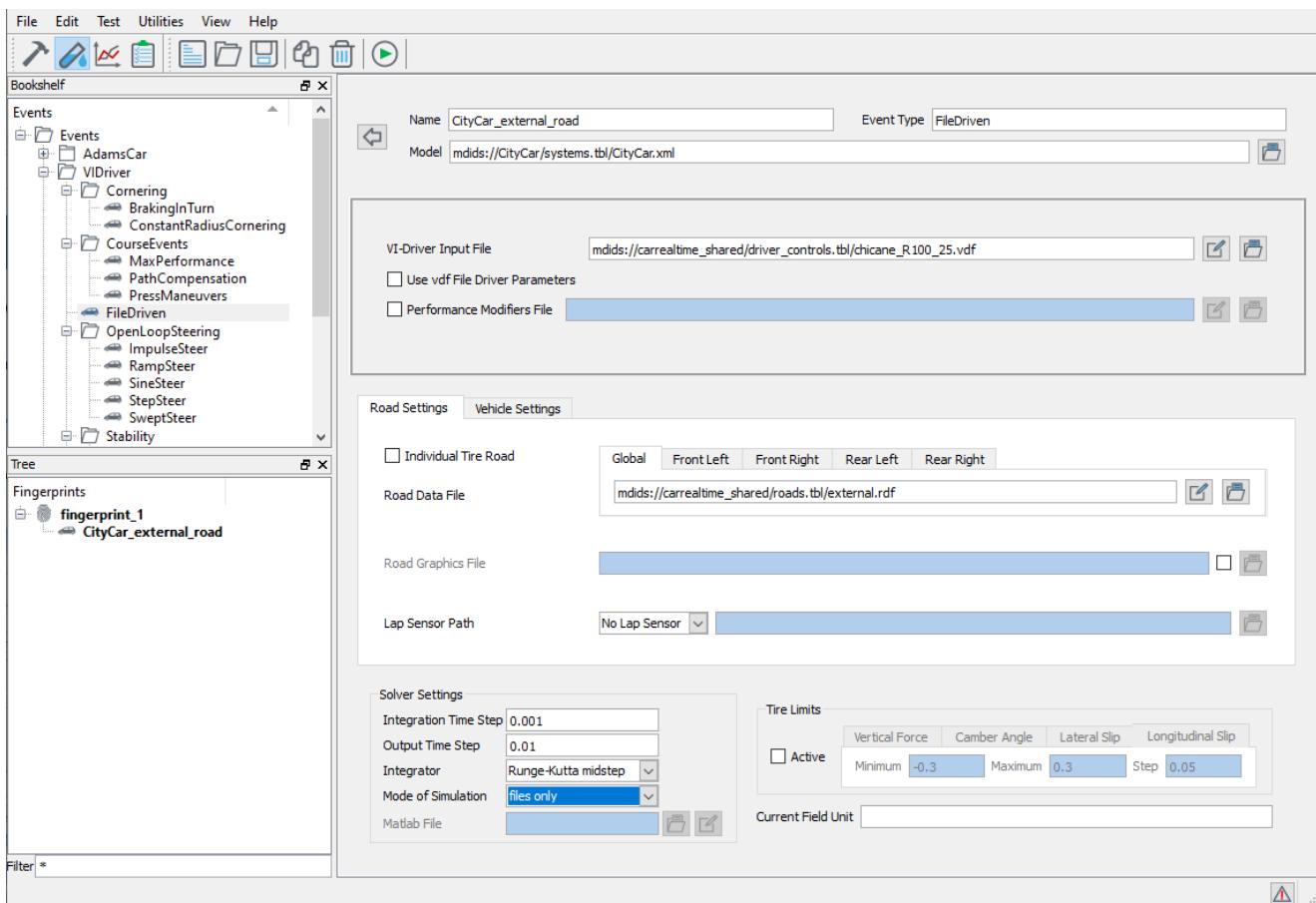
Switch to **Test Mode** using the  button.

Create a new **File Driven** event (VIDriver -> *FileDriven*) using **VI_CRT_Demo** model and set the following parameters:

- **chicane_R100_25.vdf** from the *carrealtime_shared* database as **VI-Driver Input File**;

VI-CarRealTime

- **external.rdf** from the *carrealtime_shared* database as **Road Data File**;
- select **files_only** as **Mode of Simulation**.
- rename the event **CityCar_external_road** as shown in the picture below.

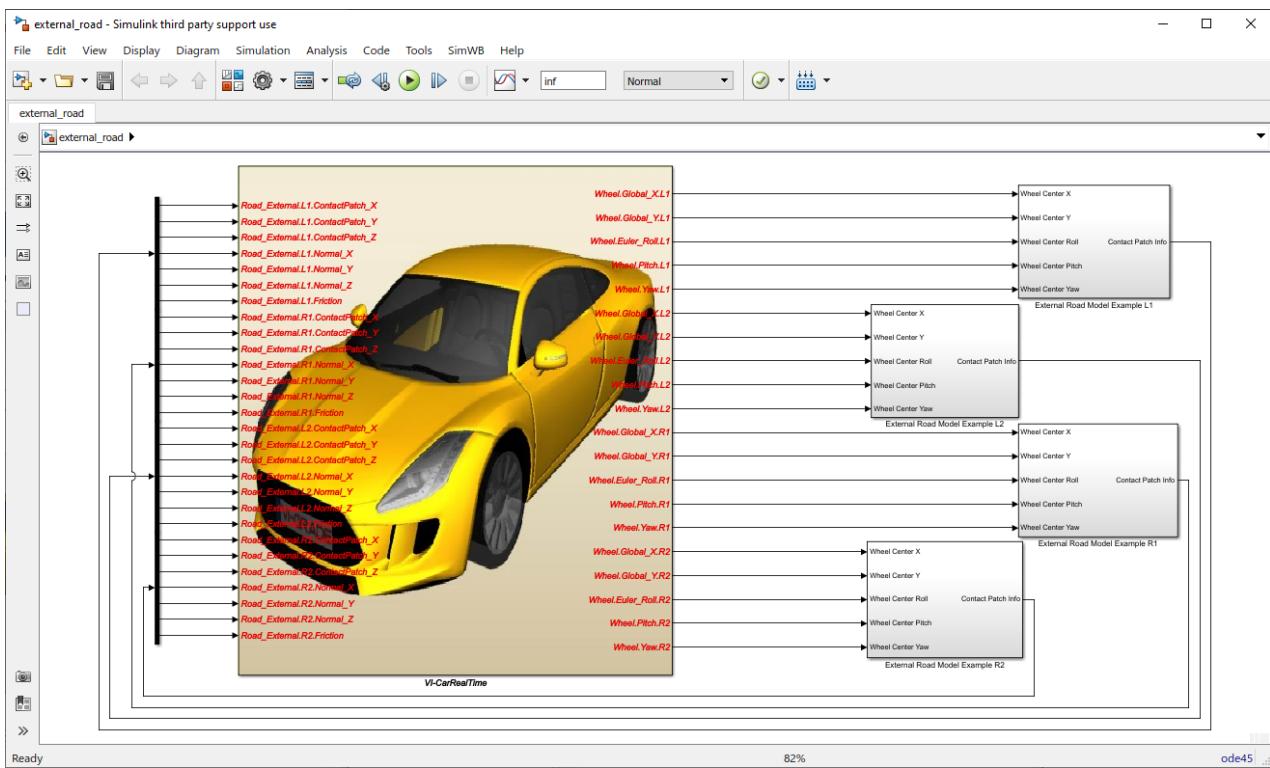


Run the event by pressing button in VI-CarRealTime so that *svm_send.xml* file will be created in the working directory; such file will be used for MATLAB and Simulink simulations.

Start MATLAB, and therein, set the same working directory as the VI-CarRealTime one.

In MATLAB Command Window, type and enter the command **addpath_vicrt_20** to add all the necessary paths for the co-simulation.

In MATLAB once again, open the following Simulink model: **\$VI-CarRealTime\$Installationdir\acarrt\examples\Simulink\external_road.mdl**



`external_road.mdl` defines a list of inputs and outputs which allows VI-CarRealTime model to communicate with the external road model.

In particular, the following inputs arrays are passed to VI-CarRealTime model for each tire in order to characterize the road:

- Contact Patch X location
- Contact Patch Y location
- Contact Patch Z location
- Road Normal X
- Road Normal Y
- Road Normal Z
- Road Friction

In this example model, for each tire, the road is defined as a flat road (Normal X and Normal Y are 0, Normal Z is 1) whose location is the projection of the wheel center location to ground (CP Z is always 0).

VI-CarRealTime



Enter the following command in your MATLAB Command Window:

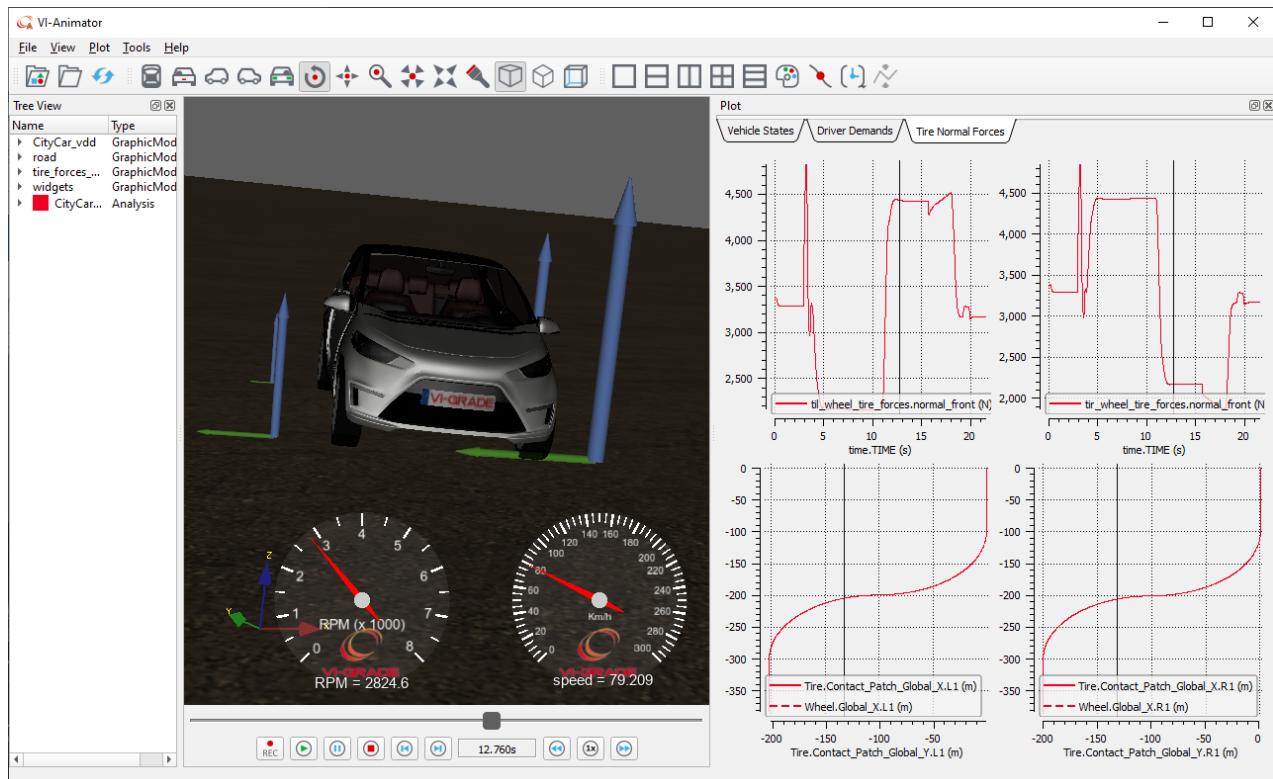
```
vicrt_inputfile = 'CityCar_external_road_send_svm.xml'
```

This command sets a variable which references the send_svm file that stores all the VI-CarRealTime model information; this variable is used in the VI-CarRealTime Simulink block.

Start the co-simulation with the button in the upper Simulink window toolbar.

When the simulation is completed, in the working directory the analysis result file is written.

Switch back to VI-CarRealTime, browse to [Review mode](#), select **CityCar_external_road** analysis and launch VI-Animator to review results.



Implementing an Electric Vehicle

The following tutorial introduces how to use the **VI-CarRealTime 20** MATLAB/Simulink interface in order to run a simulation of EV vehicle with external electric motor and battery.

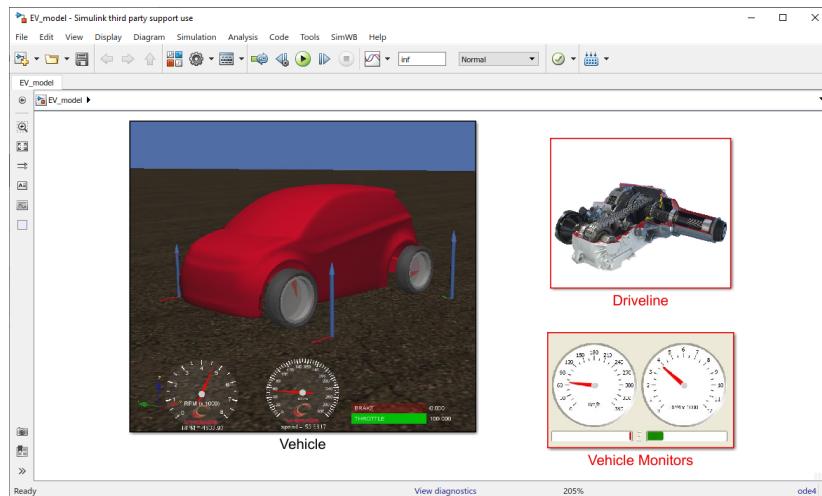
The tutorial is divided into the following sections:

- [Introduction of Simulink model](#)
- [Starting from kml path](#)
- [Creating send file](#)
- [Running analysis on Simulink](#)

Introduction of Simulink model

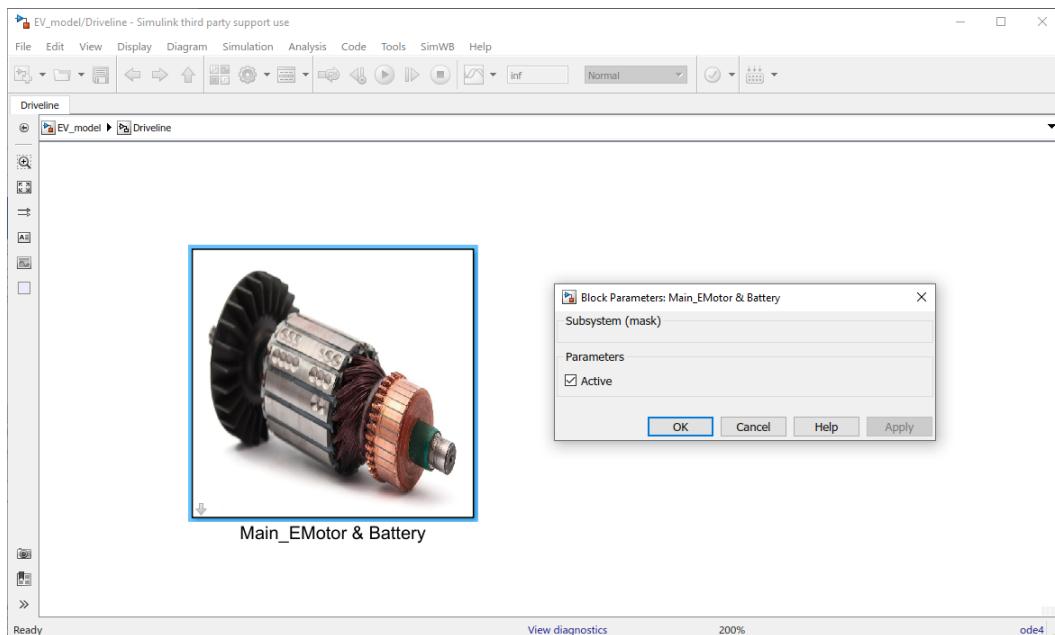
Presentation of Simulink Model

The simulink model **"EV_model.mdl"** is contained in **\$VI-CarRealTimeInstallationdir\acarrt\examples\Simulink**. The driveline block contains all the components of an electric drive system.



"Main_EMotor & Battery" block is activated. In this way the internal motor defined using **VI-CarRealTime** GUI is deactivated and this one will be used.

VI-CarRealTime

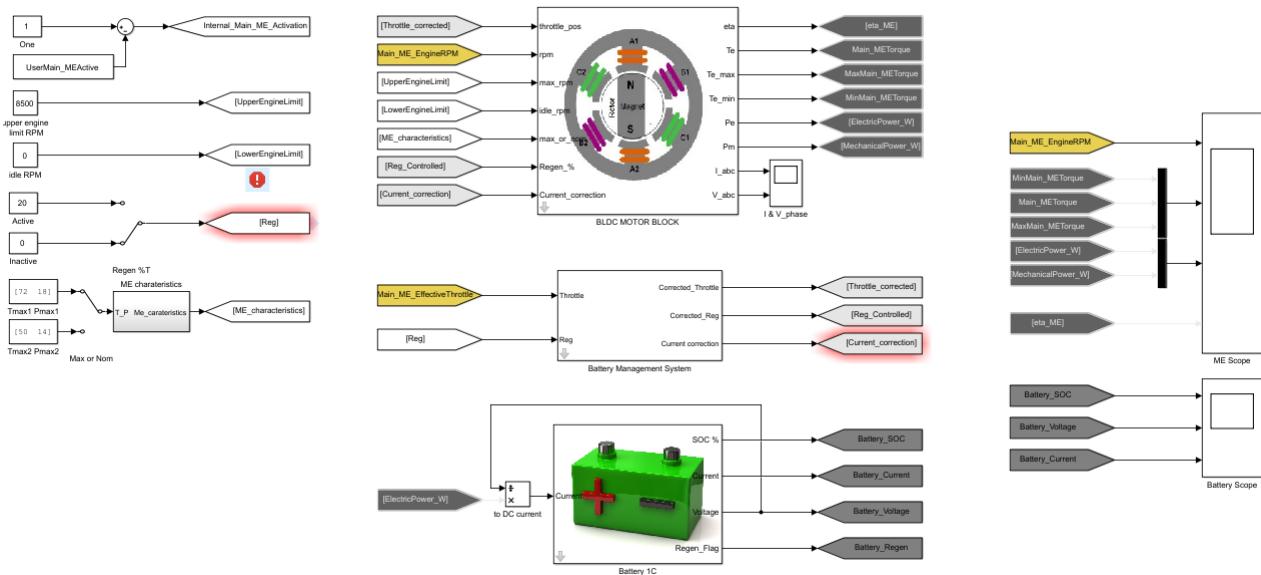


Looking under mask the "Main_EMotor & Battery" block we see at the left side the definition of some important variables for the models and at the right side the 3 systems, from top to bottom respectively:

Electric Motor

Battery Management System

Battery



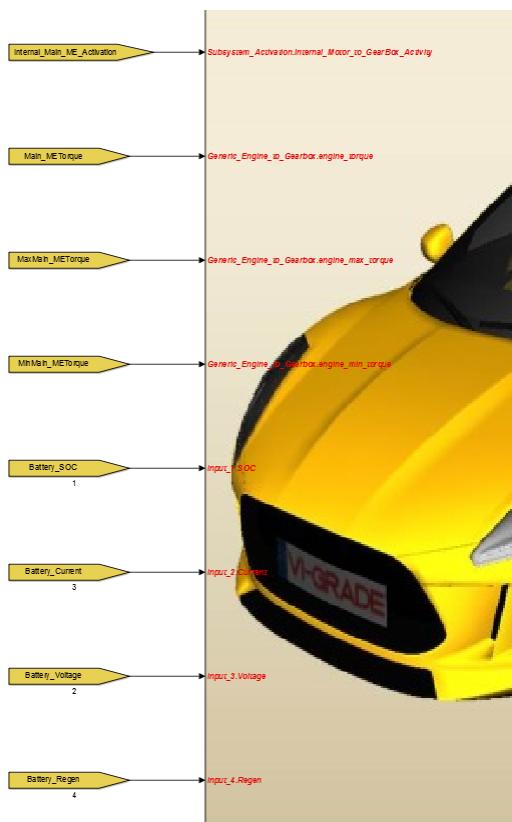
The electric motor model represents a BLDC motor. It is composed by an rpm limit block and a principal block where, starting from throttle and motor speed the block gives actual, maximum and minimal torque signals necessary for VI-CarRealTime Simulink Interface. In the same block it is also contained the regenerative logic activated during throttle release and the calculation of the electrical signals using Park transform. Using block's mask it is possible to configure the principal values of BLDC motor.

Remember that if the mask parameters change too much from standard configuration, it needs to change the file reference of "alpha_table" and the "Im_table" used inside the "BLDC MOTOR BLOCK" in order to have correct electrical signals.

The Battery block can simulate different type of batteries: the behavior of Li-ion, Lead acid, NiCd or NiMh battery can be represented. Using the mask of this block, the type of battery and the most important parameters can be set.

The "Battery Management System" is needed to limit the electric motor behavior: this block controls the SOC limit and the maximum value of current delivered or absorbed by the battery. The current limit can be set by double-clicking on the block.

In the vehicle block the signals that communicate with VI-CarRealTime Matlab/Simulink interface are those represented in the following figure.



"Subsystem_Activation.Internal_Motor_to_GearBox_Activity" is used to disable the internal (VI-CarRealTime) Main Motor in order to use an external (Simulink) one.

The three torque channels:

- "Generic_Engine_to_Gearboxengine_torque"
- "Generic_Engine_to_Gearboxengine_max_torque"
- "Generic_Engine_to_Gearboxengine_min_torque"

represent respectively the torque delivered by external motor, the maximum and the minimum torque at specific motor speed defined in the engine map contained in Simulink model. The last two values are used by VI-Driver to correctly compute the throttle signal value.

The last 4 channels are related to battery model:

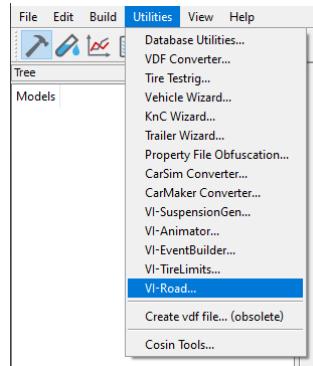
- "Battery_SOC"
- "Battery_Current"
- "Battery_Voltage"
- "Battery_Regen"

VI-CarRealTime

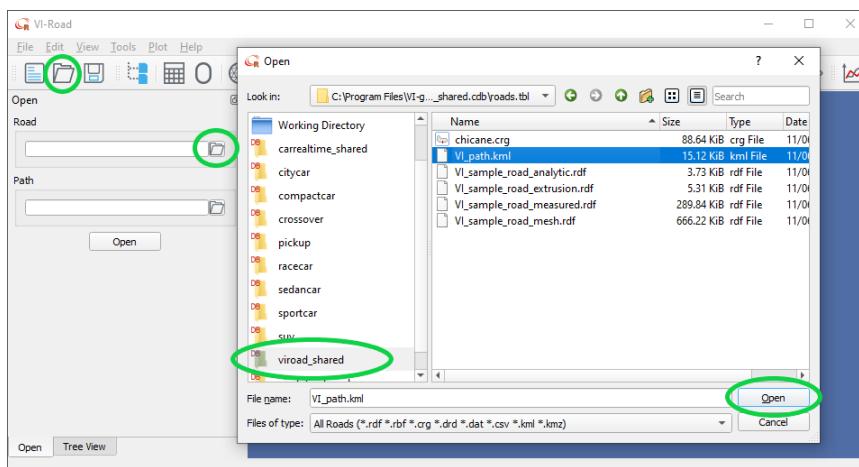
Having these input signals is important in order to write them in the analysis result file and facilitate their visualization in **VI-Animator**.

Starting from kml path**Create a road data file from kml in VI-Road**

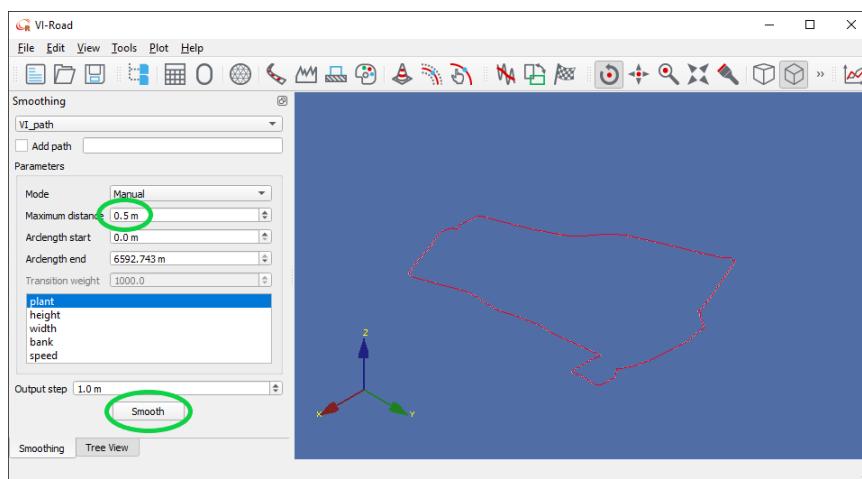
Open **VI-Road** from **VI-CarRealTime** Utilities menu as shown in the figure below.
In this way the working directory of **VI-CarRealTime** and **VI-Road** will be the same.



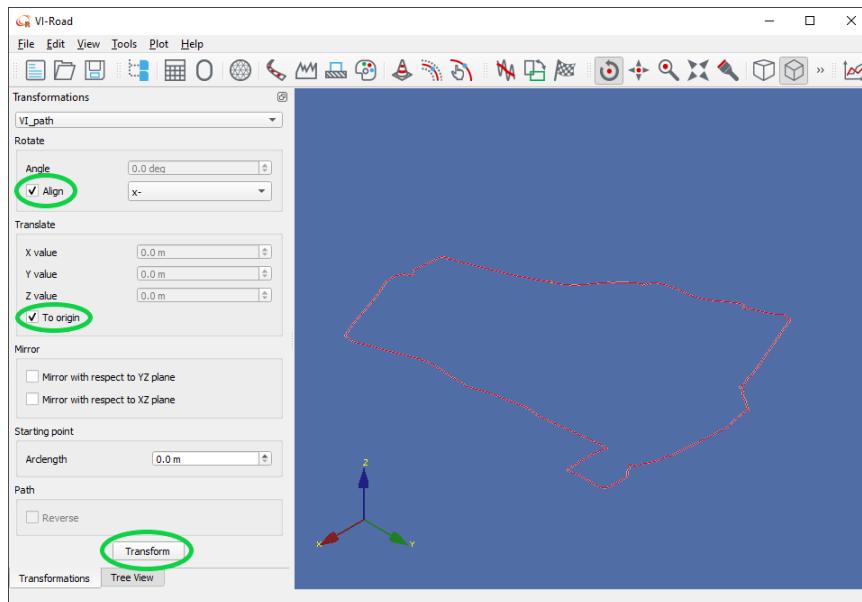
Once **VI-Road** GUI shows up, use Open tool and load ".kml" file in the "Road" field: select "**VI_path.kml**" from **viroad_shared** database. Set the Road width field to 5,0 m then click "Open" button.



Now you can display the imported road. Importing road data from others application can generate some sharp-cornered road path. A useful tool that can be used is **Smoothing** tool clicking on button circled in the next figure. The level of smoothness obtained depends on the selected maximum distance.

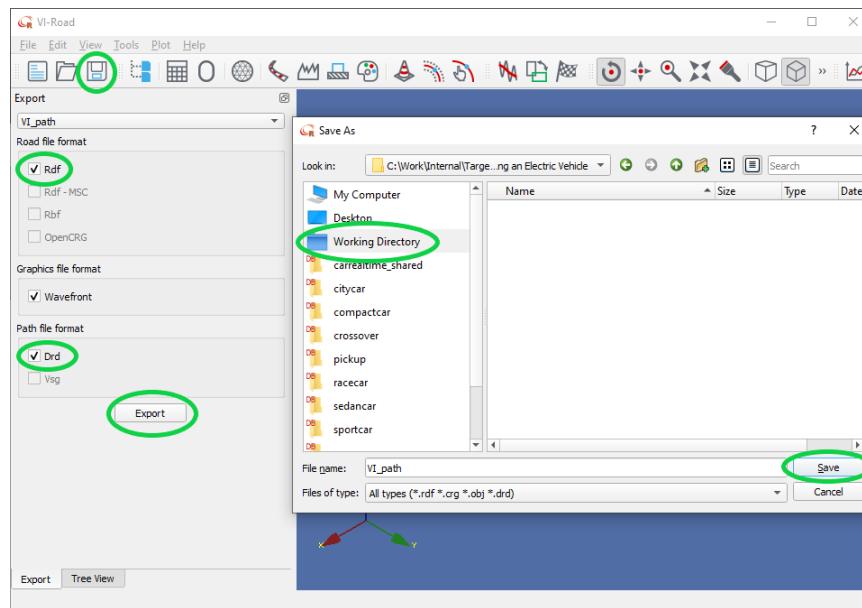


In order to have the same reference system defined in **VI-CarRealTime** it's necessary to rotate and translate the path by using Transformation tool . Use the option represented in the following figure and then click "Transform" button.



The last procedure in **VI-Road** consists in exporting both road and path data files. Ensure that "Rdf" and "Drd" flags are selected. Click on "Export" button and select "Working Directory" then "Save" button.

VI-CarRealTime

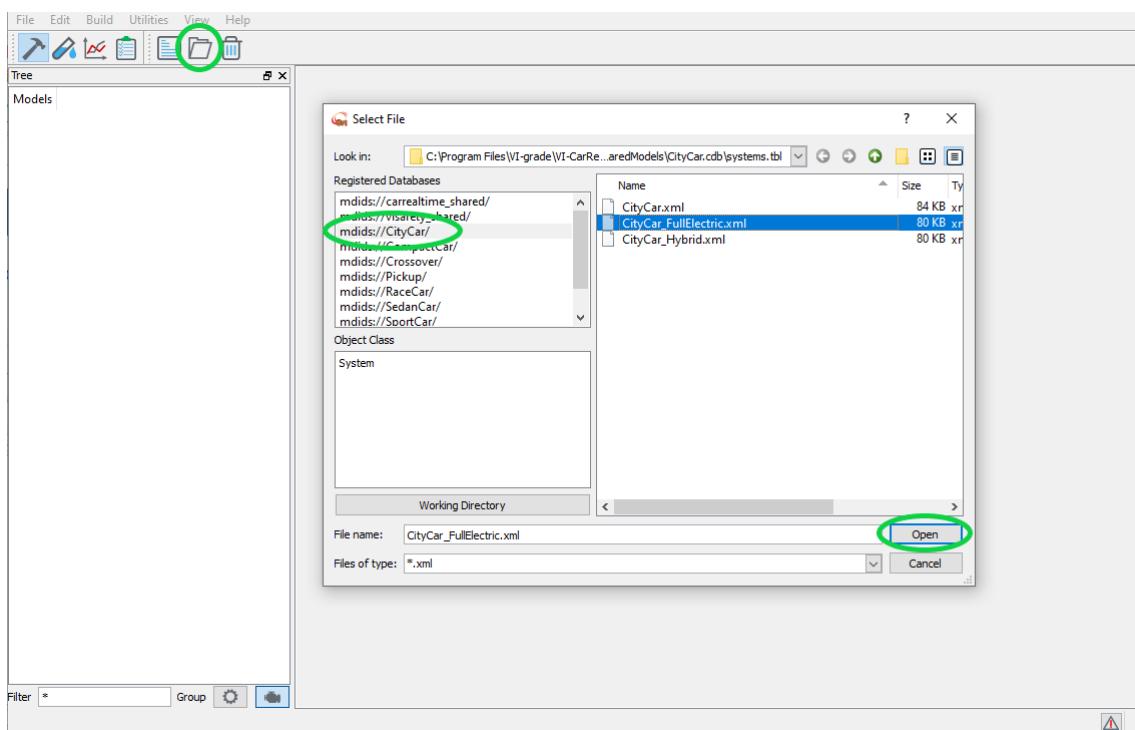


Now you can close **VI-Road** and come back to **VI-CarRealTime** window.

Creating send file

Modification of CityCar model

Now load the car model: click on "Load Model" button , select **CityCar** database on the left of the window and then open **CityCar_FullElectric.xml**

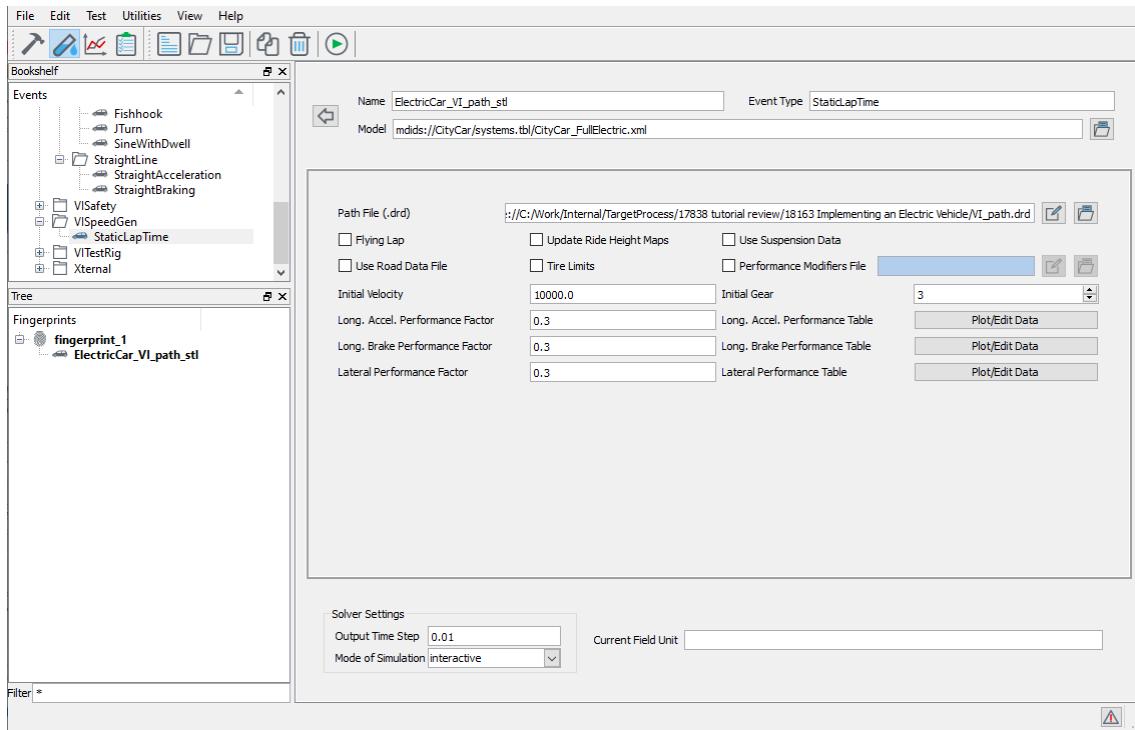


The vehicle model will appear in the tree window.

Create a send file for Matlab/Simulink

The **VI-CarRealTime** Interface for Simulink needs a file in order to link to appropriate files. For creating that, switch to Test Mode using  button.

As first step create a **StaticLap** event and rename it as "ElectricCar_VI_path_stl". Now load the "VI_path.drd" created previously in the working directory with **VI-Road**. Using  button in "Path File(.drd)" line and open the file from your working directory.



In order to generate a not excessive velocity target, insert the value **0.3** instead of 1.0 in the performance factors cells without changing the other standard values.

Long. Accel. Performance Factor	0.3
Long. Brake Performance Factor	0.3
Lateral Performance Factor	0.3

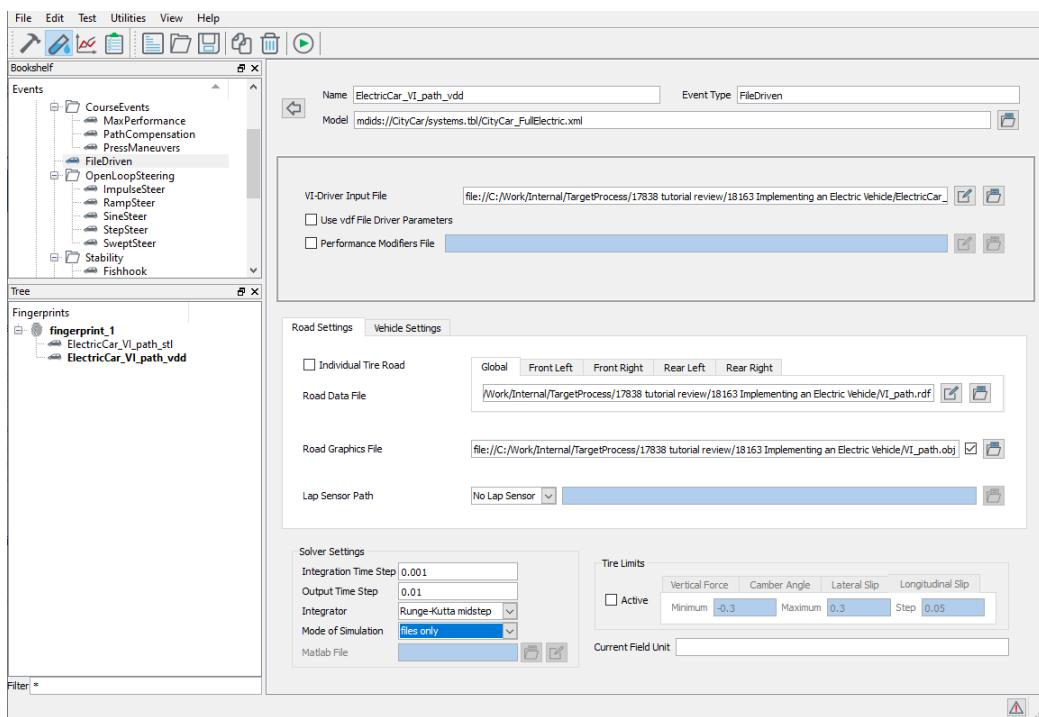
Press  button to launch the simulation.

The next step is to launch a **FileDriven** event. Double click on "FileDriven" event in events tree and rename it as "ElectricCar_VI_path_vdd".



Now click on  in "VI-Driver Input File" field. Open "**ElectricCar_VI_path_stl_vdd_event.vdf**" from your working directory created during the previous step. Open "**VI_path.rdf**" and "**VI_path.obj**" files in your working directory respectively in Road Data File field and in Road Graphics File field as shown in the figure.

In order to generate only the necessary files without launching the dynamic simulation in **VI-CarRealTime** Standalone system select "files only" in Mode of Simulation field.



Press button. It takes few seconds to generate the files.

Now you are ready to go to the next section.

Running analysis on Simulink

Run EV_model simulation

Start **MATLAB**, and therein, set the same working directory as the **VI-CarRealTime** one.

In Matlab Command Window, type and enter the command **addpath_vicrt_20** to add all the necessary paths for the co-simulation.

In MATLAB once again, open the following Simulink model: **\$VI-CarRealTimeInstallationdir\acarrt\examples\Simulink\EV_model.mdl**.

Enter the following commands in your Matlab Command Window in order to define the variables for setting the send file and the output file name need for **VI-CarRealTime** Simulink Interface:

```
vicrt_inputfile = 'ElectricCar_VI_path_vdd_send_svm.xml'
```

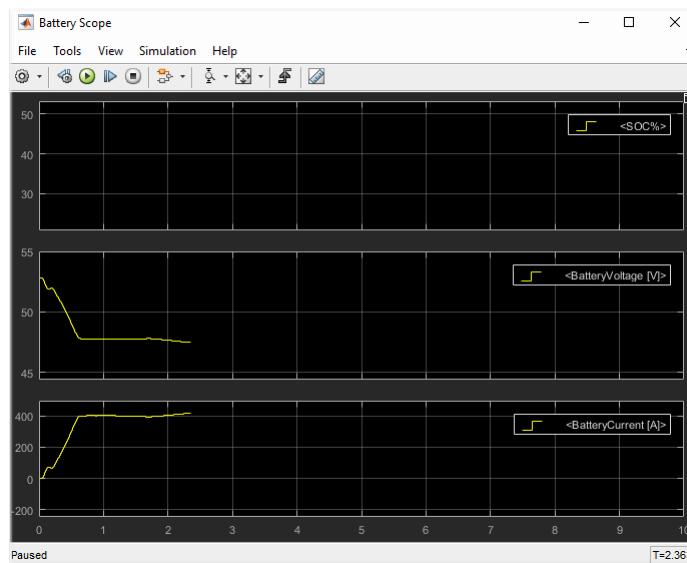
```
vicrt_outputprefix= 'ElectricCar_VI_path_vdd_NO_REGEN'
```

Before launching the simulation, it could be useful to press “CTRL + D” in order to compile the model and ensure that all the files necessary have been loaded correctly.

If there are not error or warning messages you can launch the simulation from Simulink using the play button ; otherwise be sure that working directory, **VI-CarRealTime** paths and all the variables have been correctly set.

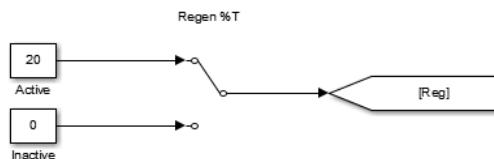
When the Simulink computation starts the appropriate ***send_svm.xml** file is used to retrieve the model parameters.

In the "TopDriveline_EMotor & Battery" block there are some scopes that are useful to inspect the electric motor and battery behaviours during simulation. For example, the next image shows the behaviour of three signals from the battery. The SOC plot starts from the value imposed by battery's mask and tends to decrease monotonously. The current pass through the battery is always positive.



Start a new simulation with regenerative braking

The next step consists in running the same event, using the same model as before, except for the fact that this time the regenerative braking model will be switched on. We can use the same exact `send_svm.xml` file used in previous simulation, but now we have to change the position of manual switch refer to regenerative braking contained in "Main_EMotor & Battery". The following figure shows the changed configuration of the switch:



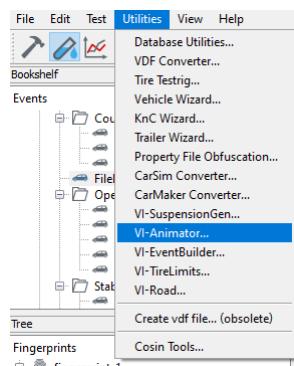
Now enter the following command in your Matlab Command Window:

```
vicrt_outputprefix= 'ElectricCar_VI_path_vdd_REGEN'
```

In this way the result files name will change and it will be possible to compare the result with the previous simulation.

From your MATLAB environment, start the Simulink simulation with the button again.

After the end of simulation compare the two simulations by using VI-Animator environment. Launch VI-Animator from **VI-CarRealTime** GUI "Utilities" menu. In this way the VI-Animator working directory and databases will be automatically set.

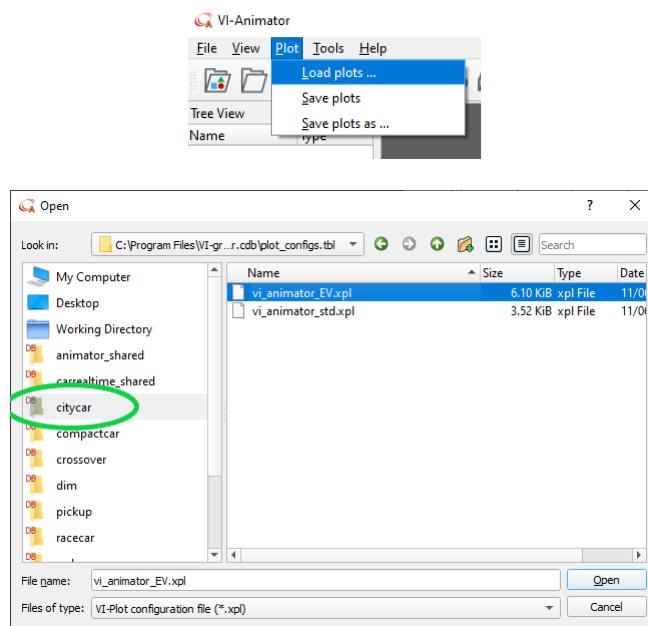


VI-CarRealTime

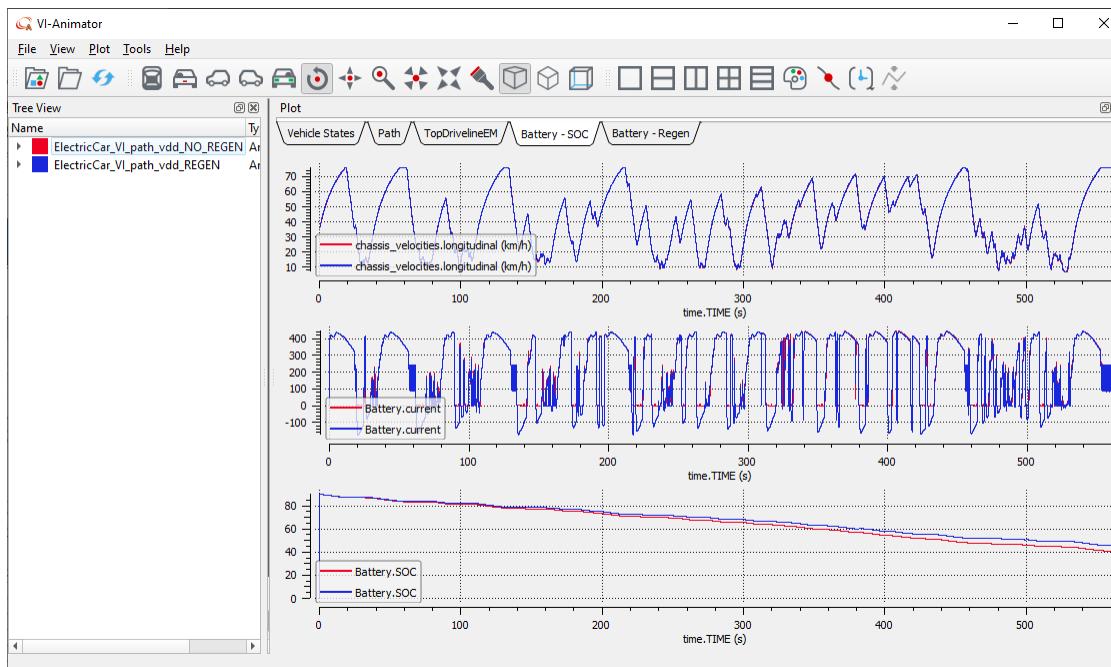
Now the two analyses have to be loaded from working directory: The name of ".res" files are

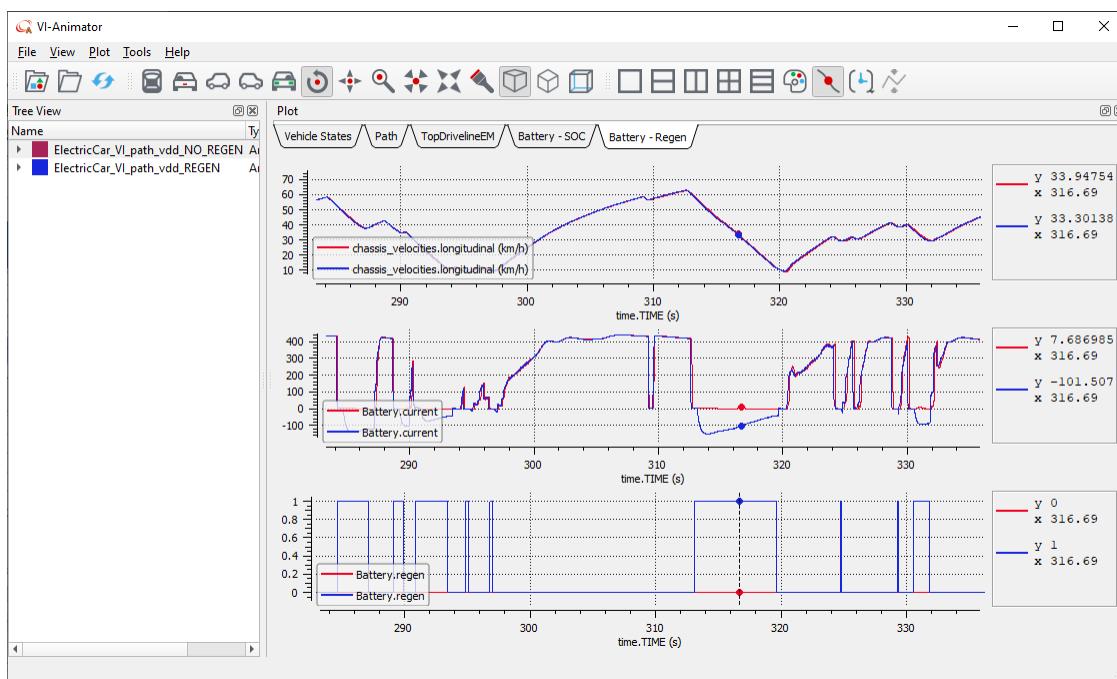
"ElectricCar_VI_path_vdd_NO_REGEN" and **"ElectricCar_VI_path_vdd_REGEN"**.

In order to compare the interesting signals between the two simulation select load "vi_animator_EV.plt" VI-Plot configuration file as shown in the next figures.



If you are looking at "Battery - SOC" tab you can appreciate the differences between the two configuration in particular in the SOC signal behaviour





Implementing an Hybrid Vehicle

Implementing an Hybrid Vehicle with torque management system

The following tutorial introduces how to use the **VI-CarRealTime** MATLAB/Simulink interface in order to run a simulation of HV vehicle with an external torque management logic.

We are going to analyse a 4WD vehicle with an electric motor for each front wheel and an internal combustion engine that supplies torque at the rear wheels through a DTC transmission.

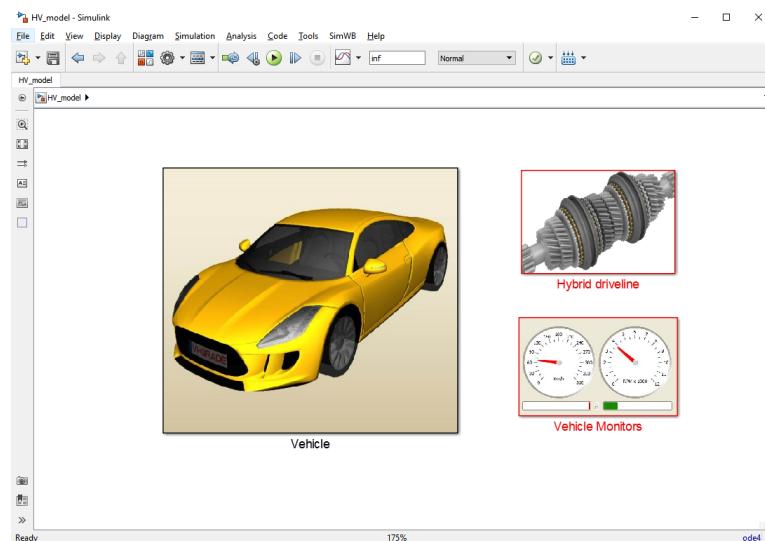
The tutorial is divided into the following sections:

- [Introduction of Simulink model](#)
- [Starting from kml path](#)
- [Creating send file](#)
- [Running analysis on Simulink](#)

Introduction of Simulink model

Presentation of Simulink Model

The simulink model "HV_model.mdl" is contained in **\$VI-CarRealTimeInstallationdir\acarrt\examples\Simulink**. The driveline block contains all the electric components of the drive system.



Starting from **VI-CarRealTime** 18 you can use all the motors defined internally in **VI-CarRealTime** and manage each motor torque externally (in this case by using Simulink/**VI-CarRealTime** interface). The deactivation of internal motor is not required.

Moreover, for each motor, it is possible to specify how to manage the torque.

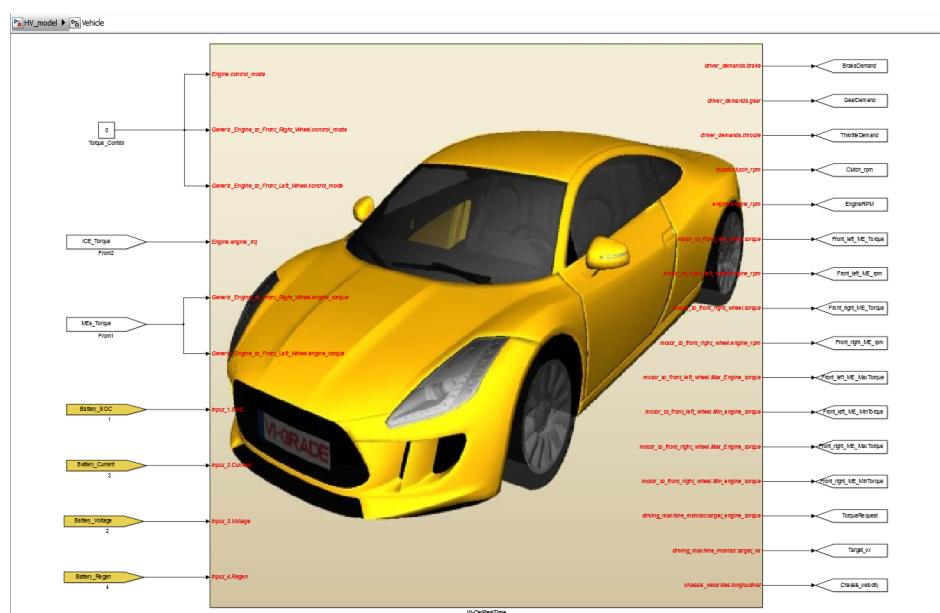
Two ways are available:

1. use a throttle scaling factor
2. directly providing the torque value.

In the second way VI-CarRealTime internal traction control and rpm limiter are bypassed.

In this tutorial it will be shown how to manage 2 electric motors and 1 internal combustion engine through torque signals.

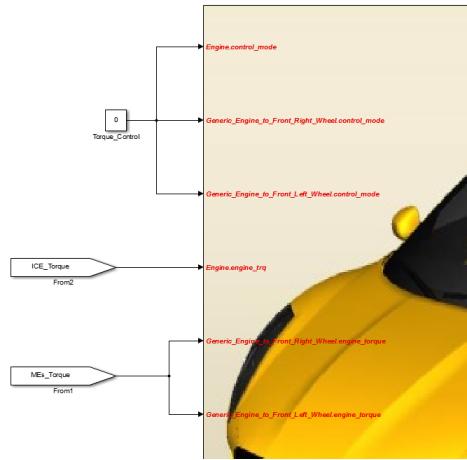
In the vehicle block the main signals that communicate with **VI-CarRealTime** Matlab/Simulink interface are those represented in the following figure:



In the figure below we can see the torque signals and "Control mode" values as input for **VI-CarRealTime**.

The control mode signal can assume 0 or 1 value: if the value is 0, the motor will be controlled by the torque otherwise if the value is 1, the motor will be controlled by means of throttle scaling factor (if not defined, value 1 will be considered).

For each motor, the maximum and the minimum torques (defined internally in **VI-CarRealTime**) limit the input torque signal.

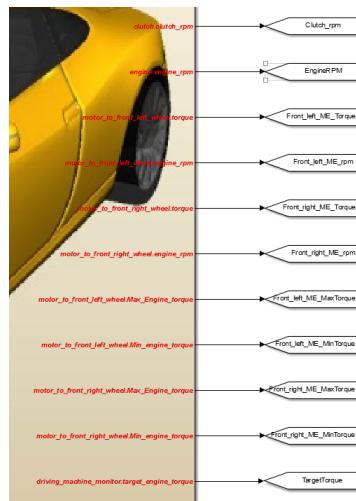


The last 4 channels are related to battery model:

- "Battery_SOC"
- "Battery_Current"
- "Battery_Voltage"
- "Battery_Regen"

Having these input signals is important in order to write them in the result file and facilitate their visualization in **VI-Animator**.

In the following figure we can see the signals used to create the logic of hybrid system:

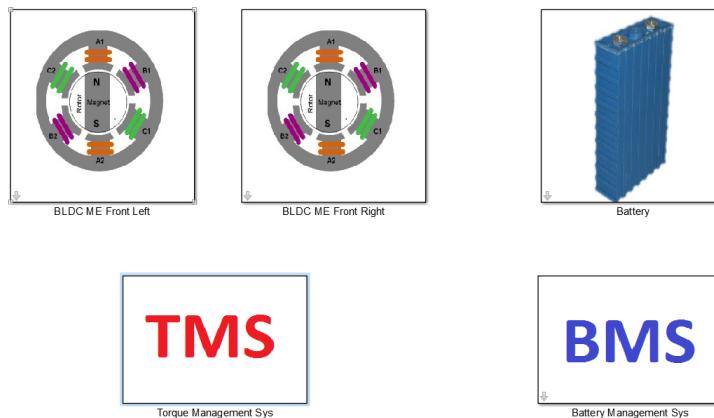


"driving_machine_monitor:target_engine_torque" signal is the request in terms of torque that the driver demands in order to reach a specific target acceleration or target velocity profile.

This signal will be used to redistribute the torque request to the different torque sources.

Looking inside the "Driveline" block there are:

- Two electric motors
- Battery
- Battery management system
- Torque management system

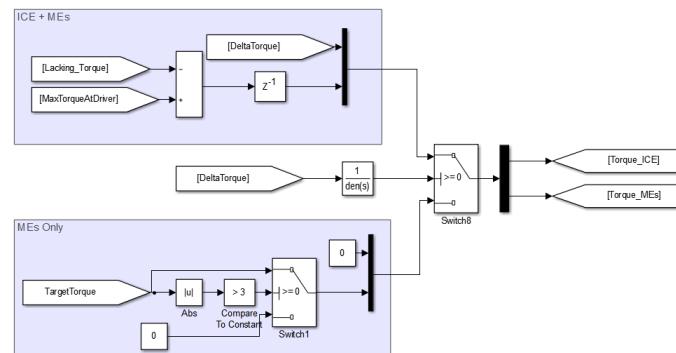


The electric motor model represents a BLDC motor: starting from torque request and motor speed the block gives the value of power delivered or recovered. To calculate the electrical signals the Park transform has been used. Using block's mask it is possible to configure the principal values of BLDC motor.

The Battery block can simulate different type of batteries: the behaviour of Li-ion, Lead acid, NiCd or NiMh battery can be represented. Using the mask of this block, the type of battery and the most important parameters can be set.

The "Battery Management System" is needed to limit the electric motor behavior: this block controls the SOC limit and the maximum value of current delivered or absorbed by the battery. The current limit can be set by using the respective mask.

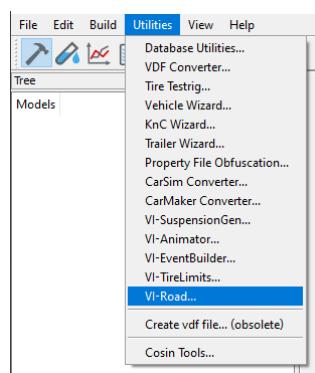
The "Torque Management System" block is an example of a logic to manage the different motors. In this case the torque request from the driver is delivered through the front electric motors when the torque request is sufficiently low. When the driver demands a higher torque the internal combustion engine will support the electric motors.



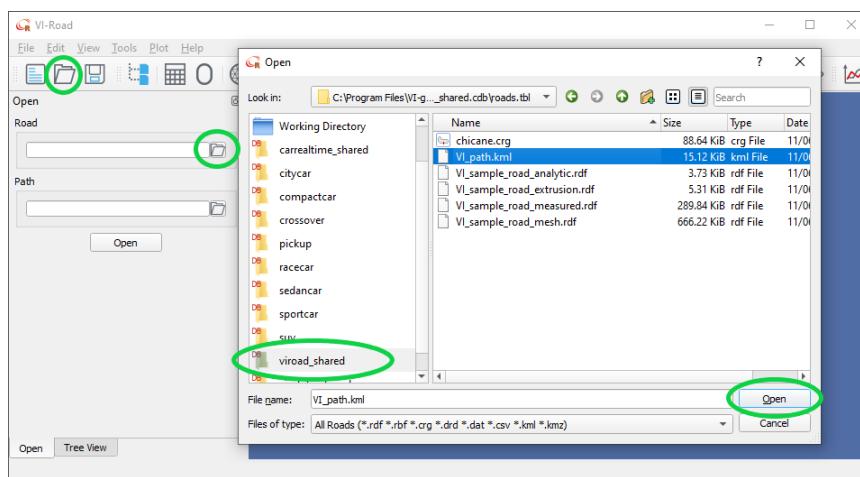
Starting from kml path

Create a road data file from kml in VI-Road

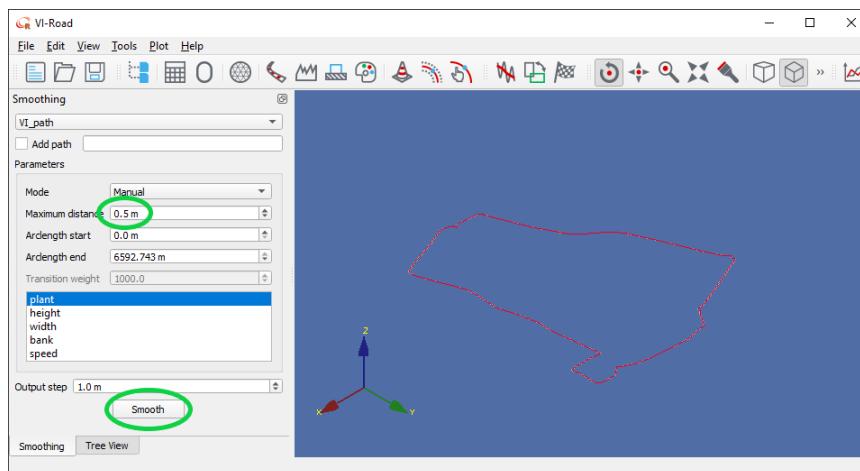
Open **VI-Road** from **VI-CarRealTime** Utilities menu as shown in the figure below. In this way the working directory of **VI-CarRealTime** and **VI-Road** will be the same.



Once **VI-Road** GUI shows up, use Open tool and load ".kml" file in the "Road" field: select "**VI_path.kml**" from **viroad_shared** database. Set the Road width field to 5.0 m then click "Open" button.

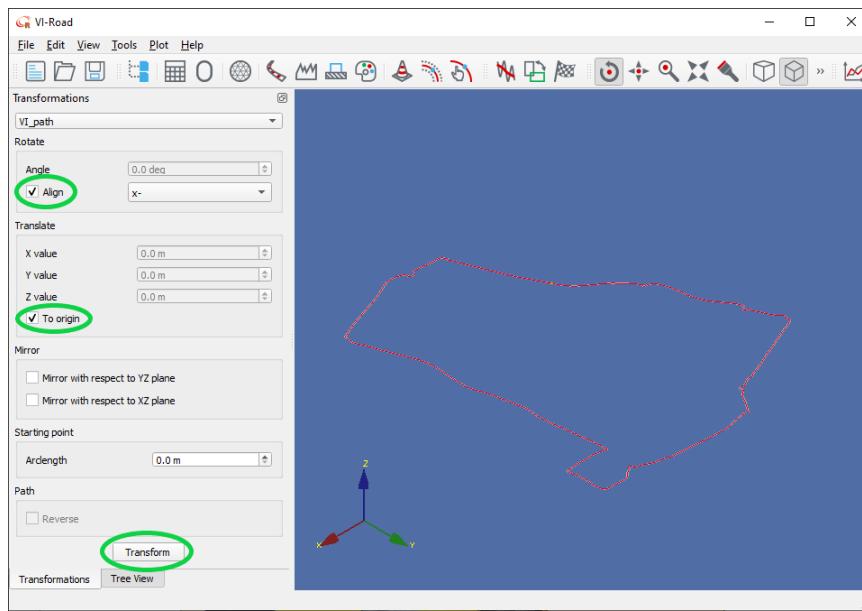


Now you can display the imported road. Importing road data from others application can generate some sharp-cornered road path. A useful tool that can be used is **Smoothing** tool clicking on button circled in the next figure. The level of smoothness obtained depends on the selected maximum distance.

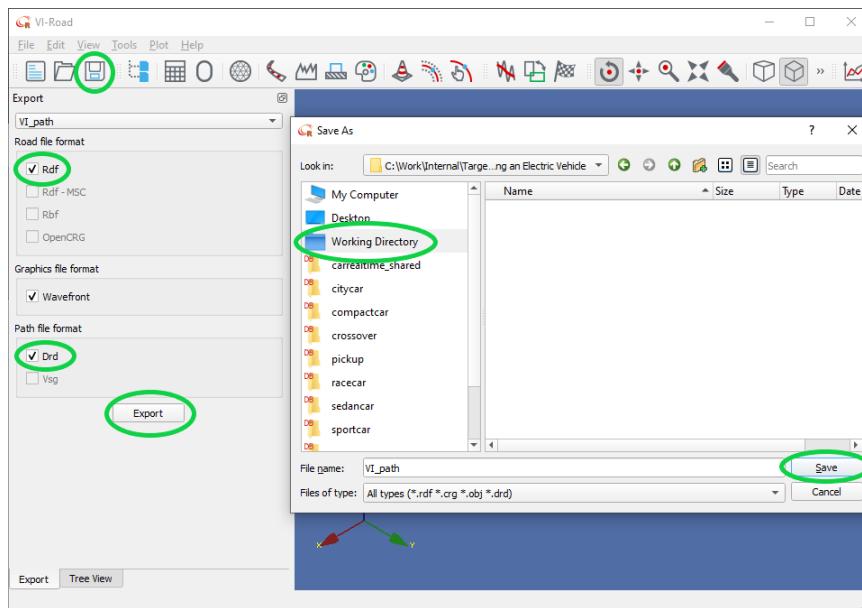


In order to have the same reference system defined in **VI-CarRealTime** it's necessary to rotate and translate the path by using Transformation tool . Use the option represented in the following figure and then click "Transform" button.

VI-CarRealTime



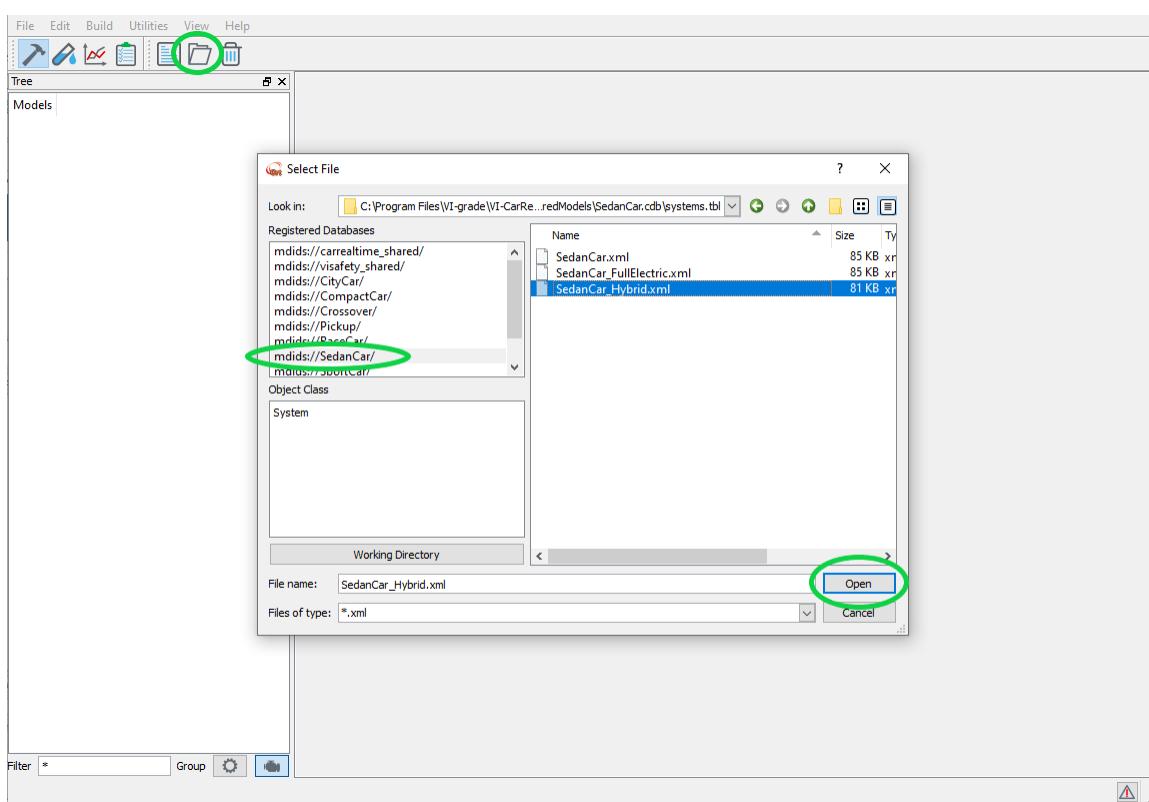
The last procedure in **VI-Road** consists in exporting both road and path data files. Ensure that "Rdf" and "Drd" flags are selected. Click on "Export" button and select "Working Directory" then "Save" button.



Now you can close **VI-Road** and come back to **VI-CarRealTime** window.

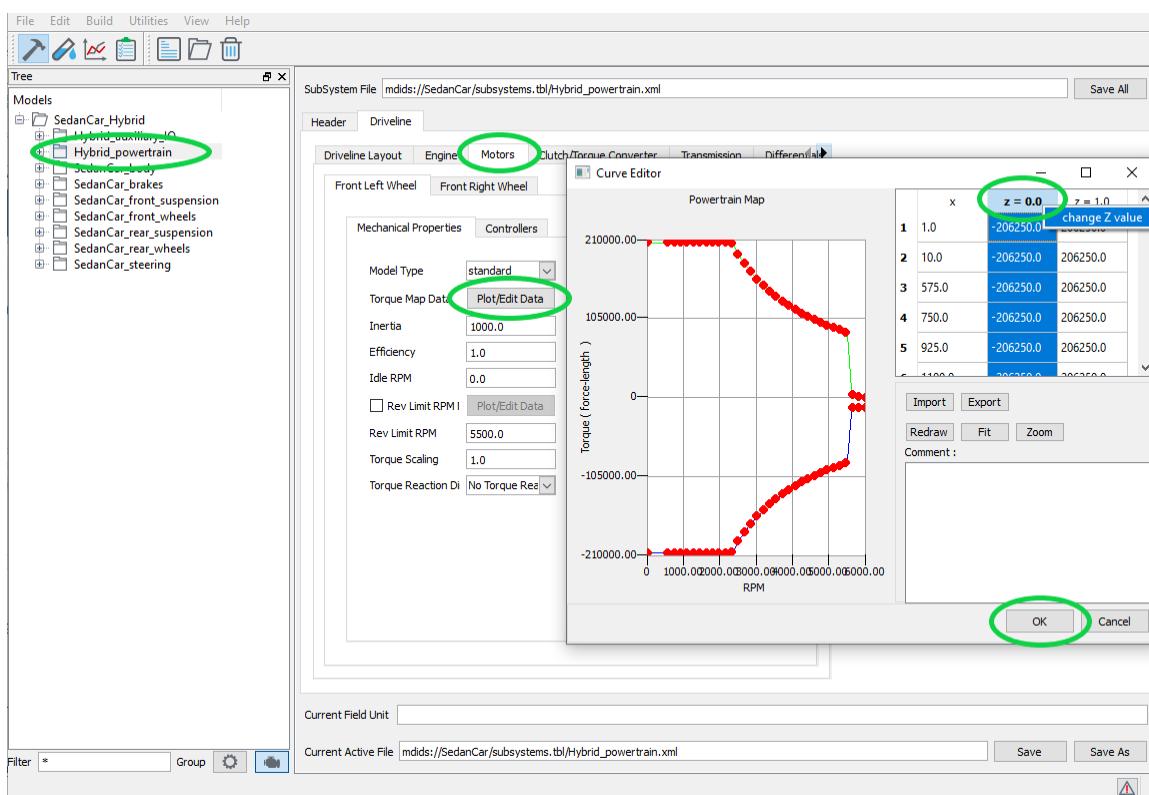
Creating send file**Modification of a SedanCar model**

Now load the car model: click on "Load Model" button  , select **SedanCar** database on the left of the window and then open **SedanCar_Hybrid.xml**



The vehicle model will appear in the tree window on the left of the GUI.

Select the Hybrid_powertrain subsystem and then from the Motors tab open the Torque Map Data and change the first Z value to 0. Repeat for the Front Right Wheel motor tab.



Create a send file for Matlab/Simulink

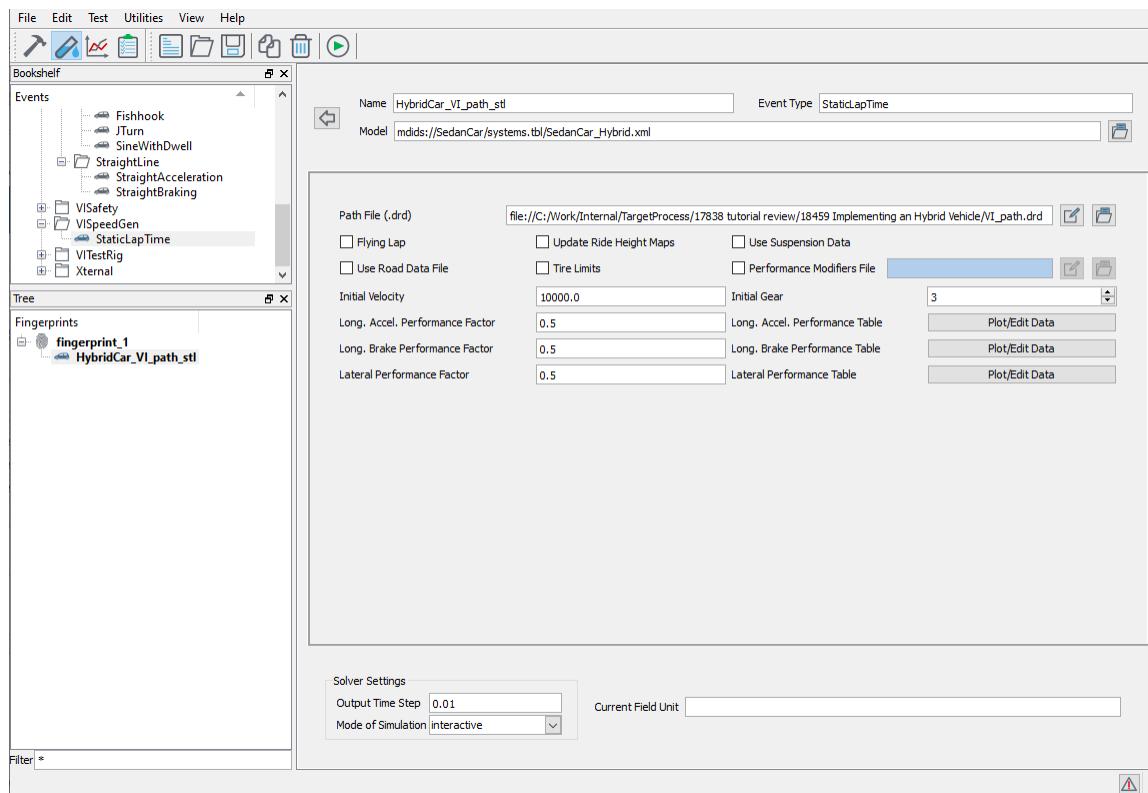
VI-CarRealTime

The **VI-CarRealTime** Interface for Simulink needs a file in order to link to appropriate files. For creating that, switch to **Test Mode** by using  button.

As first step create a **StaticLap** event and rename it as "HybridCar_VI_path_stl". Now load the "VI_path.drd"

created previously in the working directory with **VI-Road**. Use  button in "Path File(.drd)" line and open the file from your working directory.

In order to generate a not excessive velocity target, insert the value **0.5** instead of 1.0 in the performance factors cells without changing the others standard values.



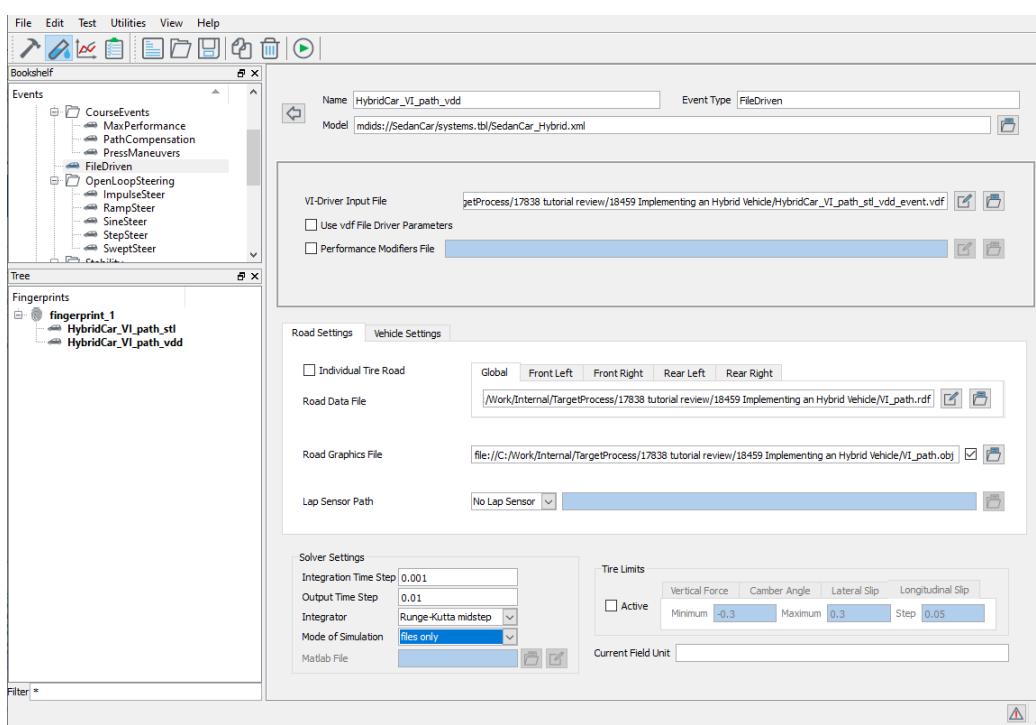
Press  button to launch the simulation.

The next step is to launch a **FileDriven** event. Double click on "FileDriven" event in events tree and rename it as "HybridCar_VI_path_vdd".



Now click on  in "VI-Driver Input File" field. Open "HybridCar_VI_path_stl_vdd_event.vdf" from your working directory created during the previous step. Open the "VI_path.rdf" and "VI_path.obj" files in your working directory respectively in Road Data File field and in Road Graphics File field as shown in the figure.

In order to generate only the necessary files without launching the dynamic simulation in **VI-CarRealTime** Standalone system select "files only" in "Mode of Simulation" field.



Press button. It takes few seconds to generate the files.

Now you are ready to go to the next section.

Running analysis on Simulink

Run HV_model simulation

Start **MATLAB**, and therein, set the same working directory as **VI-CarRealTime** one. In Matlab Command Window, type and enter the command **addpath_vicrt_20** to add all the necessary paths for the co-simulation.

In MATLAB once again, open the following Simulink model: **\$VI-CarRealTime\Installationdir\acarr\examples\Simulink\HV_model.mdl**.

Enter the following commands in your Matlab Command Window in order to define the variables for setting the send file and the output file name need for **VI-CarRealTime** Simulink Interface:

```
vicrt_inputfile = 'HybridCar_VI_path_vdd_send_svm.xml'
```

```
vicrt_outputprefix= 'HybridCar_VI_path_vdd'
```

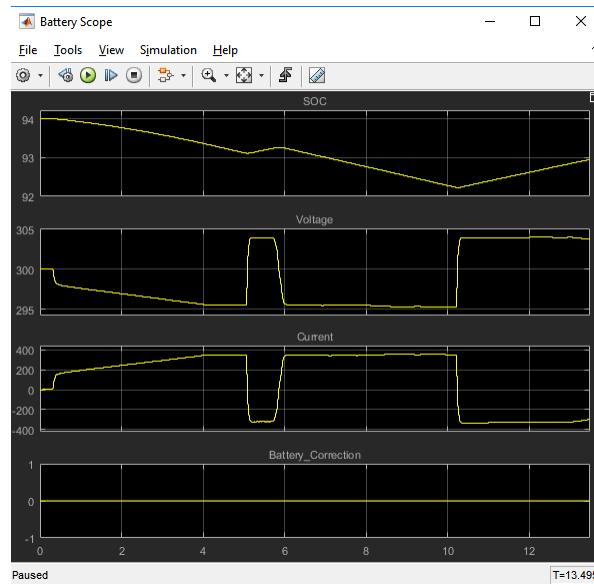
Before launching the simulation, it could be useful to press “CTRL + D” in order to compile the model and ensure that all the files necessary have been loaded correctly.

If there are not error or warning messages you can launch the simulation from Simulink using the play button . Otherwise be sure that working directory, **VI-CarRealTime** paths and all the variables are correctly set.

When the Simulink computation starts the appropriate ***send_svm.xml** file is used to retrieve the model parameters.

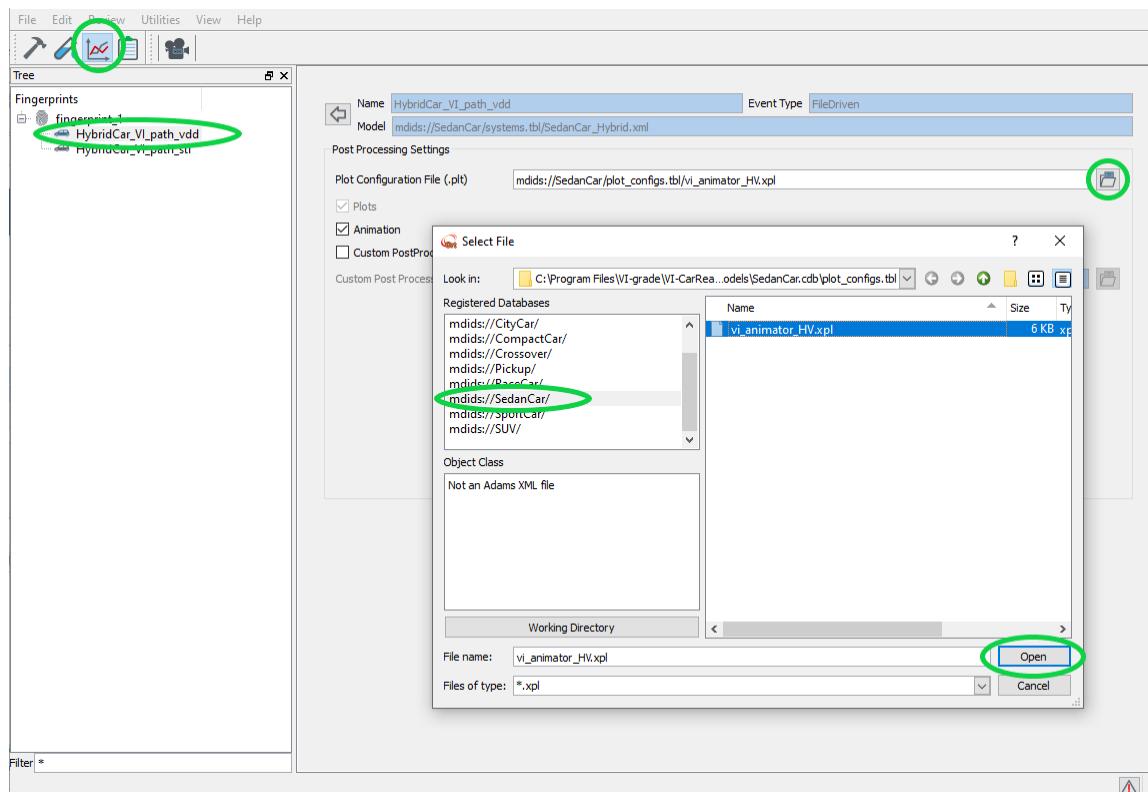
In the “Vehicle Monitor” block there are some scopes that are useful to inspect the electric motor and battery behaviours during simulation. For example, the next image shows the behaviour of four signals related to the battery. The SOC decreases when the driver demands a positive torque and the opposite happens when the torque request is negative and a regenerative braking is used.

VI-CarRealTime

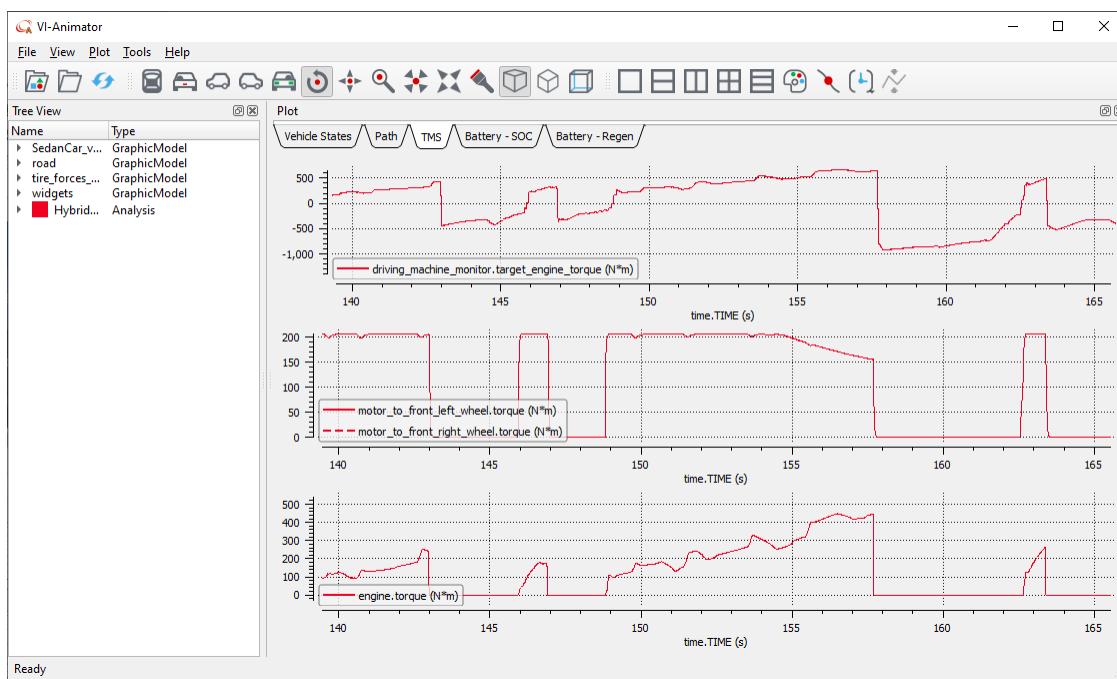


At the end of simulation we can see the results by using VI-Animator environment. Open VI-Animator using **VI-CarRealTime GUI**. Switch to **Review** mode by pressing the "Review Mode" button.

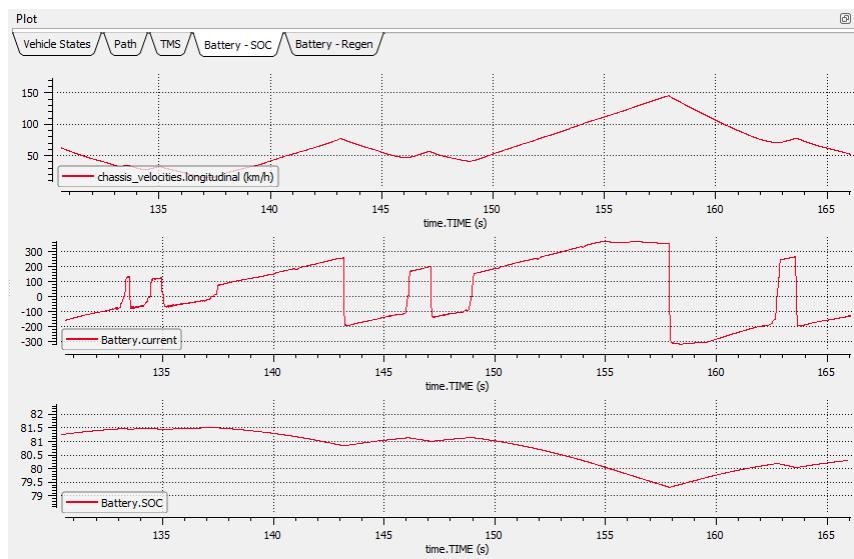
Selecting "HybridCar_VI_path_vdd" result in the tree window, change the plot configuration file (.xpl) clicking on "open" button and select, from SedanCar database, "vi_animator_HV.xpl". Then press "Execute postprocessing" button in the Toolbar. The result analysis file will be loaded in VI-Animator.



In "TMS" tab it is clear how the torque is split between the various motors.



If you look at "Battery - SOC" tab you can appreciate the effect of regenerative braking in particular in the SOC signal behaviour.



Connecting External Steering Model

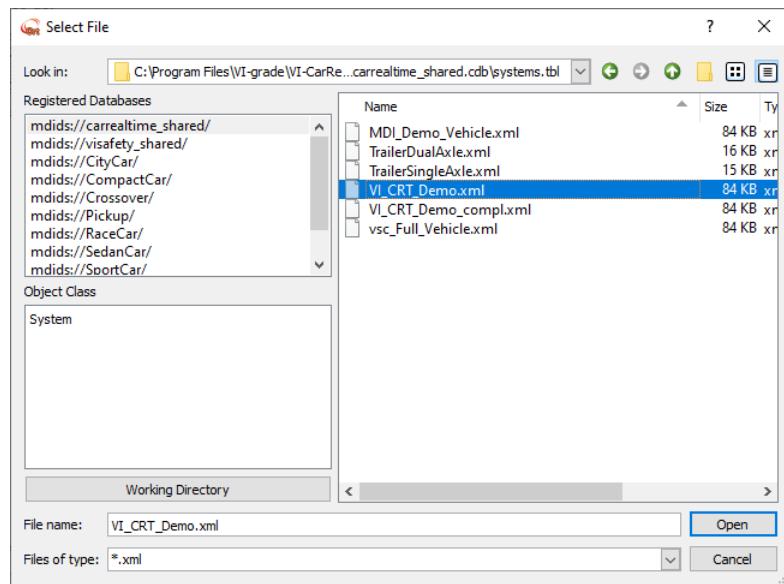
The aim of this tutorial is to instruct the user on how to connect an external steering model implemented in Simulink to a VI-CarRealTime full vehicle. VI-CarRealTime internal steering column will be switched off and rack displacement runtime input will be provided by Simulink to drive the vehicle.

Please refer to [Rack-Pinion Steering](#) chapter for all the details about how to interface the steering system with the full-vehicle model.

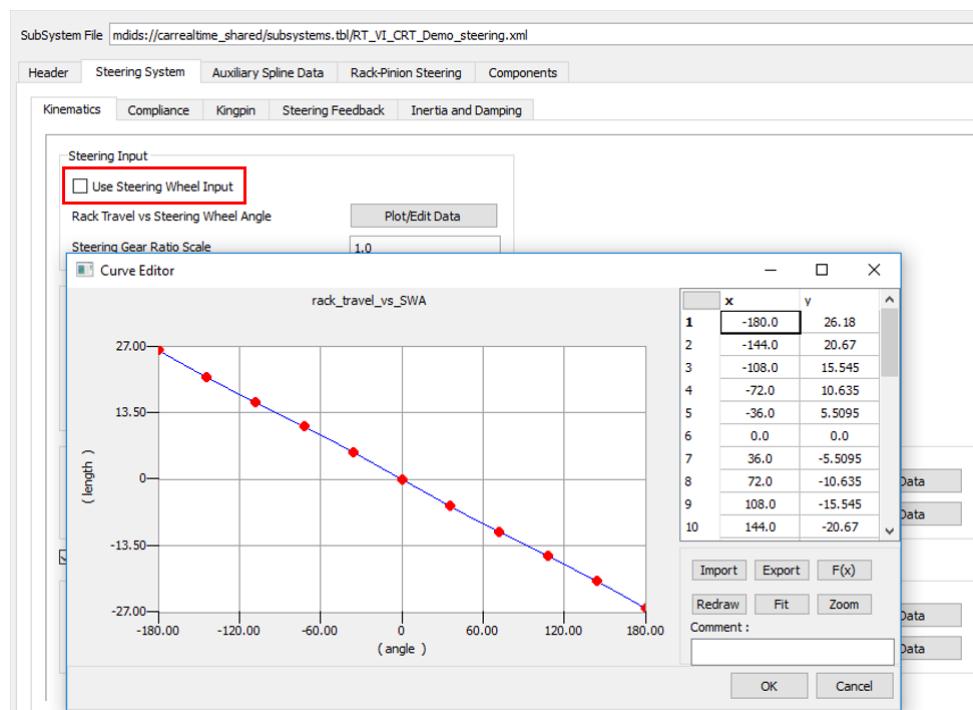
Start a new VI-CarRealTime session using the **vicrt20** command from a DOS (cmd) shell, or by using the VI-CarRealTime shortcut from the Windows Start Menu.

Starting in **Build mode**, open the model **VI_CRT_Demo.xml** from the shared database.

VI-CarRealTime



Since the external steering model provides a rack displacement input to VI-CarRealTime internal model, it is mandatory to switch off [Use Steering Wheel Input](#) toggle button in steering subsystem Kinematics panel as shown in the picture below:



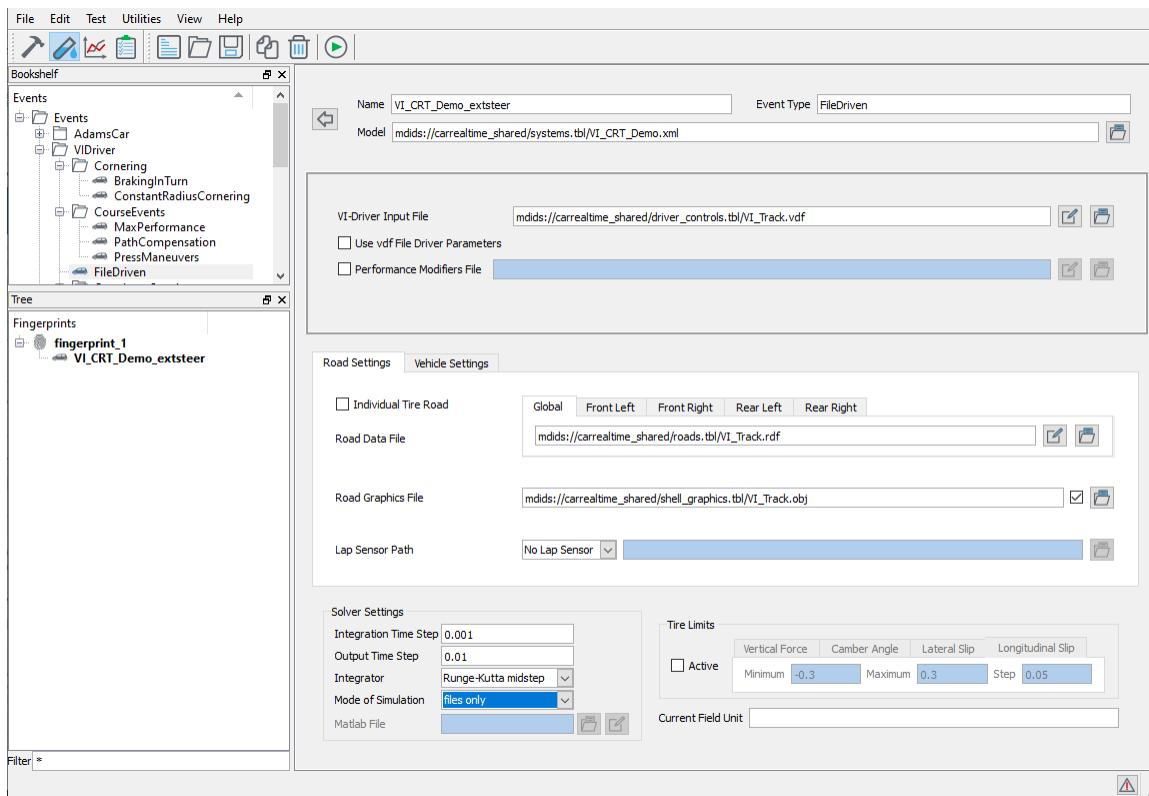
In this way all the steering curves will have *rack displacement* as independent variable.

Note: when using an external steering model in closed loop events, the user must modify [rack travel vs. steering wheel angle](#) spline according to the external steering ratio: VI-CarRealTime solver doesn't directly use such spline, but the slope of the curve is used to initialize VI-Driver lateral controller.

Switch to **Test Mode** using the  button.

Create a new **File Driven** event (VIDriver -> FileDriven) using **VI_CRT_Demo** model and set the following parameters:

- **VI_Track.vdf** from the carrealtime_shared database as **VI-Driver Input File**;
- **VI_Track.rdf** from the carrealtime_shared database as **Road Data File**;
- **VI_Track.obj** from the carrealtime_shared database as **Road Graphics File**;
- select **files_only** as **Mode of Simulation**.
- rename the event **VI_CRT_Demo_extsteer** as shown in the picture below.

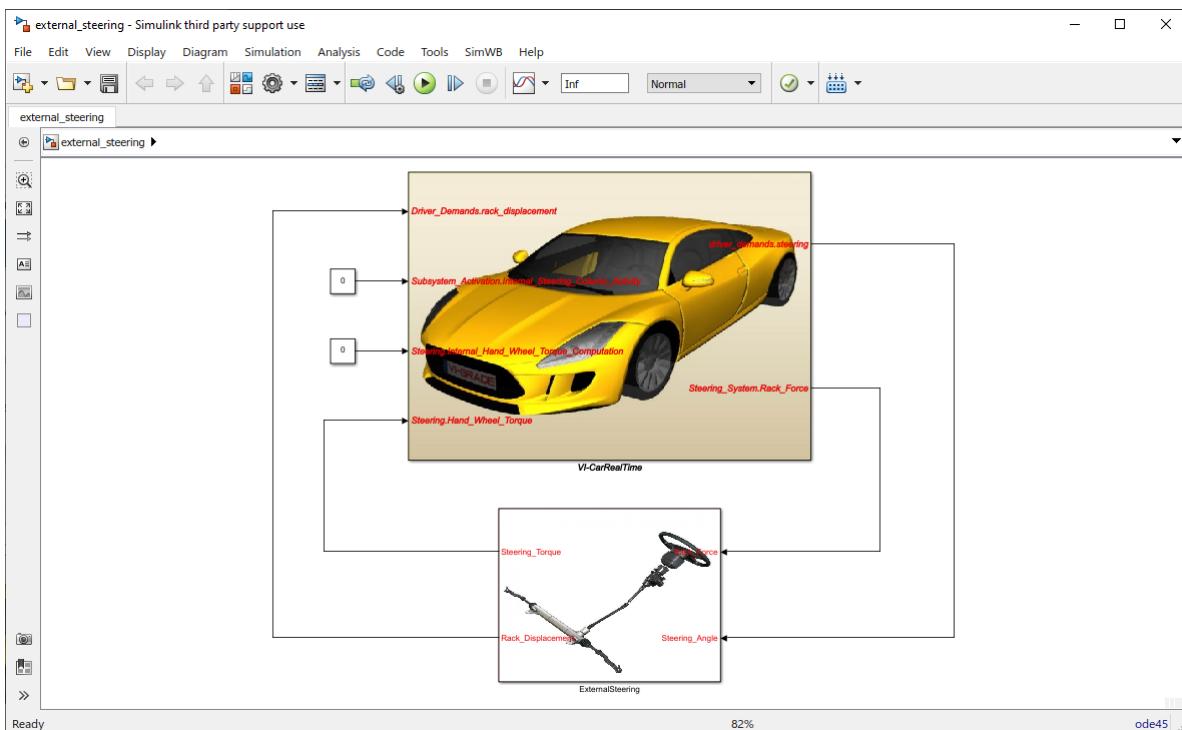


Run the event by pressing button in VI-CarRealTime so that `svm_send.xml` file will be created in the working directory; such file will be used for MATLAB and Simulink simulations.

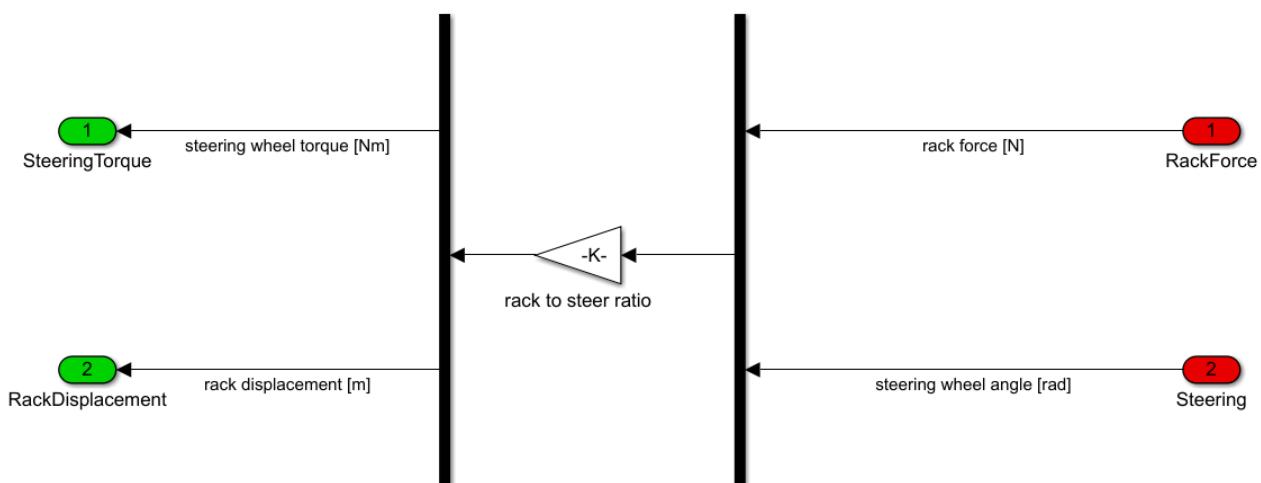
Start **MATLAB**, and therein, set the same working directory as the VI-CarRealTime one.

In MATLAB Command Window, type and enter the command `addpath_vicrt_20` to add all the necessary paths for the co-simulation.

In MATLAB once again, open the following Simulink model: **\$VI-CarRealTimeInstallationdir\acarr\examples\Simulink\external_steering.mdl**



The model represents a very simple external steering model, which computes the rack displacement as the product between VI-CarRealTime model steering wheel angle and a constant rack to steering ratio:



The inputs to VI-CarRealTime subsystem are:

- **Driver_Demands.Rack_Displacement** computed in Simulink;
- **Subsystem_Activation.Internal_Steering_Column_Activity** flag set to 0 in order to switch off internal steering system column.

Note: both VI-CarRealTime subsystem input and output units are in **SI**.

Enter the following command in your MATLAB Command Window:

```
vicrt_inputfile = 'VI_CRT_Demo_extsteer_send_svm.xml'
output = ''
```

This command sets a variable which references the send_svm file that stores all the VI-CarRealTime model information; this variable is used in the VI-CarRealTime Simulink block.

Start the co-simulation with the  button in the upper Simulink window toolbar.

When the simulation is completed, in the working directory the analysis result file is written.

Switch back to VI-CarRealTime, browse to [Review mode](#), select **VI_CRT_Demo_extsteer** analysis and launch VI-Animator to review results.



Rack Displacement input provided by Simulink steering model allows VI_CRT_Demo model to complete the VI_Track lap.

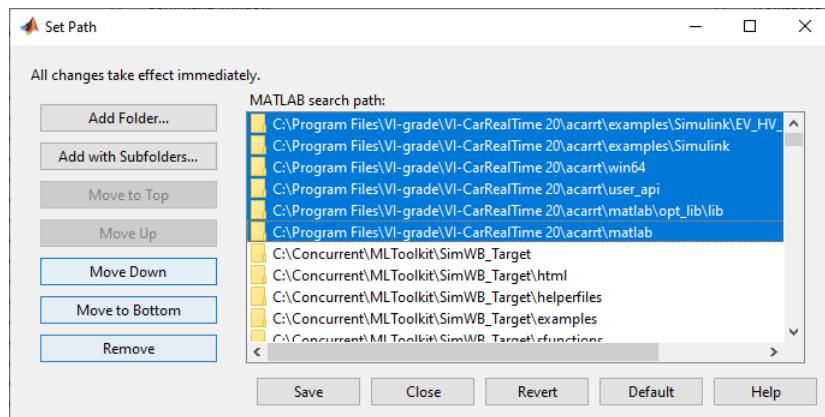
VI-CarRealTime Solver Plugin Export from Simulink

This chapter is about the export of a Simulink Model as a VI-CarRealTime solver plug-in, using Simulink Coder / Real time workshop.

Note: Exporting a Simulink Model as a VI-CarRealTime solver plug-in is only possible with MATLAB 2015a or higher.

Note: For the definition of the VI-CarRealTime inputs in the Simulink model, it is only possible to use a send.xml with the VI_CRT_Demo_output_map.xml.

In order to use the VI-CarRealTime systems target in MATLAB/Simulink the MATLAB search path must be properly updated. To do so, open MATLAB in Windows and enter the "addpath_vicrt_20" script (without quotes) into the MATLAB command window. This command adds the necessary VI-CarRealTime directories to the MATLAB search path for the current session (you would need to repeat the aforementioned procedure if MATLAB were closed and reopened). In order to keep the VI-CarRealTime paths saved after running "addpath_vicrt_20", the MATLAB search path list must be saved within the Set Path menu from MATLAB's toolbar (simply press the Save button once your Set Path window contains the same highlighted directories shown in the screen-shot below).



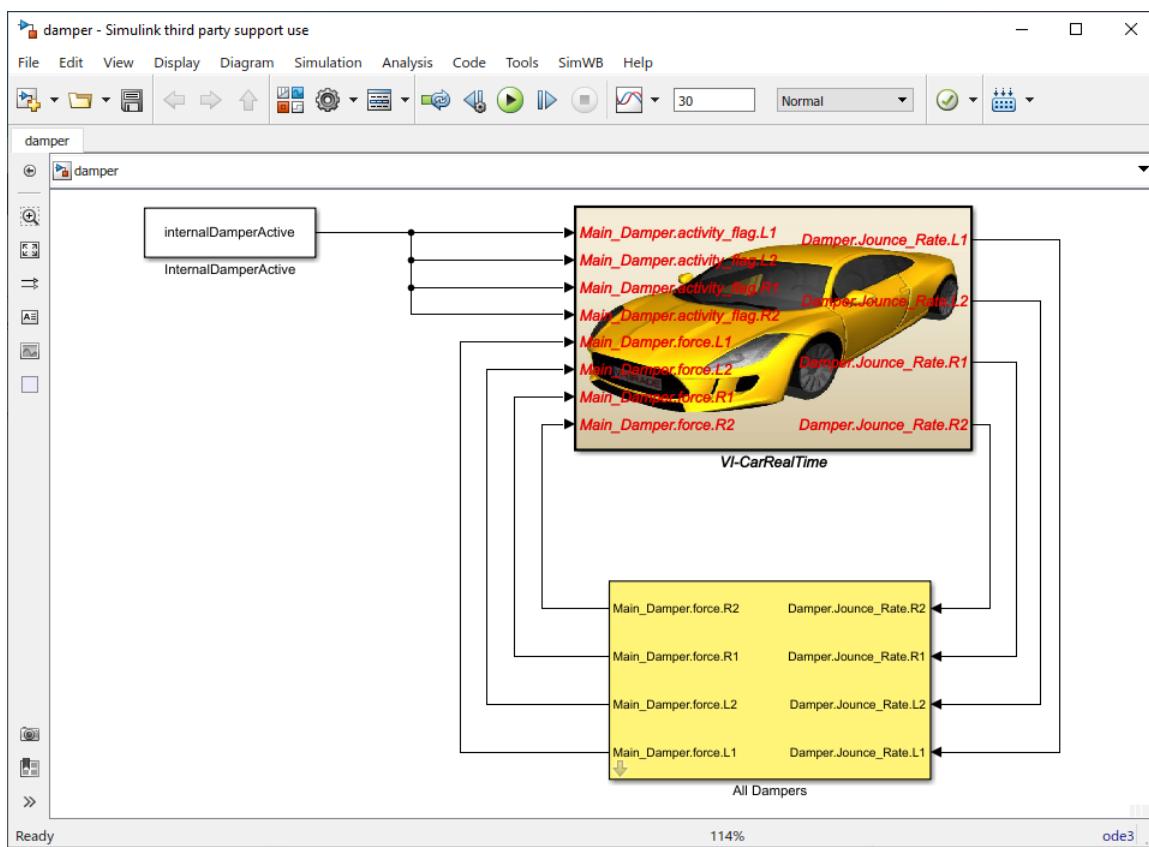
The tutorial is composed by the following topics:

- [Model Configuration](#)
- [Model I/O](#)
- [Model Parameters](#)
- [Model Build/Export](#)

Model Configuration

This tutorial assumes that the user already knows the VI-CarRealTime MATLAB/Simulink interface, and knows how to generate an event file to be executed in MATLAB. More detail can be found in the tutorial dedicated to [Co-Simulation](#).

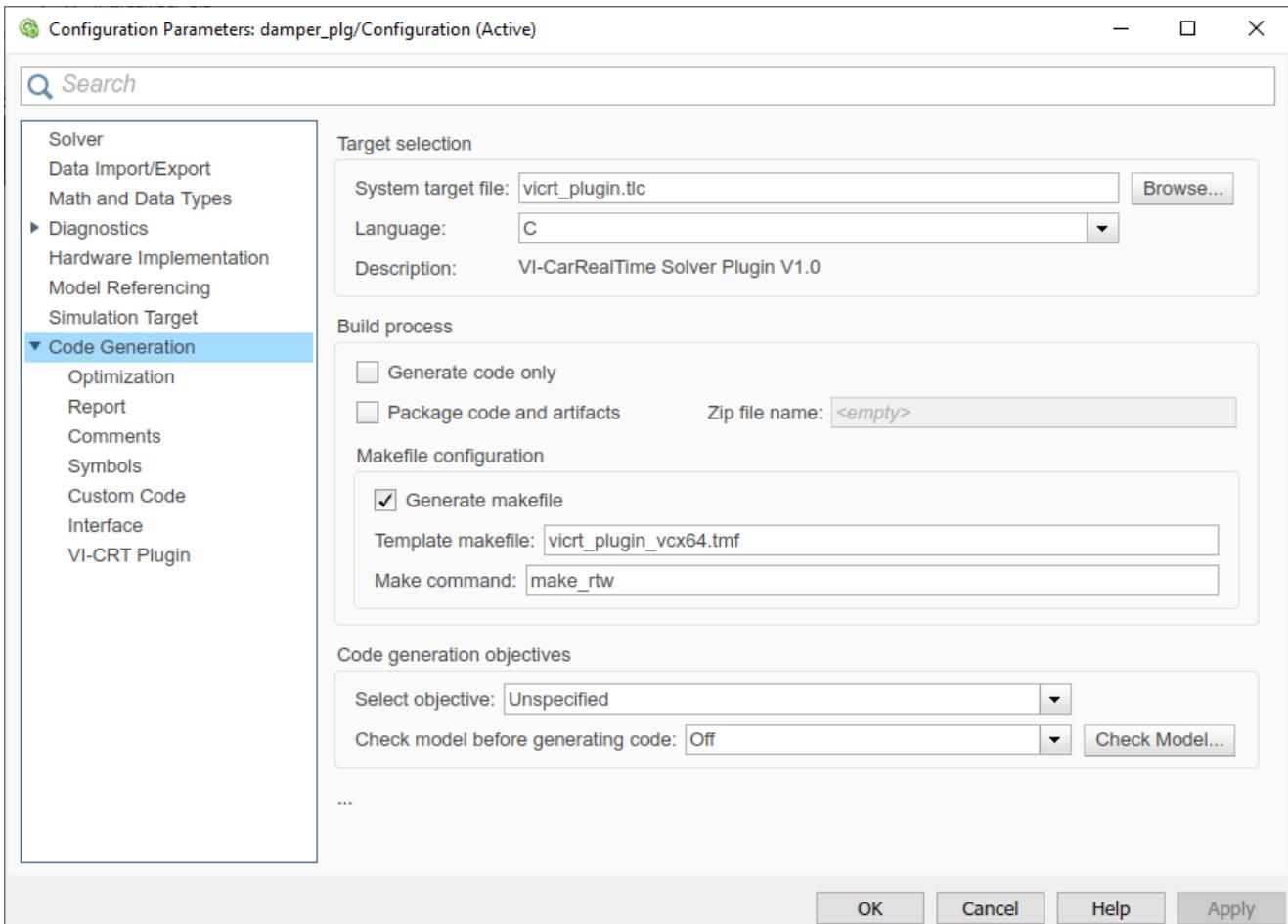
In MATLAB open the following Simulink model: \$VI-CarRealTimeInstallationdir\acarrt\examples\Simulink\damper.mdl



Save the model to the current folder with name `damper_plg`, to indicate that it will be update for the export of the VI-CarRealTime solver plug-in.

To configure the model properly for the export of a VI-CarRealTime solver plug-in:

- Open the **Configuration Parameters** dialog.
- Select **Solver** and make sure that Solver Type is set to **Fixed-step**, and Stop time to **Inf**.
- Select **Code Generation** and press the **Browse...** button in the Target Selection group.
- In the **System Target File Browser** search **vicrt_plugin.tlc**, select it and press **OK** to confirm.
- Choose the proper **.tmf** file in the **Template makefile** field according to the target machine & compiler:
 - `vicrt_plugin_vc64.tmf` (for Microsoft Visual C compiler Windows)
 - `vicrt_plugin_mingw64.tmf` (for MinGW compiler Windows)
 - `vicrt_plugin_ccur.tmf` (for Linux).
- In **Code Generation/VI-CRT Plugin** set the VI-CRT TOPDIR with the VI-CarRealTime installation path (the default installation path is `C:\Program Files\VI-grade\VI-CarRealTime 20`). If needed, set also PACKNGO MATLAB ROOT which represents the path of the Matlab folder extracted from the PackNGo archive (generated when the option *Package code and artifacts* is checked).
- In **Code Generation/Symbols** check the Maximum identifier length and set it to 80. This number represent the longest name for the VI-CarRealTime inputs.
- Press **Apply** in the **Configuration Parameters** dialog to apply changes and keep it open.



For the purpose of this tutorials we want to export all the model parameters to the [Auxiliary Subsystem](#), so that it will be possible to tune them directly in VI-CarRealTime Build Mode. To make sure that the parameters are available in the generated code:

- Expand the **Optimization** Group, and select **Signals and Parameters**, and make sure that **Default parameter behavior** is set to **Tunable**.
- Press **OK** in the **Configuration Parameters**.
- Save the model.

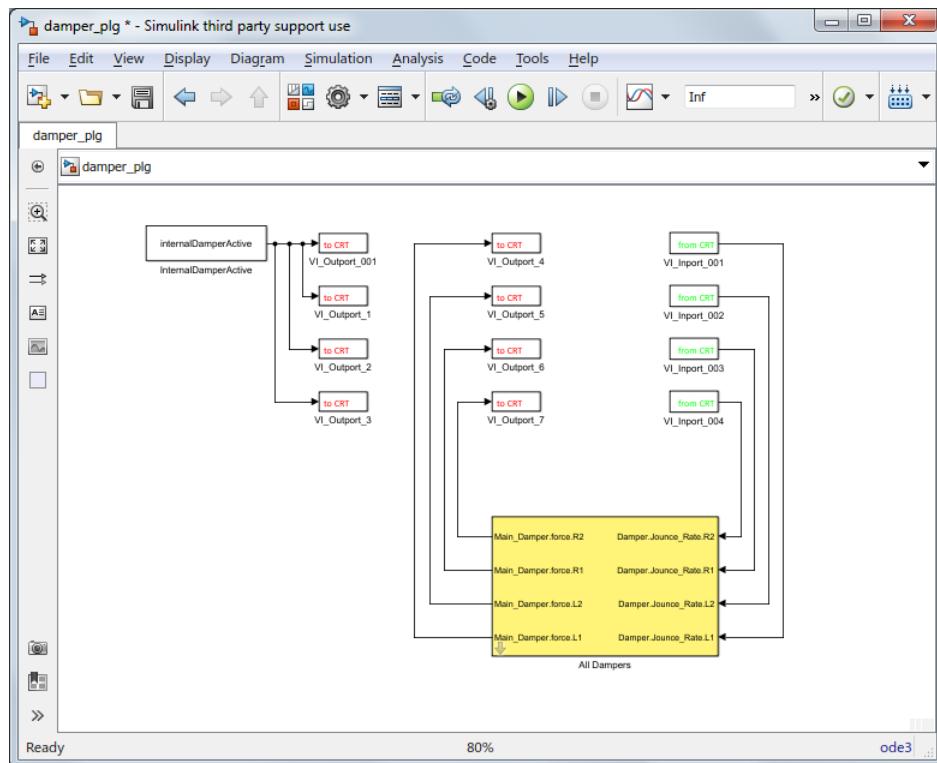
A parameter called "vicrt_inputfile" referencing a valid *_send_svm.xml vehicle model file must be defined in the MATLAB Workspace.

Model I/O (Obsolete)

This section is obsolete but still functional. Skip to the next section to continue the tutorial.

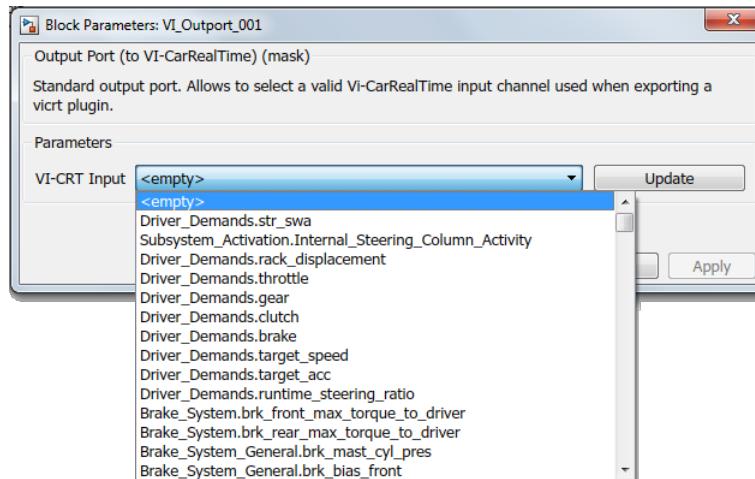
In order to export a model that is properly connected to VI-CarRealTime Solver, we need to replace the connections between the VI-CarRealTime S-Function and the rest of the model with appropriate Imports/Outports blocks.

In the damper_plg model remove the VI-CarRealTime model interface and replace each input to VI-CarRealTime with a **toCRT** block from the VI-CarRealTime Library, and each output from VI-CarRealTime with a **fromCRT** block from the VI-CarRealTime Library. Once this is done the model should look as follows:

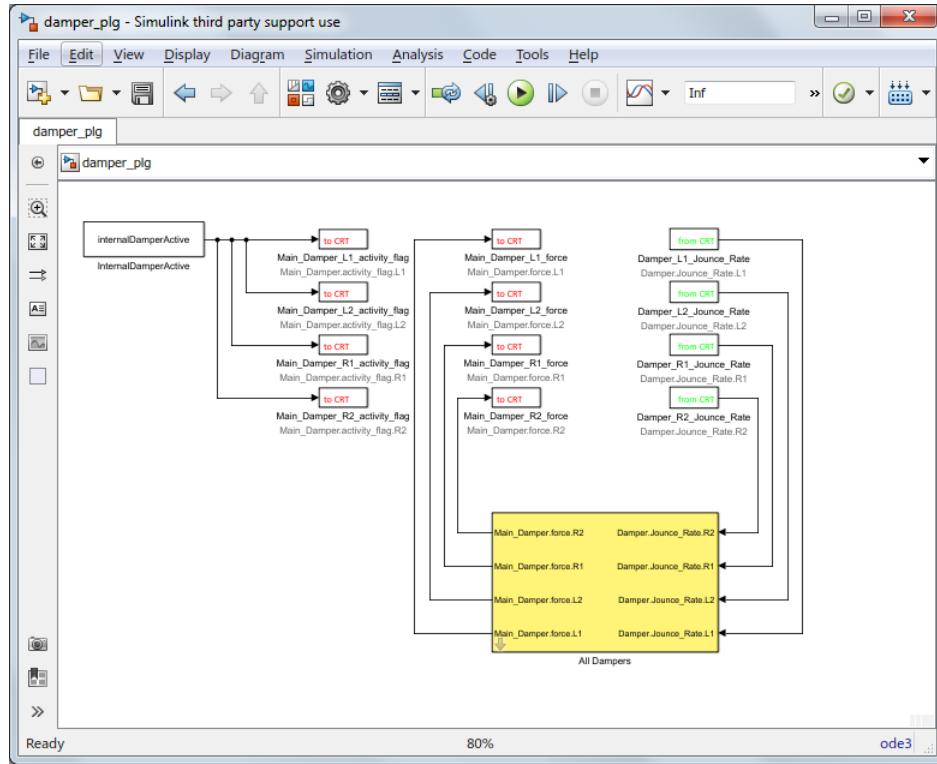


Each of the connection has been replaced with a corresponding input/output port. In order to configure the connection properly we need to provide to the model the name of a valid VI-CarRealTime `<event_name>_send_svm.xml` file: to do this, set the MATLAB variable **vicrt_inputfile** to the full path name of the event file.

Once the event file name is stored in **vicrt_inputfile** variable, open one of the input/outport blocks mask, and press the update button. The lists of VI-CRT Input and VI-CRT Output values will be automatically updated according to the selected model:



Select the desired input/output channels. The name of each block will change accordingly to the selected channel name:



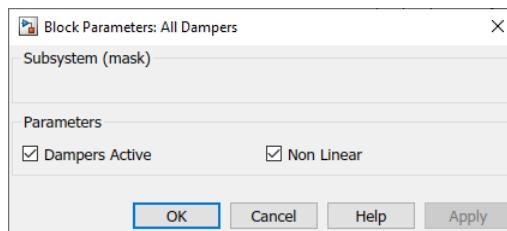
Model Parameters

Since we specified in the Configuration Parameters dialog that the model parameters are Tunable, the exported code will contain all the parameters exposed in the blocks used to build the model.

In addition to those parameters the model also contains two Global Simulink Parameters:

- **internalDamperActive**: when this flag is set to 1 the damper formulation embedded in the VI-CarRealTime model is active and the external damper are unused. When it is set to 0 it is the opposite way around.
- **useSpline**: the parameter switches the external damper formulation from linear to spline.

When using the damper model in Simulink these parameters can be specified through the mask of the All Damper Subsystem:



When exporting the model using Simulink Coder, these Global Parameters will appear with their name in the VI-CarRealTime [auxiliary subsystem](#).

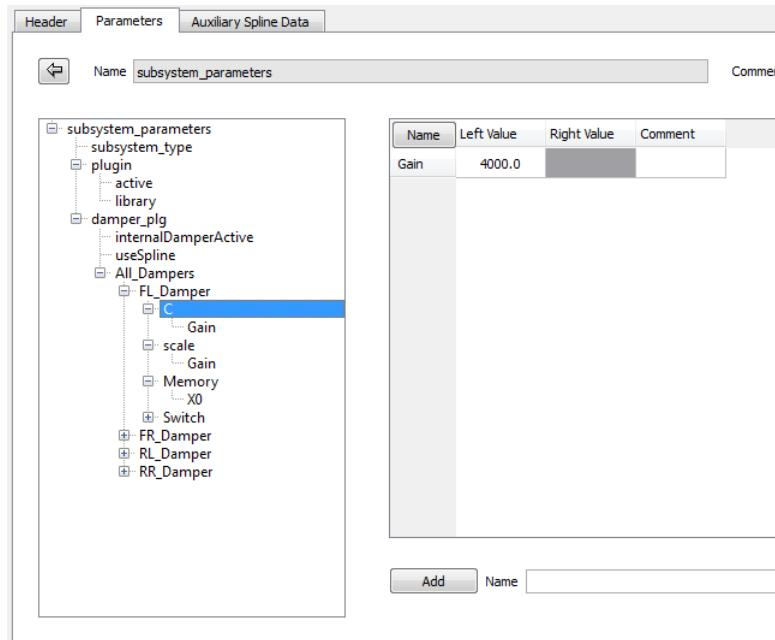
Model Build/Export

The model can now be built pressing the **Build Model** button (Ctrl+B).

The Simulink Diagnostic Viewer will show the status of the execution, and will report errors if they occur. If the code exports, the generation of the VI-CarRealTime Solver Plugin interface code, generation of the auxiliary subsystem and creation of the dynamic library are successful, the Diagnostic Viewer will show the message "Build process Completed successfully".

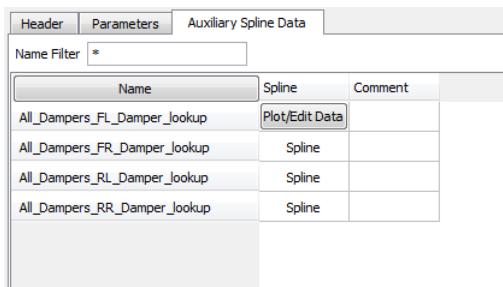
All the outputs of the build process are stored in a subdirectory of the MATLAB working directory, called <model_name>_crt_rtw. This directory contains the intermediate files generated during the build process, and the final result, the dynamic library with the compiled plugin and the auxiliary subsystem file.

Once imported in VI-CarRealTime, the auto generated auxiliary subsystem will show the following content:



Model parameters are stored in a sub-tree with the same name of the model. Global Simulink parameters are exposed directly under the model, while the other parameters follows the same hierarchy of the original Simulink Model.

An exception is represented by Simulink lookup table blocks that are transformed in Auxiliary splines in the subsystem:



Additional information on how to use a VI-CarRealTime Solver Plugin in simulation are available in [VI-CarRealTime Customization Tutorials](#)

Run Investigation with a Solver Plugin

The tutorial shows the user how to integrate an Auxiliary Subsystem and a solver plug-in generated using Simulink Coder / Real time workshop in VI-CarRealTime and then set up and submit an investigation changing the parameters from the Auxiliary Subsystem.

In particular:

1. we will start from a Simulink model implementing a rear limited slip differential;
2. we'll export the model as a plug-in dll plus auxiliary subsystem with `Tunable` parameters, so that we can change the values of the Simulink model blocks directly from VI-CarRealTime;
3. we'll integrate the exported auxiliary subsystem in the RaceCar model;
4. we'll submit an investigation using as factors the LSD parameters from the auxiliary subsystem.

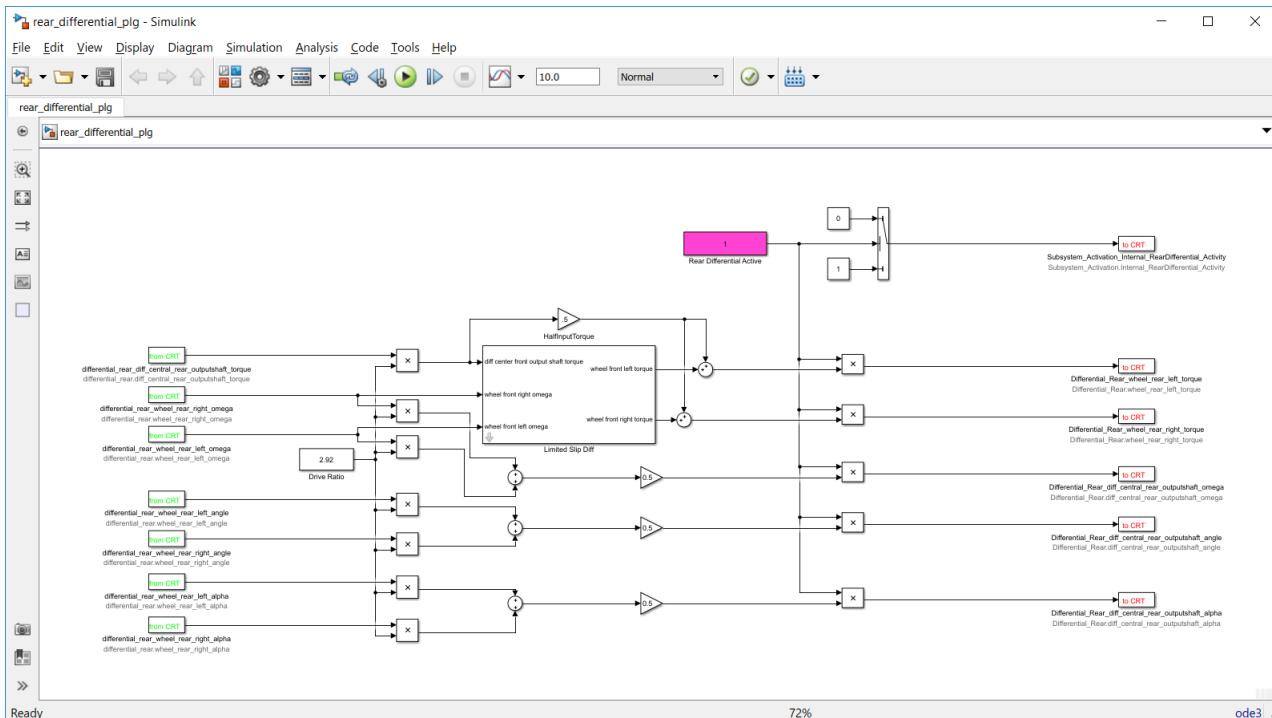
The tutorial consists of the following sections:

- [Build/Export Simulink Model](#)
- [Add Auxiliary Subsystem in VI-CarRealTime](#)
- [Set Up the Event](#)
- [Set Up the Investigation](#)
- [Run the Investigation](#)
- [Review Results](#)

Build/Export Simulink Model

This tutorial assumes that the user already knows the VI-CarRealTime MATLAB/Simulink interface, and knows how to generate an event file to be executed in MATLAB. More detail can be found in the tutorial dedicated to [Code Simulation](#).

In MATLAB open the following Simulink model: \$VI-CarRealTime\$Installationdir\acarr\examples\Simulink\rear_differential_plg.mdl



The Simulink model represents a simplified version of a rear LSD differential.

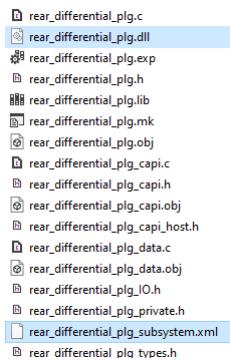
The [Model I/O](#) and the Configuration Parameters have already been set-up for export the solver plug-in for VI-CarRealTime.

Anyway it's recommended to double check that all the Configuration Parameters are compliant with the user layout. For all the details please refer to [Model Configuration](#) topic.

Once the model configuration has been verified, the model can now be built pressing the **Build Model** button (Ctrl+B).

The Simulink Diagnostic Viewer will show the status of the execution, and will report errors if they occur. If the code exports, the generation of the VI-CarRealTime Solver Plugin interface code, generation of the auxiliary subsystem and creation of the dynamic library are successful, the Diagnostic Viewer will show the message "Build process Completed successfully".

All the outputs of the build process are stored in a subdirectory of the MATLAB working directory, called <model_name>_crt_rtw. This directory contains the intermediate files generated during the build process, and the final result, the **dynamic library** with the compiled plugin and the **auxiliary subsystem** file.

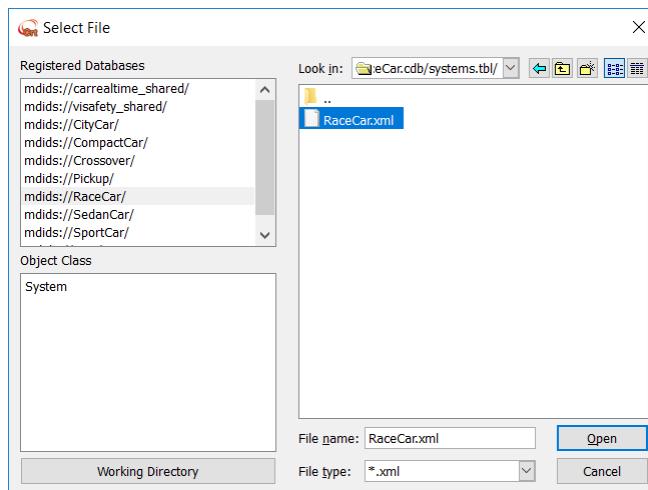


Copy **rear_differential_plg.dll** file and paste it in MATLAB working directory.

Add Auxiliary Subsystem in VI-CarRealTime

Open a new VI-CarRealTime session and set as working directory the MATLAB working directory.

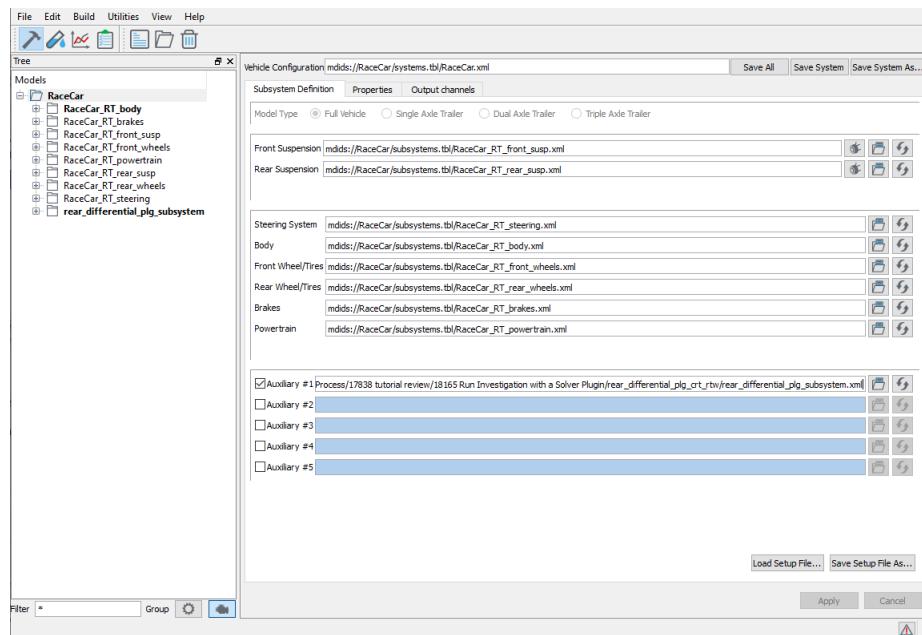
In **Build mode**, go to Build -> Load Model, and load the **RaceCar.xml** vehicle model from the *RaceCar* database.



Now we will add the auxiliary subsystem to RaceCar model. To do it:

VI-CarRealTime

- Check **Auxiliary #1** toggle button and add the auxiliary subsystem created in the previous section;
- The subsystem (**rear_differential_plg_subsystem.xml**) is stored in **rear_differential_plg_crt_rtw** folder;
- press **Apply** button.

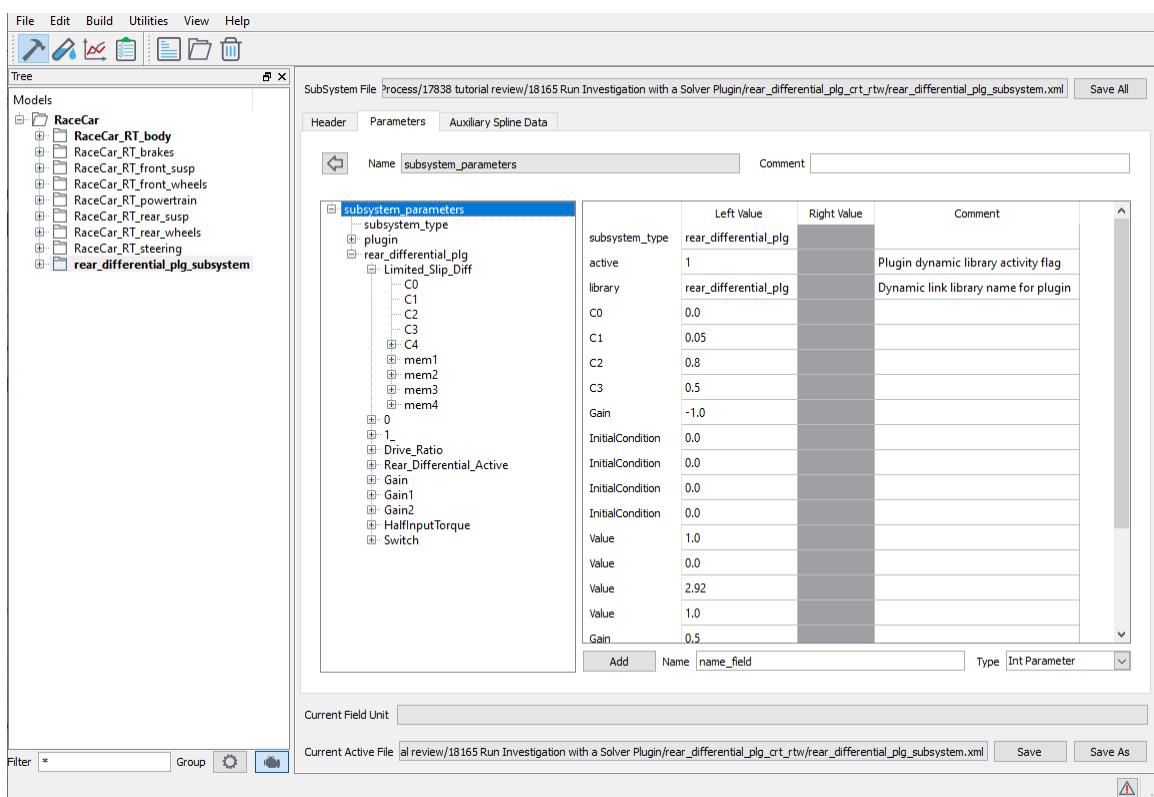


The current version of the RaceCar model will have its rear differential modeled externally in the compiled dll library referenced through the auxiliary subsystem.

In particular, its internal (powertrain subsystem) rear differential will be switched off due to the fact that the following Input has been set to 0 in Simulink:



Open the auxiliary subsystem, and expand the tree to see the parameters that can be tuned:



In particular, for the purpose of the tutorial, will change the values of the coefficient C1 and C2 of the differential. For all the details about LSD differential implementation, please refer to [Differentials](#) topic.

Note: the dynamic link library is referenced in the Auxiliary Subsystem with its name (`rear_differential_plg`), so it's mandatory that `rear_differential_plg.dll` is stored in VI-CarRealTime working directory.

Set Up the Event

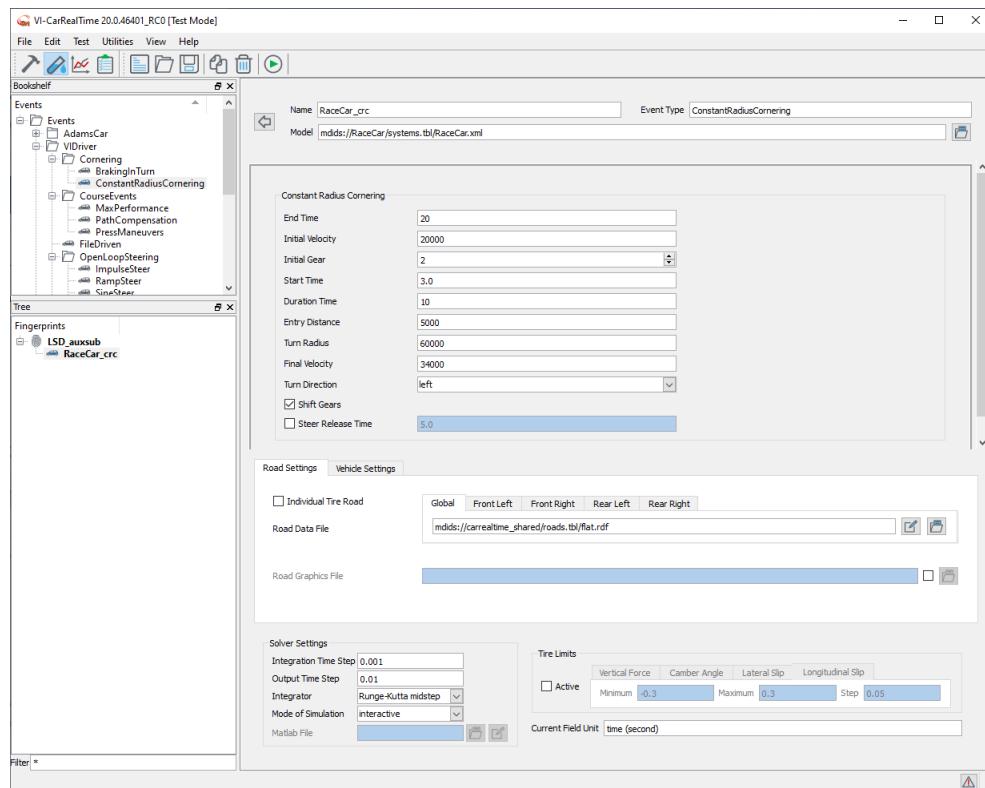
The next step consists in setting up an event which will be used as reference event for the investigation analysis that we're going to run.

Switch to **Test Mode** add a **ConstantRadiusCornering** event.

Set the following parameters for the event:

- change the fingerprint name to **LSD_auxsub**;
- set **End Time** to **20.0 s**;
- set **Initial Velocity** to **20000** (20 m/s);
- set **Start Time** to **3 s**;
- set **Duration Time** to **10.0 s**;
- set **Entry Distance** to **5000** (5 m);
- set **Turn Radius** to **60000** (60 m);
- set **Final Velocity** to **34000** (34 m/s);

VI-CarRealTime



Save the current fingerprint.

Note: it's not necessary to run the event. We'll use such event in the next section to create the investigation.

Set Up the Investigation

In this section, we'll set up the investigation.

The investigation will use the *ConstantRadiusCornering* event as reference event and we'll vary the C1 and C2 coefficients of the auxiliary subsystem to explore how the vehicle variants behaves in such event.

Switch to **Investigation Mode**

To create a New Investigation:

- right-click on **RaceCar_crc** event and select **Add Event To New Investigation**;

The investigation will be added in the Tree panel:



Now we have to set the investigation parameters. In particular:

- [Factors](#)
- [Responses](#)
- [Investigation Parameters](#)

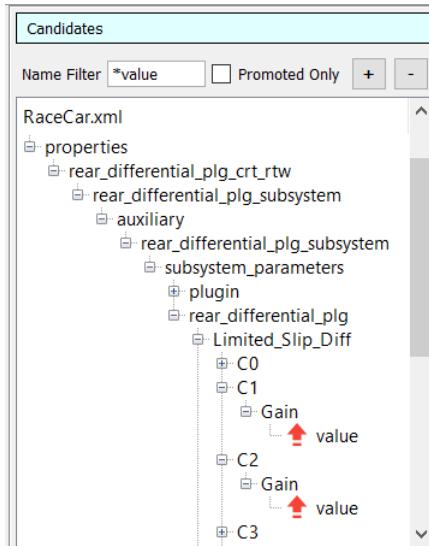
Factors

Since we want to investigate the effect on vehicle results when changing C1 and C2 parameters of the LSD differential referenced in the auxiliary subsystem, we need to promote the following two candidates to factors:

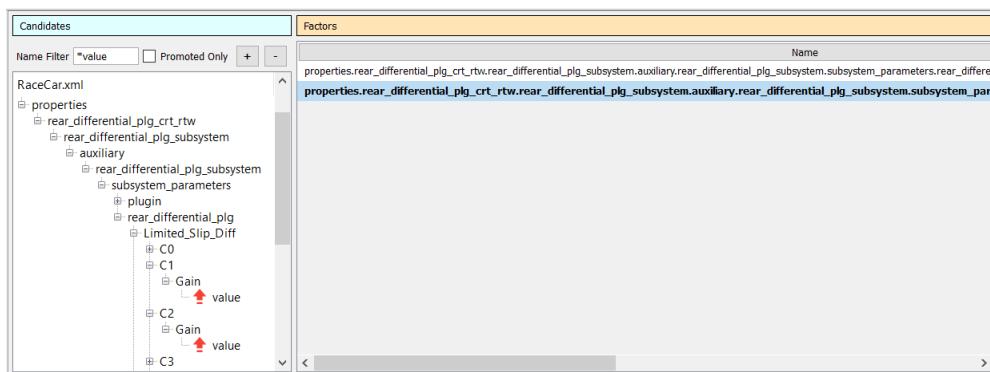
1. `properties.rear_differential_plg_crt_rtw.rear_differential_plg_subsystem.auxiliary.rear_differential_plg_subsystem.subsystem_parameters.rear_differential_plg.Limited_Slip_Diff.C1.Gain.value`
2. `properties.rear_differential_plg_crt_rtw.rear_differential_plg_subsystem.auxiliary.rear_differential_plg_subsystem.subsystem_parameters.rear_differential_plg.Limited_Slip_Diff.C2.Gain.value`

To promote a candidate to a factor:

- expand the candidates tree to find the targets (enter `value` in **Name Filter** field to narrow the search)



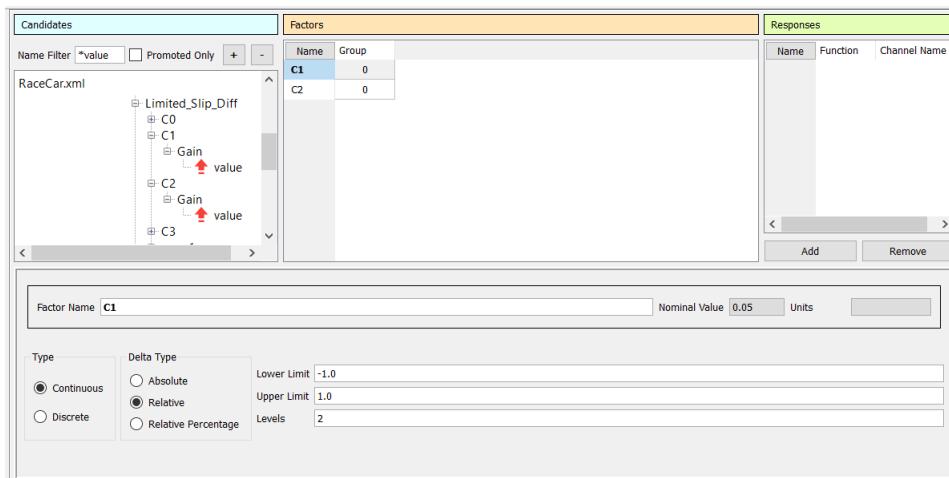
- right-click on the candidate and select **Promote to Factor** (otherwise simply double click on it)



The name of the factor can be very long since its name is composed by the complete tree structure where it's nested; it's possible to rename each factor assigning a custom name.

To do it:

- double click on the first factor;
- edit the name of the **Factor Name** field changing it to **C1**;
- press **Enter** to apply the change;
- repeat the same operation for the second factor entering **C2** as **Factor Name**.



Now we need to set the range of variation of the two factors for the investigation.

To do it:

- left click on **C1**;
- set **Type** to **Continuous**;
- set **Delta Type** to **Absolute**;
- set **Lower Limit** to **0.0**;
- set **Upper Limit** to **0.05**;
- set **Levels** to **5**;

Factor Name	Nominal Value	Units
C1	0.05	

Type	Delta Type	Lower Limit	Upper Limit	Levels
<input checked="" type="radio"/> Continuous	<input checked="" type="radio"/> Absolute	0.0	0.05	5
<input type="radio"/> Discrete	<input type="radio"/> Relative			
	<input type="radio"/> Relative Percentage			

- left click on **C2**;
- set **Type** to **Continuous**;
- set **Delta Type** to **Absolute**;
- set **Lower Limit** to **0.1**;
- set **Upper Limit** to **1.0**;
- set **Levels** to **3**;

Factor Name	Nominal Value	Units
C2	0.8	

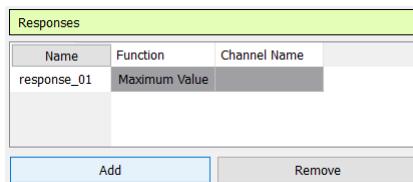
Type	Delta Type	Lower Limit	Upper Limit	Levels
<input checked="" type="radio"/> Continuous	<input checked="" type="radio"/> Absolute	0.1	1.0	3
<input type="radio"/> Discrete	<input type="radio"/> Relative			
	<input type="radio"/> Relative Percentage			

Responses

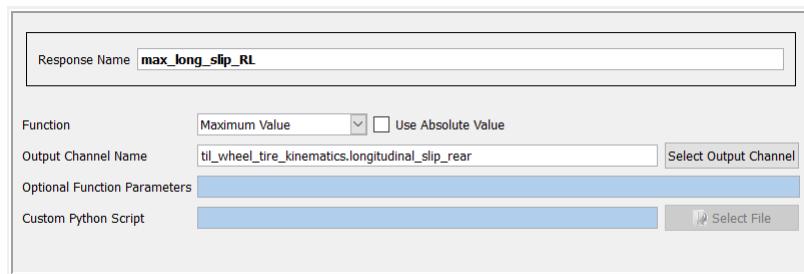
Our target is to monitor the longitudinal slip of the internal rear wheel (rear left tire), so we need to create one response which monitors such output and computes its maximum value.

To create a new Response:

- click **Add** button in responses section



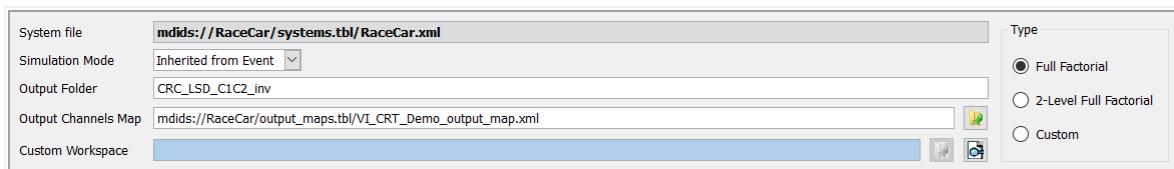
- left click on **response_01** to open the response section panel;
- in **Response Name** field, change the response name to **max_long_slip_RL**;
- set **Function** option menu to **Maximum Value**;
- click on **Select Output Channel** button and select **til_wheel_tire_kinematics.longitudinal_slip_rear** output.



Investigation Parameters

The bottom part of the investigation section defines general investigation parameters. In particular for our purpose we set:

- **Output Folder** name to **CRC_LSD_C1C2_inv**;
- **Type** to **Full-Factorial**.



The investigation is now set-up. 15 combinations of factors will be run as can be seen in the HTML Investigation Design Space report.

Investigation Name	investigation_1
System File	mdids://RaceCar/systems.tbl/RaceCar.xml
Strategy	Full Factorial
Output Folder	CRC_LSD_C1C2_inv

Selected Event: LSD_auxsub.RaceCar_crc

Identifier	C1	C2	Success	max_long_slip_RL
v0001	0.00000	0.100000	?	N/A
v0002	0.00000	0.550000	?	N/A
v0003	0.00000	1.00000	?	N/A
v0004	0.0125000	0.100000	?	N/A
v0005	0.0125000	0.550000	?	N/A
v0006	0.0125000	1.00000	?	N/A
v0007	0.0250000	0.100000	?	N/A
v0008	0.0250000	0.550000	?	N/A
v0009	0.0250000	1.00000	?	N/A
v0010	0.0375000	0.100000	?	N/A
v0011	0.0375000	0.550000	?	N/A
v0012	0.0375000	1.00000	?	N/A
v0013	0.0500000	0.100000	?	N/A
v0014	0.0500000	0.550000	?	N/A
v0015	0.0500000	1.00000	?	N/A

Note: in the report Success column values have ? and the related response is set to N/A since the investigation hasn't been run yet.

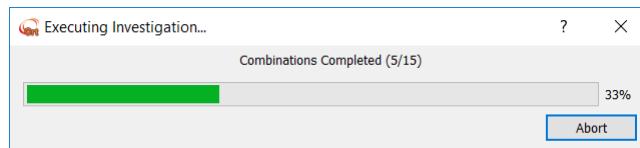
Click to save the investigation xml file in the working directory.

Run the Investigation

Click button to run the investigation.

All the combinations will be run in the investigation folder (CRC_LSD_C1C2_inv).

In VI-CarRealTime GUI, a progress dialog will be displayed to show the combinations completed and the percentage status:



Note: once the investigation xml file has been saved, it's also possible to run the investigation in batch mode. Please refer to [Run the Investigation in batch mode](#) for the details.

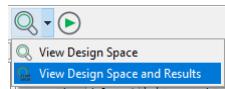
Wait until all the investigation analyses have been run.

Review Results

When the investigation is completed, it's possible to review and analyze the results. In particular, in VI-CarRealTime working directory the HTML report can be found.

To open it:

- double click on *CRC_LSD_C1C2_inv_summary.html* file in VI-CarRealTime working directory
- or
- click on lens stack button arrow and select **View Design Space and Results**.



The report shows the response values for all the factor combinations:

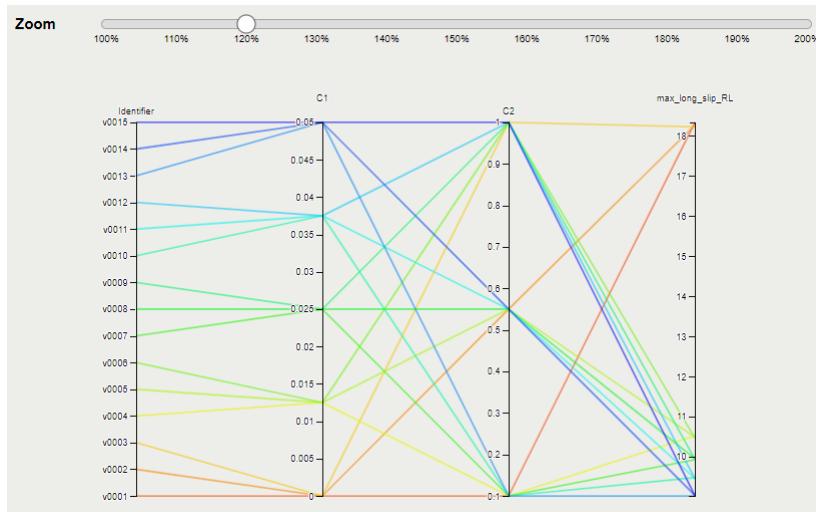
Investigation Summary Report

Investigation Name	investigation_1
System File	mdids://RaceCar/systems.tbl/RaceCar.xml
Strategy	Full Factorial
Output Folder	CRC_LSD_C1C2_inv

Selected Event: LSD_auxsub.RaceCar_crc

Identifier	C1	C2	Success	max_long_slip_RL
v0001	0.00000	0.100000	✓	18.3389
v0002	0.00000	0.550000	✓	18.2697
v0003	0.00000	1.00000	✓	18.2270
v0004	0.0125000	0.100000	✓	10.5242
v0005	0.0125000	0.550000	✓	10.4875
v0006	0.0125000	1.00000	✓	10.4656
v0007	0.0250000	0.100000	✓	9.92928
v0008	0.0250000	0.550000	✓	9.95103
v0009	0.0250000	1.00000	✓	9.93412
v0010	0.0375000	0.100000	✓	9.48064
v0011	0.0375000	0.550000	✓	9.45855
v0012	0.0375000	1.00000	✓	9.44528
v0013	0.0500000	0.100000	✓	9.02062
v0014	0.0500000	0.550000	✓	9.03502
v0015	0.0500000	1.00000	✓	9.02426

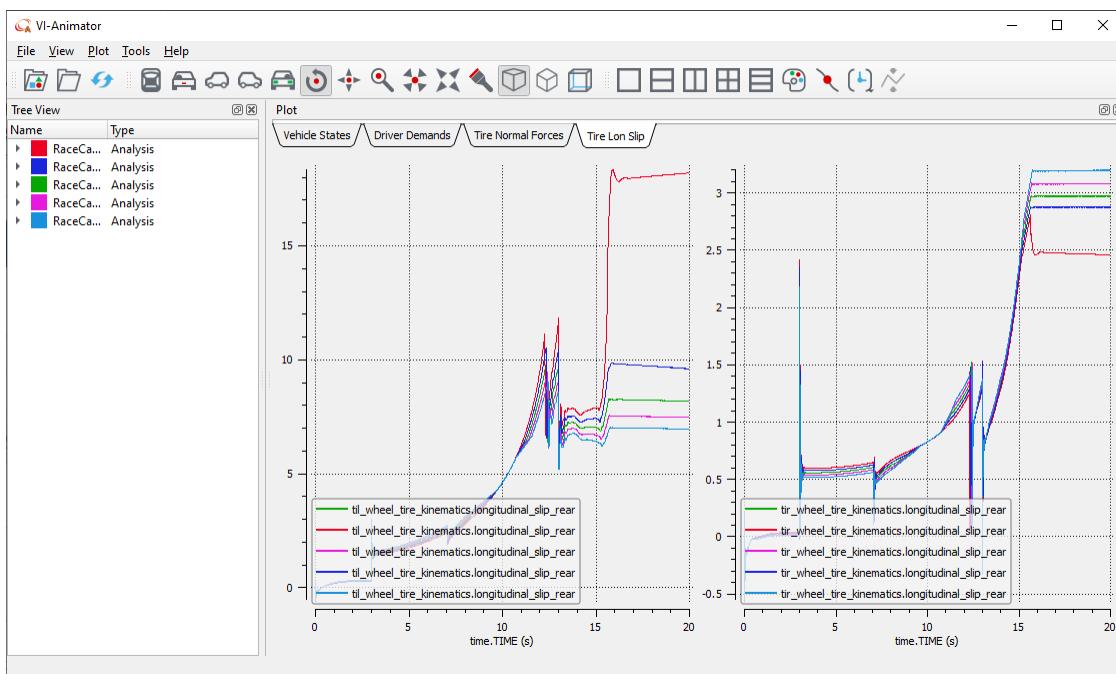
Moreover, the last part of the report shows the plot of the two factors together with the response:



So, the combination v0013 provides the lowest value of the maximum longitudinal slip of the rear left tire (**9.02062**):

Identifier	C1	C2	Success	max_long_slip_RL
v0001	0.00000	0.100000	✓	18.3389
v0002	0.00000	0.550000	✓	18.2697
v0003	0.00000	1.00000	✓	18.2270
v0004	0.0125000	0.100000	✓	10.5242
v0005	0.0125000	0.550000	✓	10.4875
v0006	0.0125000	1.00000	✓	10.4656
v0007	0.0250000	0.100000	✓	9.92928
v0008	0.0250000	0.550000	✓	9.95103
v0009	0.0250000	1.00000	✓	9.93412
v0010	0.0375000	0.100000	✓	9.48064
v0011	0.0375000	0.550000	✓	9.45855
v0012	0.0375000	1.00000	✓	9.44528
v0013	0.0500000	0.100000	✓	9.02062
v0014	0.0500000	0.550000	✓	9.03502
v0015	0.0500000	1.00000	✓	9.02426

Additional considerations can be done by loading some of the simulation results file of the investigation (stored in `CRC_LSD_C1C2_inv` folder) in VI-Animator:



API Toolkit

The following chapter comprises a series of tutorials which will teach a user how to approach an optimization chain of VI-CarRealTime and VI-SuspensionGen models, using the MATLAB [API Tool](#).

The two first exercises contained in this chapter show the user how to properly use the MATLAB [API Tool](#) to change parameters and data in VI-CarRealTime and VI-SuspensionGen models, how to call the appropriate solvers using MATLAB, and how to manage and plot corresponding simulation results.

The third tutorial exercise, on the other hand, is a more advanced exercise whose purpose is to give the user an understanding of how to modify the layout of a suspension, in order to reach a particular pre-defined target value. The three subsections of this MATLAB [API Tool](#) are as follows:

1. [Manage VI-CarRealTime Model](#)

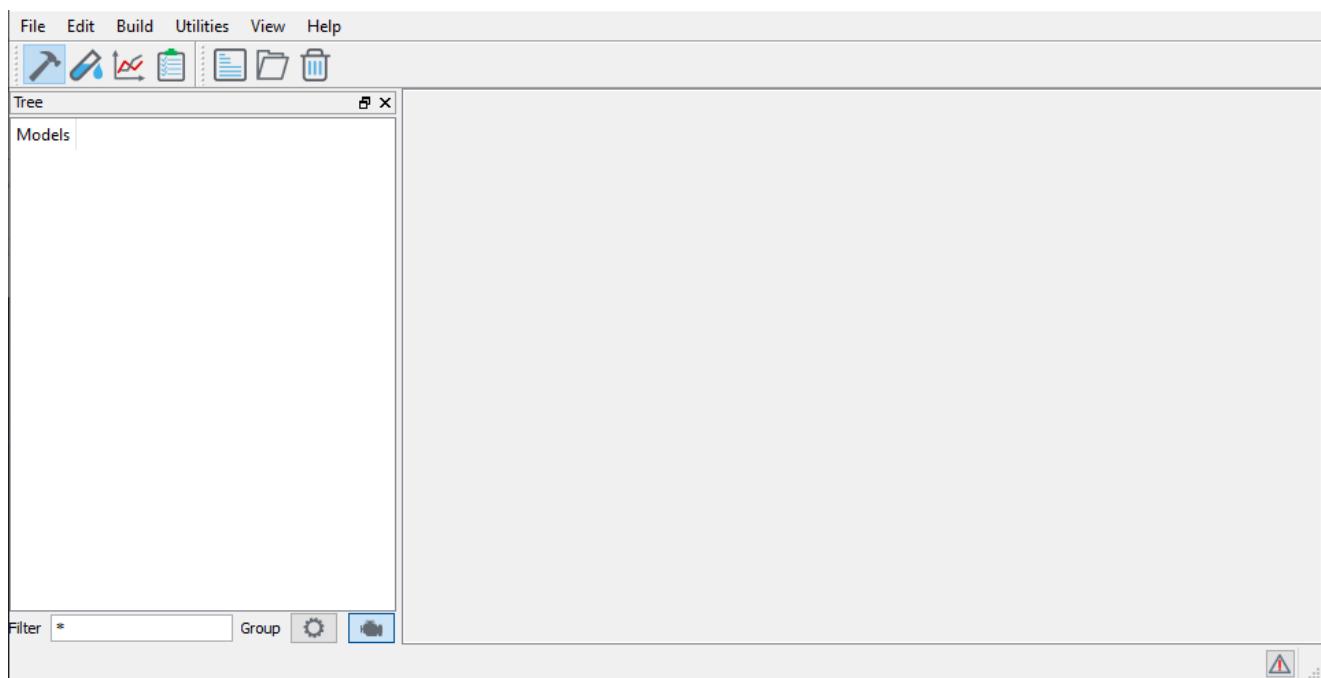
2. [Manage VI-SuspensionGen Model](#)
3. [Suspension Optimization](#)

Note: the MATLAB script files used to run the following tutorials are stored in the VI-CarRealTime Installation directory under `\acarrt\matlab\opt_lib\example` folder.

Manage a VI-CarRealTime Model

The following tutorial gives an overview of the basic VI-CarRealTime MATLAB library capabilities including model management, analysis, and post-processing of results. To elaborate, this tutorial will investigate how the frame center of gravity height influences the body roll angle during a step steer analysis.

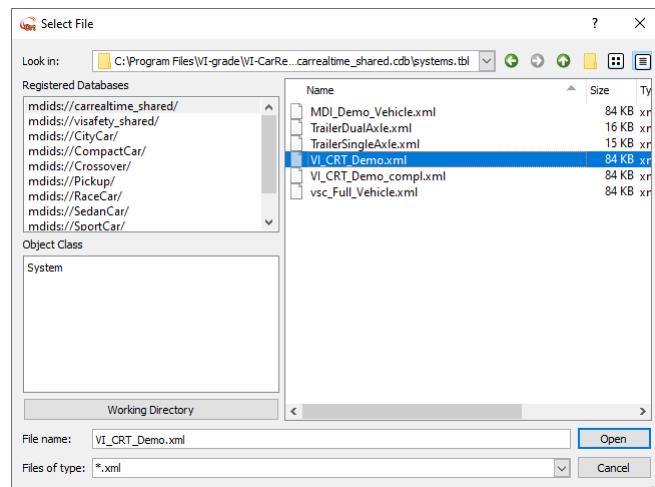
Start a new VI-CarRealTime session using the Windows Start Menu or by typing `vicrt20` in a DOS (cmd) shell.



VI-CarRealTime will open in [Build Mode](#) (by default), as shown by the highlighted hammer icon in the toolbar; if another mode is highlighted instead, please use the toolbar to switch to Build Mode.

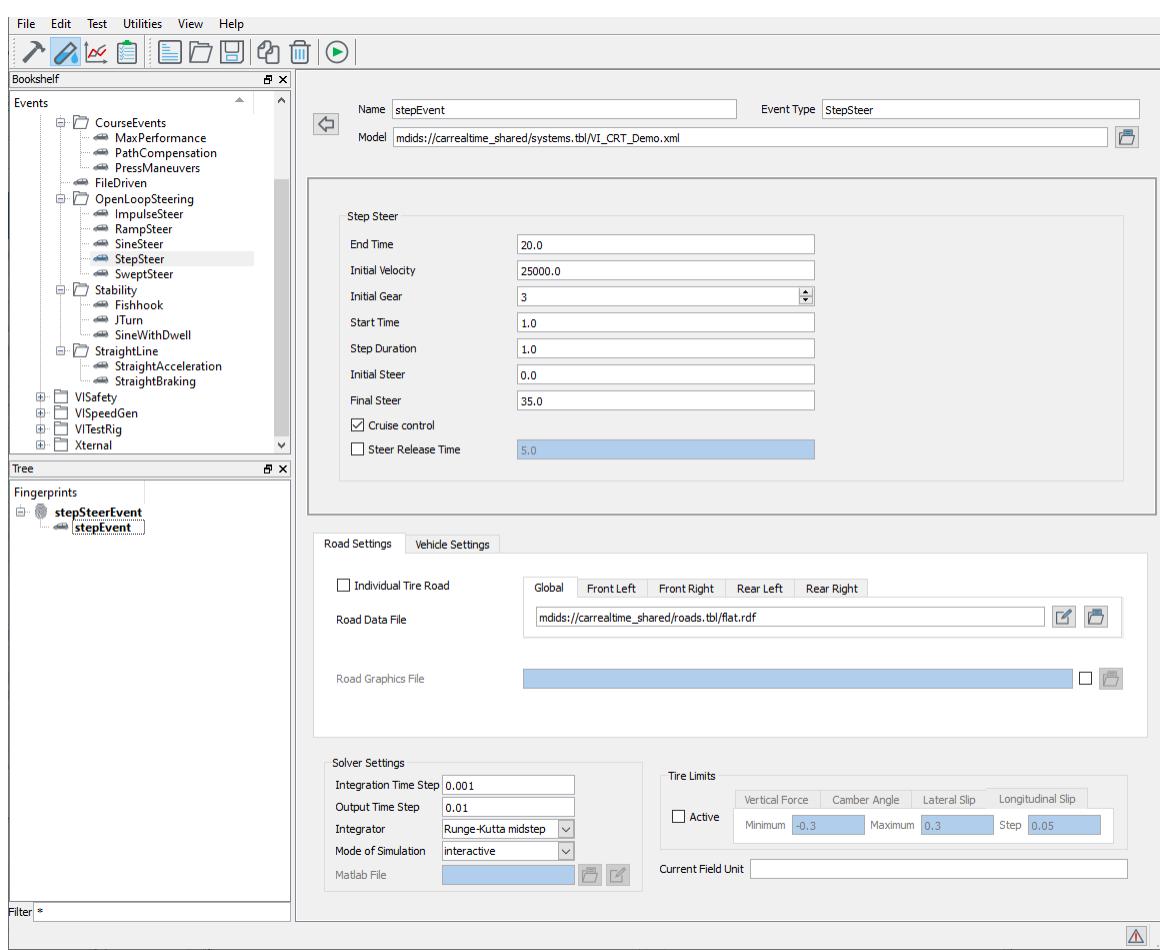
In order to create the necessary event, we need to load a vehicle model into our VI-CarRealTime session. Since the analysis will be performed in a MATLAB environment (as opposed to within the VI-CarRealTime environment), we may choose a generic model to create a fingerprint template; MATLAB API will automatically update the fingerprint with relevant parameters before running each analysis.

Open the usual `VI_CRT_Demo.xml` by clicking on the icon (alternately, in the VI-CarRealTime menu, go to Build -> Load Model...) and then locate the vehicle model in the `carrealtime_shared` database.

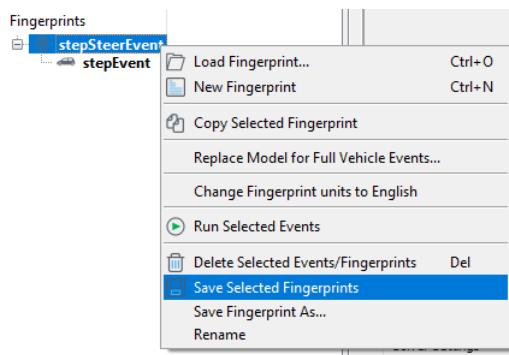


Switch to [Test Mode](#), using the button, and in the left hand tree view, add a [Step Steer](#) event (found in Events -> VIDriver -> OpenLoopSteering) to a new fingerprint. Next, fill in the event parameters as shown below:

- **End Time** = 20 s
- **Initial velocity** = 25000 mm/s
- **Initial gear** = 3
- **Start Time** = 1 sec
- **Step Duration** = 1 sec
- **Initial Steer** = 0
- **Final Steer** = 35 degrees
- Check the **Cruise control** checkbox



Rename the new event **stepEvent**, and the new fingerprint **stepSteerEvent**, and then save the fingerprint to your current VI-CarRealTime working directory as shown below:

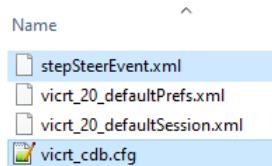


In your working directory, you will now find a file called [vicrt_cdb.cfg](#), in which the databases registered in the current VI-CarRealTime session are stored. Open this file (using a text editor program such as Notepad, or Notepad++) and verify that the following lines of code are present:

```
!-----!
! ***** VI-CarRealTime Database Configuration File ****!
!-----!
!           Database name      Path of Database
!-----!
DATABASE  carrealtime_shared  C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/CARREA~1.CDB
```

Afterwards, browse your working directory once again, where you should find the following two files:

- `stepSteerEvent.xml` : the xml file that deals with embedding a fingerprint
- `vicrt_cdb.cfg` : the same configuration file discussed in the previous paragraph, which deals with embedding databases



From here on out, we will switch to the MATLAB environment; please open MATLAB from the Windows Start Menu.

In your MATLAB session, set your working directory to the same working directory used for VI-CarRealTime, and then create a new script file. Inside this new script, copy and paste the following lines of code:

```
addpath_vicrt_20; %%add library
```

The previous command includes the VI-CarRealTime library to be added to MATLAB search path. With the commands below, we will prepare necessary MATLAB string variables for our input file.

```
cfgFile = 'vicrt_cdb.cfg';
systemFile = 'mdids://carrealtime_shared/systems.tbl/VI_CRT_Demo.xml';
baseFingerprint = 'stepSteerEvent.xml';
```

The next step involves the creation of a MATLAB structure variable to collect relevant vehicle data.

```
modelDirFiles = 'tmpFolder';
systemStruct = generateSystemStruct(systemFile,createCfg(cfgFile),modelDirFiles);
```

Six different frame center of gravity heights will be analyzed in the range between 400 mm and 500 mm. A plot with the different time histories for body roll angle, with respect to a respective road, will be generated. The code below will perform these two aforementioned tasks.

```
cogZArray =[400:20:500];

figure;
hold on;
title('Body Roll Angle');
Legend=cell(length(cogZArray),1);
```

Note: the script we are working on is embedded in the "vicrt_installation_dir\acar\matlab\opt_lib\example\tutorial\Vicrt" folder.

The next step consists in creating the script's main loop. In the first part of the loop, the height of the center of gravity is updated. In order to do so, a cell array of variables collecting the list of strings required to resolve the corresponding xml tree will hence be created.

```
<Subsystem active="true" name="RT_VT_CRT_Demo_body" userDefined="false">
  <CRTSprungMass active="true" name="body" cgHeight="500" axialStiffness="1"
    413.404" front_ixx="79304700" front_ixy="904901" front_ixz="-15032600" fro
    /VI_CRT_Demo_RT.cmd" ixx="158609000" ixy="1809800" ixz="-30065100" iyy="77
    lateralStiffnessActiveFlag="false" mass="1147.66" rear_cgX="1566.29" rear_
    419919000" rear_mass="573.829" torsionalStiffness="1" torsionalStiffnessAc
    verticalStiffnessActiveFlag="false" wheelbase="2920"/>
```

```

for i=1:length(cogZArray)
    keys{1,1}='CRTSprungMass';
    keys{2,1}='body';

    attributes= cell(1);
    attributes{1}='cgHeight';

    values=cell(1);
    values{1}=cogZArray(i);

    systemStruct = bodySetValue(systemStruct,keys,attributes,values);

```

At this point, the step event will be performed with the modified model, using the code below.

```

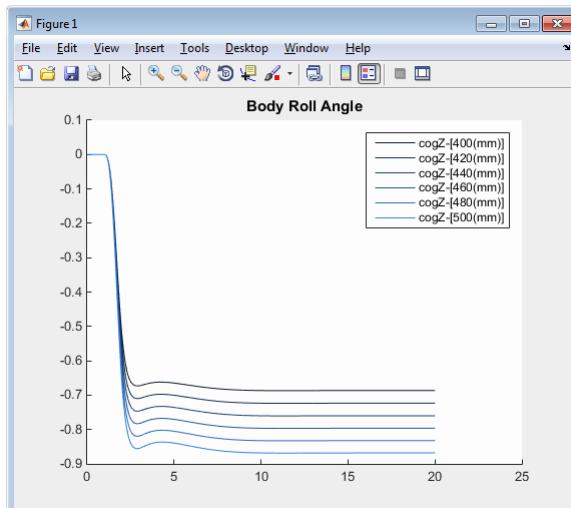
%%%%%%%%%%%%%
% update fingerprint and run %
%%%%%%%%%%%%%
workingDir ='crtWork';
[successFlag, outputNameList]=runViCrt(baseFingerprint,systemStruct,'',workingDir);

out_prefix = sprintf('%s-[%g(mm)]', 'cogZ', cogZArray(i));
Legend{i}=out_prefix;
% load and plot cab roll angle
output=sprintf('%s.mat',outputNameList{1});
out=load(output);
plot(out.time,out.chassis_displacements_roll*180/pi,'color',[0.2*i/
length(cogZArray)), ...
0.5*i/(length(cogZArray)) ,0.9*i/(length(cogZArray))]);
clear out;
hold on;
end
legend(Legend)

```

Finally, save and run the script.

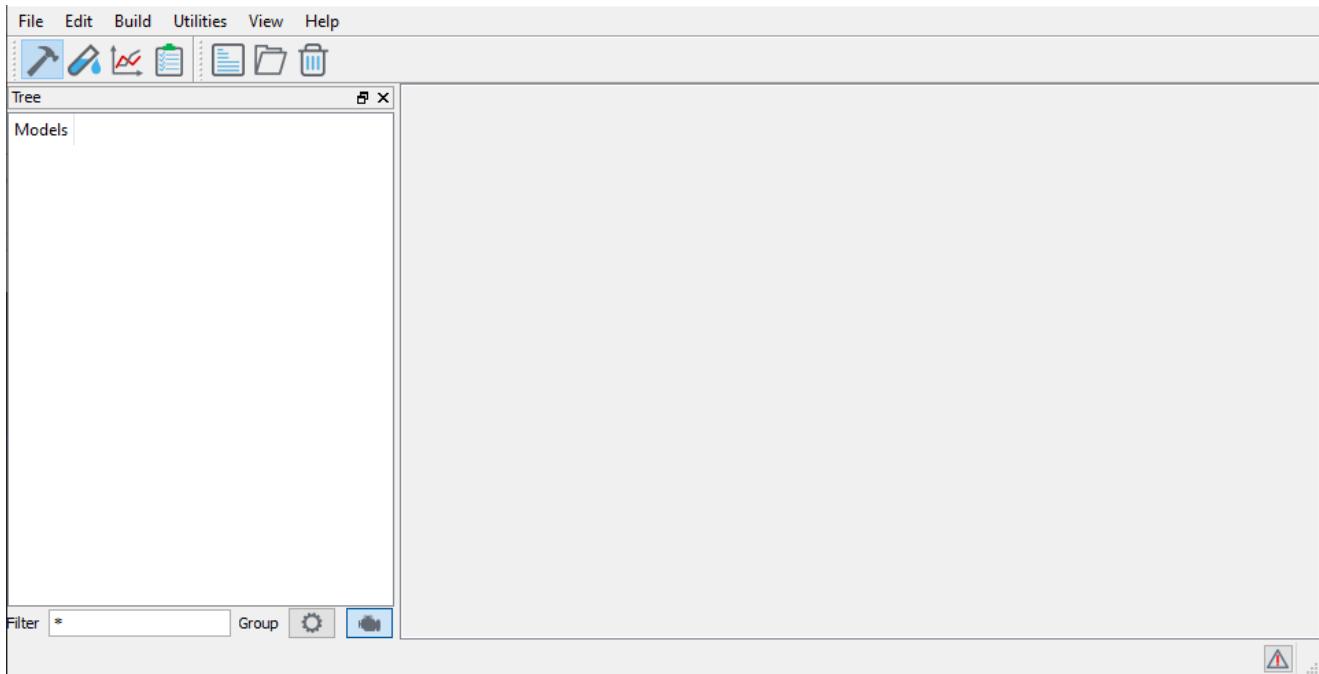
The body roll angle for each individual simulation appears in the MATLAB plot shown below, which will be summoned by running the script you just wrote.



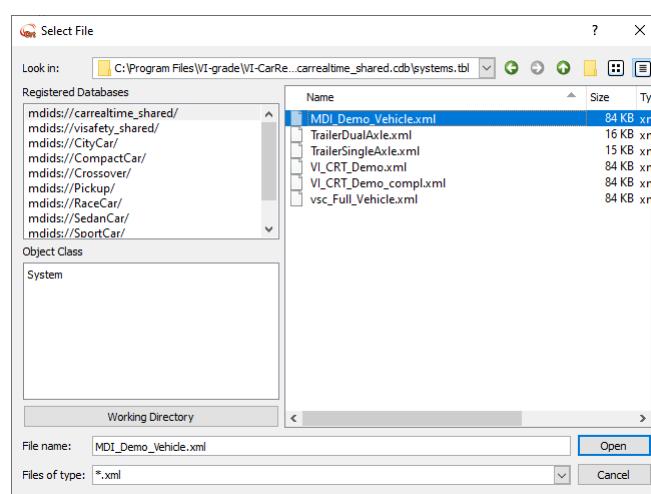
Manage a VI-SuspensionGen Model

This particular tutorial teaches a user how to deal with VI-SuspensionGen suspension files using the [Matlab API Toolkit](#). Specifically, certain key parameters of an sgs file will be changed in MATLAB; namely, a suspension file will be modified and used to run under the VI-SuspensionGen solver. The output of this simulation will be loaded into MATLAB and have its relevant values plotted.

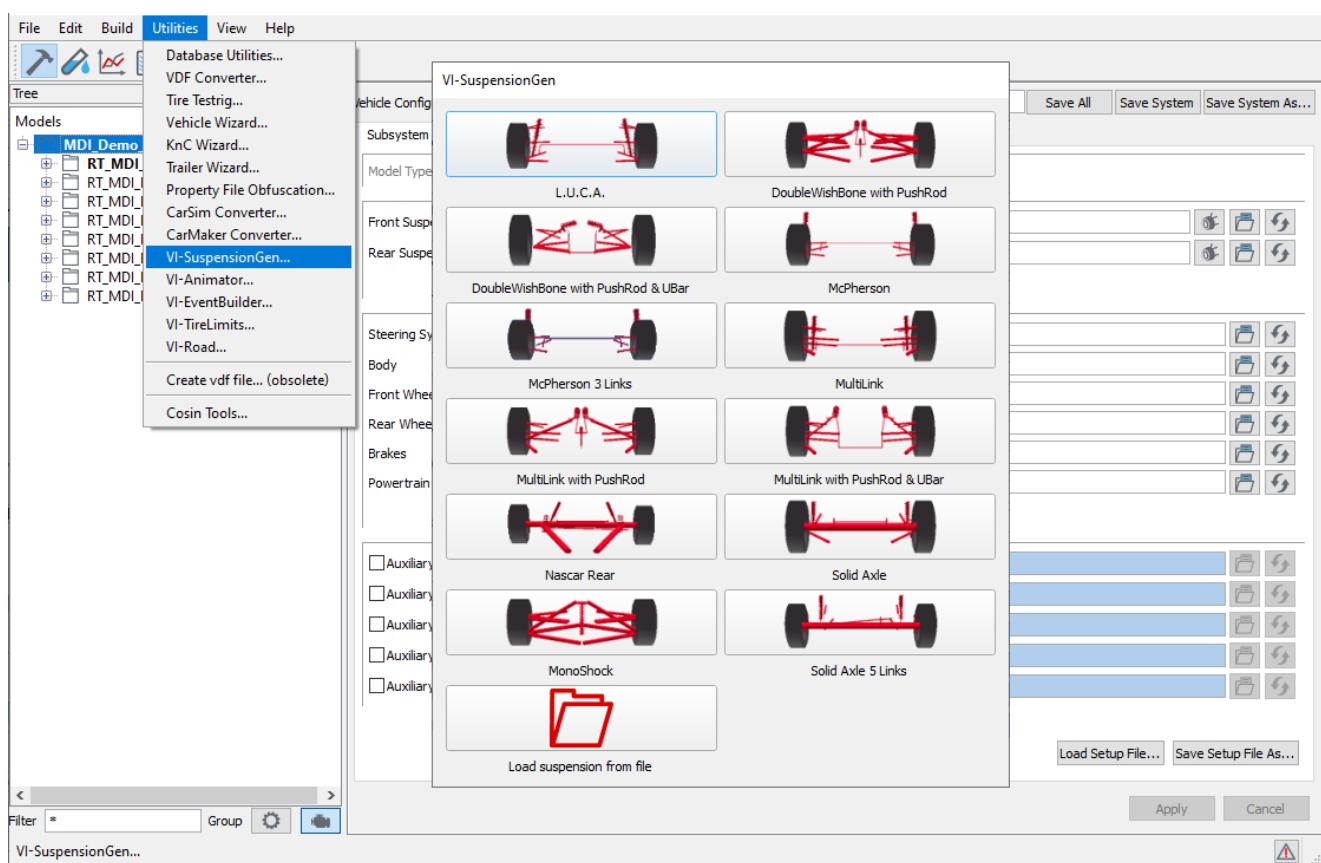
Start a new VI-CarRealTime session using the Windows Start Menu or by typing `vicrt20` from a DOS (cmd) shell.



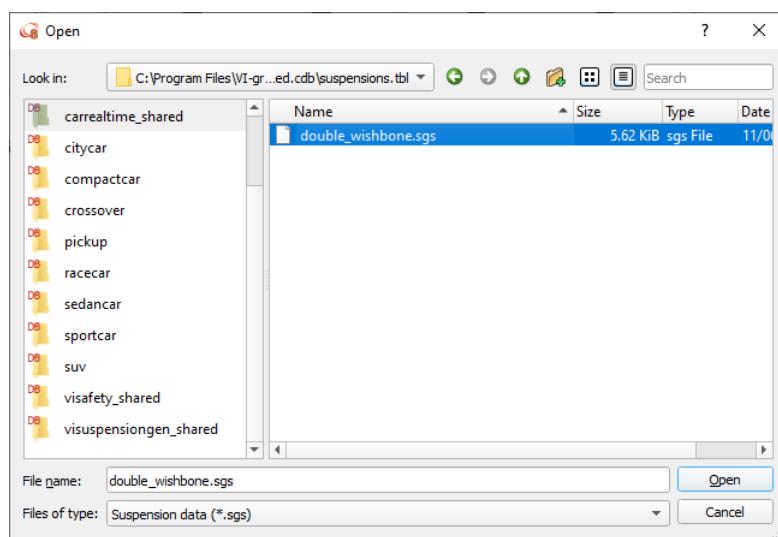
Open the `MDI_Demo_Vehicle.xml` vehicle model in **Build Mode** by clicking on the  icon, and locate it in the `carrealtime_shared` database.



Select **Utilities -> VI-SuspensionGen**. The VI-SuspensionGen main panel will then appear, as shown below. VI-SuspensionGen main panel shows up.

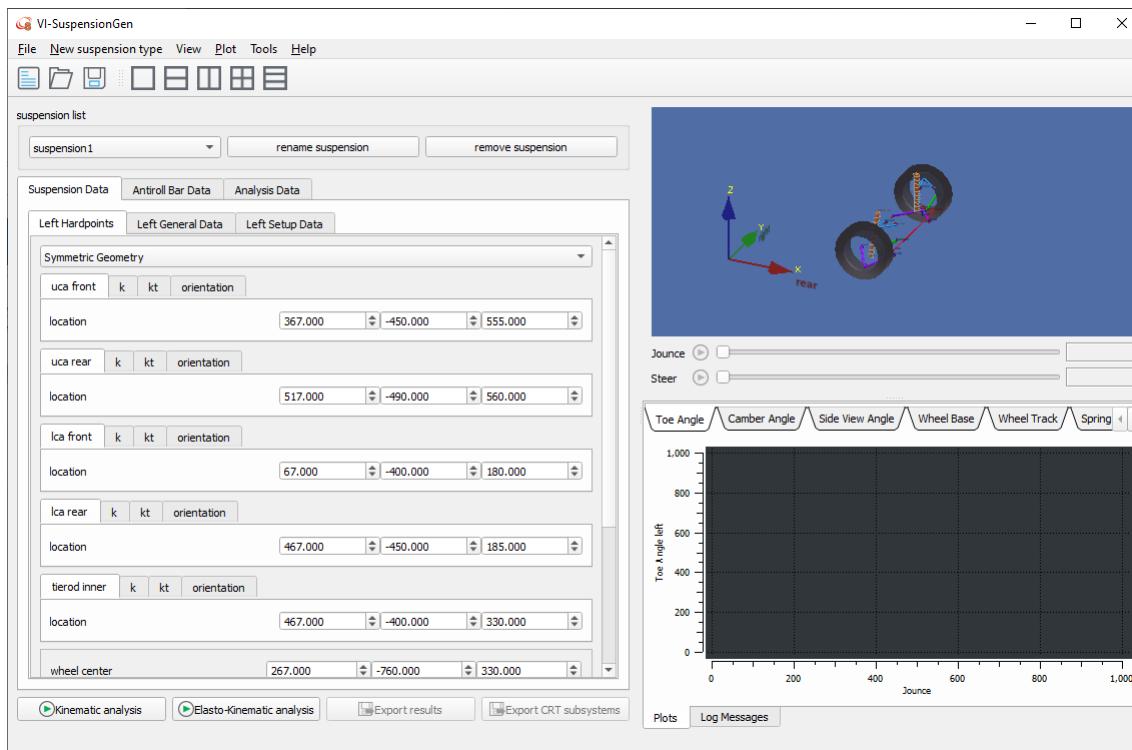


Left click on the **Load suspension from file** button to open the panel shown below, browse the `carrealtime_shared` database and select the **double_wishbone.sgs** suspension file.



This `double_wishbone.sgs` suspension file represents the `MDI_Demo_vehicle`'s front suspension, and will be used by the [Matlab API Tool](#) later in this exercise.

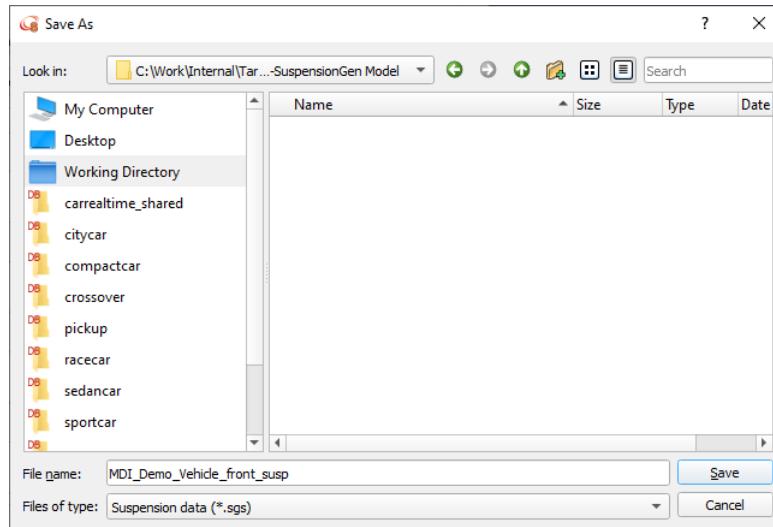
VI-CarRealTime



Note: Verify that the **Unit** for length (found under *Tools -> Units* in the main menu) of the current VI-SuspensionGen session are set to mm (Millimeter), because the MATLAB script file expects to receive its suspension data in millimeter.

Please save a copy of the *double_wishbone.sgs* suspension file in your current VI-CarRealTime working directory (the VI-SuspensionGen working directory is the same as the VI-CarRealTime's by default, as can be confirmed by selecting *Tools -> Settings* in VI-SuspensionGen's main menu), so as to edit it non-destructively.

To do so, click the **Save Suspension** icon (the blue floppy disk icon) in the VI-SuspensionGen toolbar, and save a copy of this suspension to your working directory, with the name **MDI_Demo_Vehicle_front_susp**.



The next part of the tutorial explains how to use the MATLAB interface to manage our double wishbone suspension parameters and to run VI-SuspensionGen in batch mode.

Notes: the script is embedded in "vict installation dir\lacarr\matlab\opt_lib\example\tutorial\ViSuspGen" folder.

As a first step, our MATLAB structure will be created from the .sgs file that was just saved to the working directory. Open MATLAB, and create a new script with the following code (which can be copied and pasted):

```
modelDirFiles='sgsDir';
cfgFile='vicrt_cdb.cfg';
suspensionGenFile='MDI_Demo_Vehicle_front_susp.sgs';

suspensionGenStruct=generateSuspensionGenStruct(suspensionGenFile,createCfg(cfg
File),modelDirFiles);
```

The suspensionGenSetValue is used to change the spring preload value.

```
%% modify single value %%
keys=cell(2,1);
keys{1}='leftSuspFile';
keys{2}='springPreload';

value = 7500; % spring preload

suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
```

The next operation, reflected in the code snippet below, changes the wheel center location by manipulating the suspensionGenSetLocation variable.

```
%% modify location %%
keys=cell(2,1);
keys{1}='leftSuspFile';
keys{2}='wheelCenter';

value = [267 -797 360]; % wheel center location

suspensionGenStruct=suspensionGenSetLocation(suspensionGenStruct,keys,value);
```

Finally the rear attribute is appropriately set, and then the VI-SuspensionGen simulation can be launched.

```
suspensionGenStruct=suspensionGenSetRole(suspensionGenStruct,'rear');

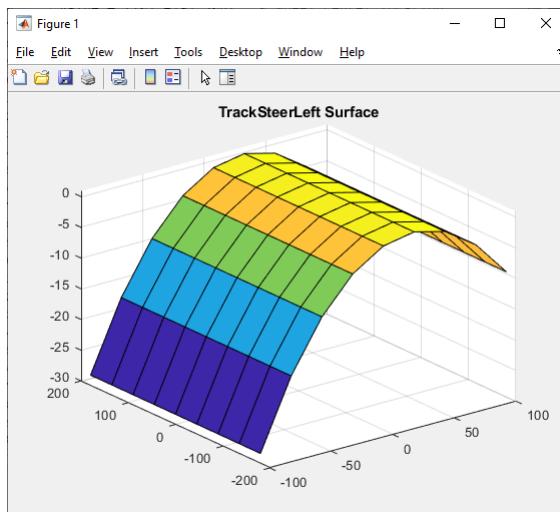
%% run vi-sg %%
outputPrefix='mySusp';
suspGenWorkingDir='suspGenWork';
[successFlag, suspensionGenStruct]= runViSg(suspensionGenStruct,
outputPrefix,suspGenWorkingDir);

%% review results
if (successFlag)
    sgOutData=load(suspensionGenStruct.output.mat);
    surf(sgOutData.JounceLeft,sgOutData.SteerLeft,sgOutData.TrackSteerLeft);
    title('TrackSteerLeft Surface');
end
```

Note: The above script is embedded in the "vicrt_installation_dir\lacart\matlab\opt_lib\example\tutorial\ViSuspGen" folder.

Save and run the script.

MATLAB Plot Tools easily allow you to visualize 3D output quantities such as, in this example, the left steer track as a function of the steering angle and of the (left) jounce.



Note: As expected, the suspension's jounce has the biggest effect on the track steer when compared to the steering angle.

Suspension Optimization

The present tutorial will give an overview of an advanced use of VI-CarRealTime and VI-SuspensionGen MATLAB library capabilities in order to perform a simplified optimization chain that will involve both solvers.

The optimization consists in modifying the suspension layout so that the vehicle yaw rate overshoot obtained by a step steer analysis is equal to a certain target.

The yaw rate overshoot is defined as follow:

$$\text{overshoot} = 100 \cdot \frac{\text{yaw rate}_{\text{MAX}} - \text{yaw rate}_{\text{SS}}}{\text{yaw rate}_{\text{SS}}}$$

The steps of the tutorial are:

- perform a step steer event in VI-CarRealTime and measure the overshoot;
- change toe spline shape in order to match the target overshoot value;
- modify suspension layout by sweeping an hardpoint location in order to match the target toe behaviour in a parallel travel analysis;
- save the optimized suspension layout and use this model in VI-CarRealTime to perform the same step steer analysis and check that the resulting yaw rate overshoot matches the target.

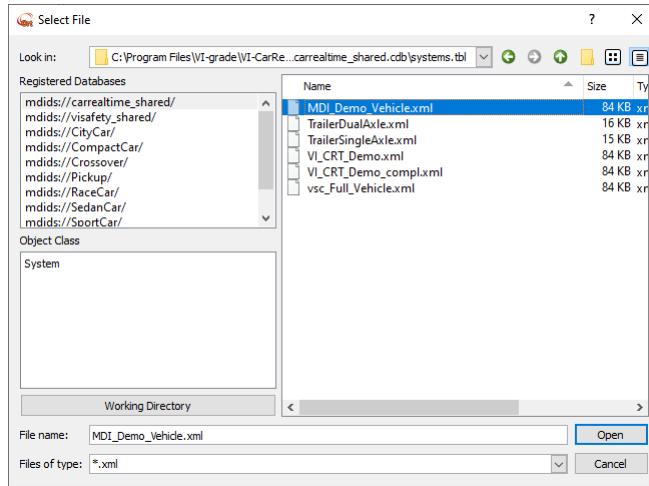
This tutorial is composed by two different sections:

- [VI-CarRealTime Interface](#)
load and modify model, run event and export all the files required by [Matlab API Toolkit](#).
- [Matlab Interface](#)
perform the optimization of the model, run VI-CarRealTime and VI-SuspensionGen solver in batch and plot results.

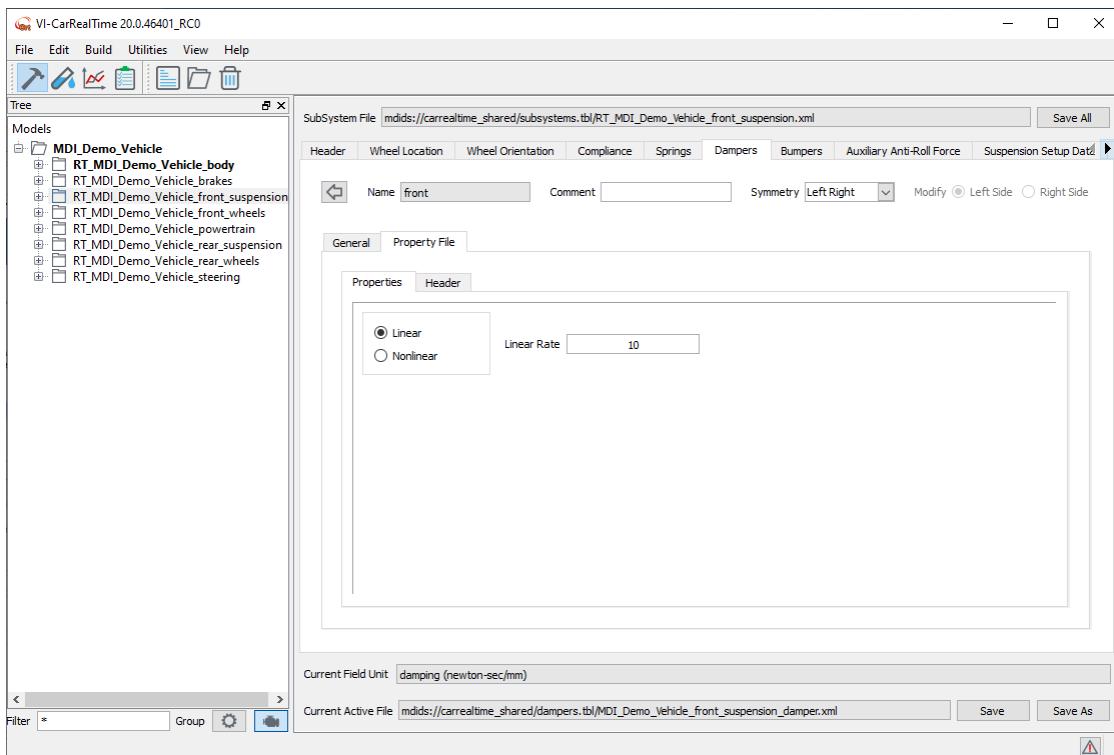
Run Event in VI-CarRealTime

As shown on [Manage VI-CarRealTime Model](#) tutorial, start a new VI-CarRealTime session using the Windows Start Menu or typing `vicrt20` from a dos shell.

Open `MDI_Demo_Vehicle.xml` by clicking on  icon and browsing to `carrealtime_shared` database.



In [Build Mode](#), browse to front suspension subsystem and then to *Damper* panel and change the **Damping Method** to **Linear** and select **10** (Ns/mm) as **Linear Rate**:



Save damper property file in working directory by selecting **Save As** button and choose **damper10.xml** as file name.

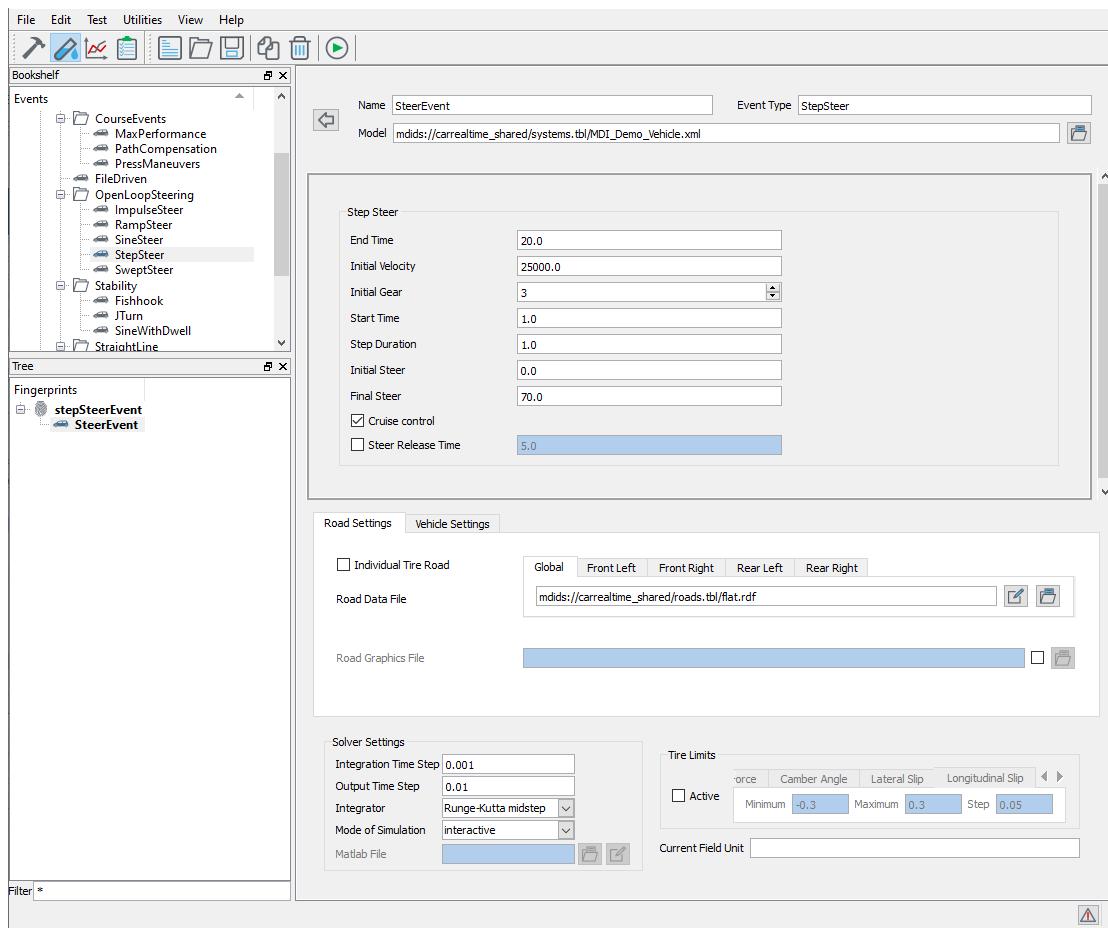
Note: check that you have in your working directory the file `damper10.xml` : this file will be used by the MATLAB script illustrated in the following topics.

Note: the damping rate is changed in order to match with the value stored in the double_wishbone.sgs file; the latter file will be used in MATLAB to run VI-SuspensionGen analysis with the modified toe curves.

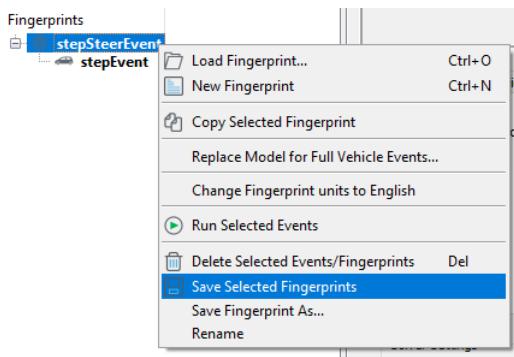


Switch to [Test Mode](#) by using button and select a [Step Steer](#) event; fill up the panel with the values shown below:

- Duration = **20** s;
- Initial velocity =**25** m/s;
- Initial gear = **3**;
- Start Time = **1** sec;
- Step duration = **1** sec;
- Initial steer = **0**;
- Final steer = **70** degrees;
- Cruise control active.



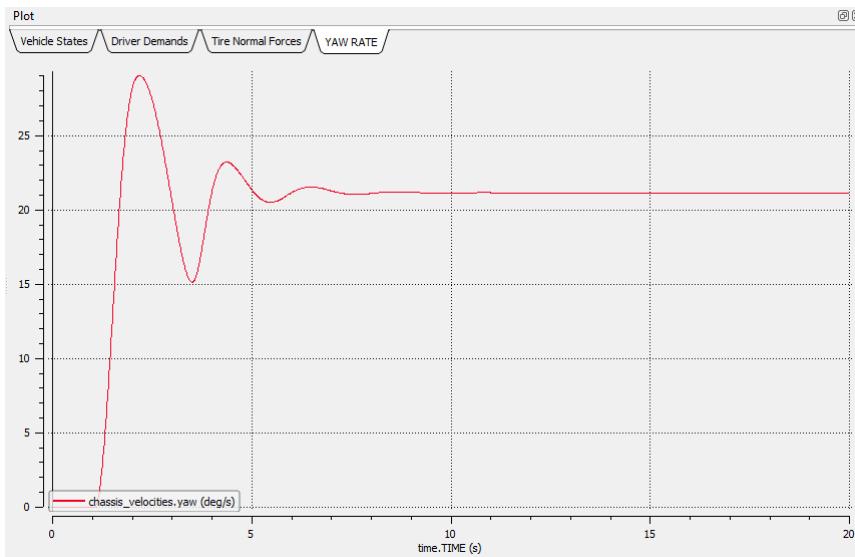
Rename the event **stepEvent**, and the fingerprint **stepSteerEvent** and save the fingerprint in your current VI-CarRealTime working directory as shown below:



Run **stepEvent** and wait until the simulation is complete.

Switch to [Review Mode](#) by using button, select the step steer event and press button to open VI-Animator.

Add a new page, rename it **YAW RATE** and plot the **chassis_velocities.yaw** vs. time.



The overshoot is:

$$\text{overshoot} = 100 \times (\text{YawRateMax} - \text{YawRateSteadyState}) \div \text{YawRateSteadyState} = 100 \times (29.02 - 21.15) \div 21.15 \\ = 37.21\%$$

Fit Toe Curves

From now on, we move to MATLAB Interface to write a script which allows us to perform the following actions:

- create systemStruct, generate toe third order polynomial curves and sweep the linear coefficient to generate new curves that will be used to run VI-CarRealTime solver; finally choose the set of toe curves which minimized the difference between the current evaluated overshoot and the target overshoot. [\[Part 1\]](#)
- create suspensionGenStruct, modify suspension tie-rod inner and outer location throughout a certain range and select the suspension layout whose toe splines best match with the previous chosen 4th set. [\[Part 2\]](#)

- run VI-SuspensionGen analysis with optimized suspension layout, replace the front suspension and steering subsystems in VI-CarRealTime model, run a VI-CarRealTime dynamic analysis and measure the resulting overshoot. [Part 3]

Change the working directory in MATLAB in order to be the same as the VI-CarRealTime one.

Open MATLAB and set as working directory the one used for VI-CarRealTime.

Notes: the script and function "vicrt_installation_dir\acarrt\matlab\opt\lib\example\tutorial\SuspOpt" are used in embedded in folder.

Write a new script file with the following commands:

```
addpath_vicrt_20; %% add library
```

Prepare MATLAB string variables for input file.

```
cfgFile='vicrt_cdb.cfg';
systemFile = 'mdids://carrealtime_shared/systems.tbl/MDI_Demo_Vehicle.xml';
baseFingerprint='stepSteerEvent.xml';
```

Create MATLAB structure variable to collect the vehicle data.

```
%% resolve system reference
systemStruct = generateSystemStruct(systemFile,createCfg(''));
```

Five different third order polynomial curve for parallel toe angle vs jounce splines will be used to modify the model front suspension. MATLAB variables `coeffL1` and `coeffR1` represent the basic coefficients for the polynomial curves. MATLAB variable `c3modifiers` represents the scaling factor for the linear order coefficient; toe splines that will be used for the optimization will have the following formulation:

$$\text{toe} = c(1)*x^3 + c(2)*x^2 + c3modifiers*c(3)*x + c(4)$$

```
overShootTarget = 31;

% initial guess polynomial coeff toe = c(1)*x^3+c(2)*x^2+c(3)*x + c4
coeffL1=[-2.806939797979820e-07    1.970828640692640e-04    -0.036059616565657
-0.023238822510822];
coeffR1=[-2.806939797979820e-07    1.970828640692640e-04    -0.036059616565657
-0.023238822510822];

jounce =[-40,-20,0,20,40];

% Study of influence of C(3) coefficient
% New spline will be generated using the following formulation
% toe = c(1)*x^3+c(2)*x^2+c3modifiers*c(3)*x + c4
c3modifiers=[0.0:0.375:1.5];

%%%%%%%%%%%%%
id_lateral_acc=figure;
title('Lateral Acceleration');
id_yaw=figure;
title('YawRate');
id_toe_L1=figure;
title('ToeL1 - toe = c(1)*x^3+c(2)*x^2+c3modifiers(i)*c(3)*x + c4');
id_toe_R1=figure;
title('ToeR1 - toe = c(1)*x^3+c(2)*x^2+c3modifiers(i)*c(3)*x + c4');
```

```

id_overShoot=figure;
title('overshoot');

Legend=cell(length(c3modifiers),3);
splineDataStoreL1 = cell(length(c3modifiers),1);
splineDataStoreR1 = cell(length(c3modifiers),1);
overShoot= ones(length(c3modifiers),1);

```

Five different third order polynomial curve for parallel toe angle vs jounce splines will be used to modify the model front suspension. MATLAB variables `coeffL1` and `coeffR1` represent the basic coefficients for the polynomial curves. MATLAB variable `c3modifiers` represents the scaling factor for the linear order coefficient; toe splines that will be used for the optimization will have the following formulation:

```

%% set constant value for damper (same as sgs)

values=cell(1);
keys=cell(2,3);
keys{1,1}='DamperPair';
keys{1,2}='Damper';
keys{1,3}='CRTDamperContext';

keys{2,1}='front';
keys{2,2}='left';
keys{2,3}='';

attributes{1} = 'propertyURI';
values{1} = 'damper10.xml';
%replace standard curve
systemStruct = frontSuspensionSetPropertyFile(systemStruct,keys,attributes,values);

```

Next step will be the steering subsystem modifications

```

%% Steering Subsystem Modifications %%
keys = cell(2,1);
keys{1,1}='CRTSteeringSystem';
keys{2,1}='steering_gear';

attributes= cell(1,1);
attributes{1}='method';

values=cell(1,1);
values{1}='use_toe';

systemStruct = steeringSetValue(systemStruct,keys,attributes,values);

```

Next step will be the creation of main loop. In the first part of loop the new toe spline2D will be generated and then will be used to modify the existing one. In order to do that, it will be create cell array variables to collect the list of **(tag : name)** strings required to resolve xml tree.

```
key(1,3)           key(2,3)
<CRISuspensionData name="toe_Jounce" active="true" userDefined="false">
...     <Spline3d>
key(1,4) name="table" key(2,4)
...     active="true"
...     userDefined="false"
...     interpolation_method="akima"
...     xLabel="Jounce"
...     yLabel="Toe"
...     xUnit="length"
...     yUnit="angle"
...     datablock="Toe vs. Jounce"
...     zLabel="Paired Wheel Jounce"
...     zUnit="length"

for i=1:length(c3modifiers)
    %% modify spline 3d kinematic
    keys=[];
    values=[];
    %update polynomial coefficients
    coeffL1Mod=coeffL1;
    coeffL1Mod(3)=coeffL1Mod(3)*c3modifiers(i);
    splineDataL1=createSplineStructByPolynomialCoeff(coeffL1Mod,jounce);

    %replace standard curve
    keys=cell(2,4);
    keys{1,1}='CRTKinematicsPair';
    keys{1,2}='CRTKinematics';
    keys{1,3}='CRISuspensionData';
    keys{1,4}='Spline3d';

    keys{2,1}='front';
    keys{2,2}='left';
    keys{2,3}='toe_Jounce';
    keys{2,4}='table';

    values=cell(1);
    values{1} = splineDataL1;
    splineDataStoreL1{i}= splineDataL1;
    %replace standard curve
    systemStruct = frontSuspensionSetSpline3D(systemStruct,keys,values);

    %% modify spline 3d kinematic
    keys=[];
    values=[];

    %update polynomial coefficients
    coeffR1Mod=coeffR1;
    coeffR1Mod(3)=coeffR1Mod(3)*c3modifiers(i);
    splineDataR1=createSplineStructByPolynomialCoeff(coeffR1Mod,jounce);

    %replace standard curve
    values=cell(1);
    keys=cell(2,4);
    keys{1,1}='CRTKinematicsPair';
    keys{1,2}='CRTKinematics';
    keys{1,3}='CRISuspensionData';
    keys{1,4}='Spline3d';

    keys{2,1}='front';
```

```

keys{2,2}='left';
keys{2,3}='toe_Jounce';
keys{2,4}='table';
values{1} = splineDataR1;
splineDataStoreR1{i}= splineDataR1;
%replace standard curve
systemStruct = frontSuspensionSetSpline3D(systemStruct,keys,values);

```

Step event will be now performed with the modified model and some MATLAB plot will appear. They collect the toe splines, the time histories for frame yaw rate and the frame lateral acceleration.

```

%% update fingerprint and run %%
%%%%%%%%%%%%%
out_prefix = sprintf('%s-[%g]', 'c3Multiplier', c3modifiers(i));

workingDir ='crtWork';
[successFlag, outputNameList]
=runViCrt(baseFingerprint,systemStruct,num2str(i),workingDir);

Legend{i}=out_prefix;
% load and plot body roll angle
output=sprintf('%s.mat',outputNameList{1});
out=load(output);
%plot yaw rate
figure(id_yaw)
title('YawRate');
plot(out.time,out.chassis_velocities_yaw,'color',[0.1*i/(length(c3modifiers)),0.5*i/(length(c3modifiers)) ,0.9*i/(length(c3modifiers))]);
hold on
%plot lateral acceleration
figure(id_lateral_acc);
title('Lateral Acceleration');
plot(out.time,out.chassis_accelerations_lateral,'color',[0.1*i/(length(c3modifiers)),0.5*i/(length(c3modifiers)) ,0.9*i/(length(c3modifiers))]);
hold on;
%compute overshoot ratio
overShoot(i)=(max(abs(out.chassis_velocities_yaw))-out.chassis_velocities_yaw(end))/out.chassis_velocities_yaw(end) *100;
clear out;
%plot toe L1 curve
figure(id_toe_L1);
title('ToeL1 - toe = c(1)*x^3+c(2)*x^2+c3modifiers(i)*c(3)*x + c4');
plot(jounce,polyval(coeffL1Mod,jounce),'color',[0.1*i/(length(c3modifiers)),0.5*i/(length(c3modifiers)) ,0.9*i/(length(c3modifiers))]);
hold on
%plot toe R1 curve
figure(id_toe_R1);
title('ToeR1 - toe = c(1)*x^3+c(2)*x^2+c3modifiers(i)*c(3)*x + c4');
plot(jounce,polyval(coeffR1Mod,jounce),'color',[0.1*i/(length(c3modifiers)),0.5*i/(length(c3modifiers)) ,0.9*i/(length(c3modifiers))]);
hold on
end

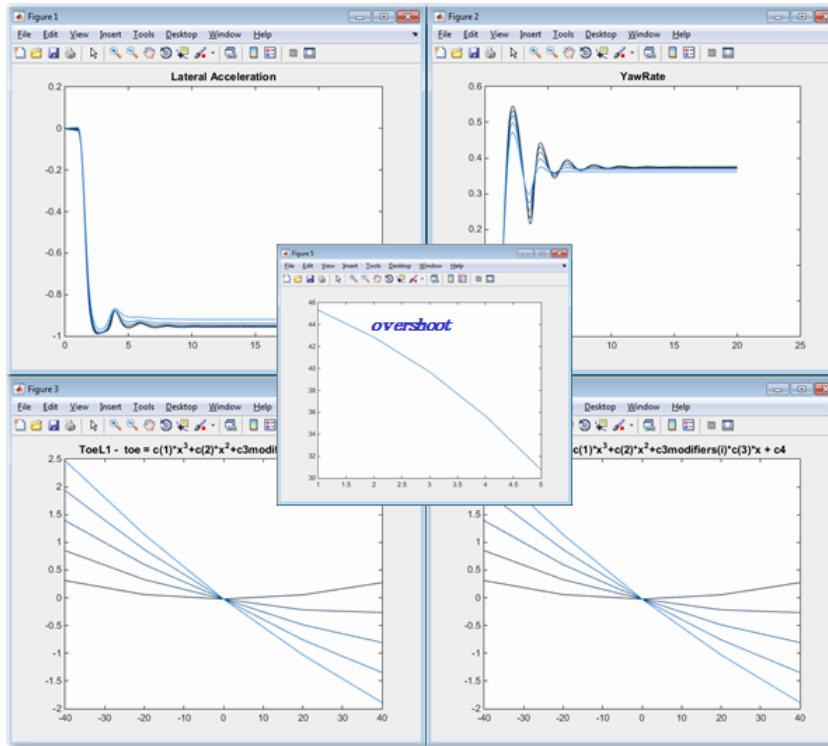
figure(id_overShoot);
title('overshoot')
plot(overShoot)

%%% search optimal maneuvre that minimize the error
[target, targetIndex] = min(abs(overShoot - overShootTarget));
targetToeL1 = splineDataStoreL1{targetIndex}.parallelToe;
targetToeR1 = splineDataStoreR1{targetIndex}.parallelToe;

```

Save and run the script.

Simulation that better matches the target of 34% for the overshoot ratio will be the number 4.
Target spline for VI-SuspensionGen will be created.



This function is used to create a spline structure from polynomial coefficients (coeff) and jounce positions vector (jounce).

For the purpose of this tutorial, it's necessary to create a function file with this code and store the file in the MATLAB working directory or in one of the directories of MATLAB search path.

```
function splineData = createSplineStructByPolynomialCoeff(coeff, jounce)

y = polyval(coeff, jounce);

splineData.x = jounce;
splineData.y = 0;

splineData.f_xy=[];
splineData.f_xy = zeros(length(splineData.x),length(splineData.y));

for i=1:length(splineData.x)
    for j=1:length(splineData.y)
        splineData.f_xy(i,j)=y(i);
    end
end

splineData.polyCoeff=coeff;
splineData.parallelToe=y;
```

Note: to create a function .m file simply create an empty file and fill it up with the code below, then save it with the same name as the function (createSplineStructByPolynomialCoeff).

In this part the MATLAB structure variable to collect the suspension data will be created. Using the template suspension `double_wishbone.sgs`, tie-rod inner and outer hardpoints will be moved along y direction in order to generate toe curves (both for left and right side) that will match with VI-CarRealTime optimized splines. Original value for y location will be altered using a delta variation. In order to retrieve the values, a cell array variables to collect the list of (tag; name) strings required to resolve sgs tree will be created.

7	<code>leftSuspFile</code>	:	suspType	DoubleWishBone		
8		:	suspName	suspension1		
9		:	latPosType	left		
10		:	lonPosType	front		
11			wheelCenter	267	-760	330
12	<code>keys</code>		ucaFront	367	-450	555
13			ucaRear	517	-490	560
14			ucaOuter	307	-675	555
15			lcaFront	67	-400	180
16			lcaRear	467	-450	185
17			lcaOuter	267	750	130
18			tierodOuter	417	-400	330
19			tierodInner	467	-750	330
20			springUpperMount	307	-500	680
21			springLowerMount	267	-600	180
22			damperUpperMount	307	-500	680
23			damperLowerMount	267	-600	180
24			arbLink	358.6	-525.34	180.28
25			toeAngle	0		
26						

```

suspensionGenFile='mdids://carrealtime_shared/suspensions.tbl/double_wishbone.sgs';

suspensionGenStruct=generateSuspensionGenStruct(suspensionGenFile,createCfg(cfgFile));

%
outputPrefix='mySuspDW';
suspGenWorkingDir='suspGenWorkDW';

numYLocRange = 5;
%% change tierod Y outer position in order to match polynomial curve %%
outTieLocLeftRange = linspace(-50,-90, numYLocRange);
%% change tierod Y inner position in order to match polynomial curve %%
inTieLocLeftRange = linspace(-50,50, numYLocRange);

suspGenStructCollector = cell(numYLocRange,1);

numJounce = 5;
jounceRange = linspace(-40, 40, numJounce);

% set suspGen jounce range and number of steps
keys=cell(2,1);
keys{1}='analysisFile';

keys{2}='jounceStart';
value = min(jounceRange);
suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
keys{2}='jounceEnd';
value = max(jounceRange);
suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
keys{2}='nJounce';
value = max(size(jounceRange))-1;
suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
keys{2}='nSteer';
value = max(size(jounceRange))-1;
suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
keys{2}='nForce';
value = max(size(jounceRange))-1;

```

VI-CarRealTime

```

suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);
keys{2}='nRoll';
value = max(size(jounceRange))-1;
suspensionGenStruct=suspensionGenSetValue(suspensionGenStruct,keys,value);

keys=cell(2,1);
keys{1}='leftSuspFile';
keys{2}='tierodOuter';
leftFrontTieOutLoc = suspensionGenGetValue(suspensionGenStruct,keys);

keys=cell(2,1);
keys{1}='leftSuspFile';
keys{2}='tierodInner';
leftFrontTieInLoc = suspensionGenGetValue(suspensionGenStruct,keys);

leftToe = zeros(numJounce, numYLocRange);
rightToe = zeros(numJounce, numYLocRange);

```

Next step will be the creation of VI-SuspensionGen main loop. In the first part of loop the hardpoint positions will be generated and then will be used to modify the existing ones. Then a suspension computation will be performed and the resultant toe angle will be saved in MATLAB variables.

```

% modify locations
for i=1:numYLocRange

    keys{2}='tierodOuter';
    value = [leftFrontTieOutLoc(1) leftFrontTieOutLoc(2)+outTieLocLeftRange(i)
    leftFrontTieOutLoc(3)];
    suspensionGenStruct=suspensionGenSetLocation(suspensionGenStruct,keys,value);

    keys{2}='tierodInner';
    value = [leftFrontTieInLoc(1) leftFrontTieInLoc(2)+inTieLocLeftRange(i)
    leftFrontTieInLoc(3)];
    suspensionGenStruct=suspensionGenSetLocation(suspensionGenStruct,keys,value);

    [successFlag, suspensionGenStruct]= runViSG(suspensionGenStruct, outputPrefix,
    suspGenWorkingDir);

    suspGenStructCollector{i}=suspensionGenStruct;

    if (successFlag)
        sgOutData=load(suspensionGenStruct.output.mat);
        leftToe(:,i) = sgOutData.ToeLeft;
        rightToe(:,i) = sgOutData.ToeRight;
    else
        error('vi-sg failure')
    end
end

```

The optimized VI-SuspensionGen configuration will be chosen by minimizing the cumulative error between simulation splines and target ones.

```

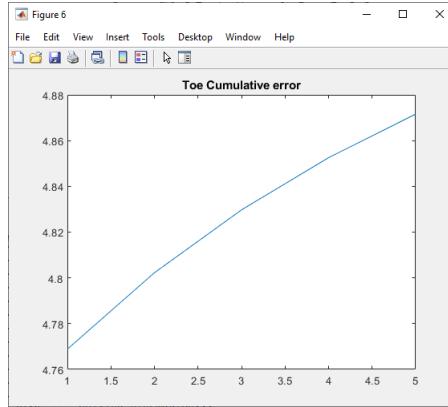
toeDeviation = zeros(numYLocRange, 1);
for i=1:numYLocRange
    toeDeviation(i) = sum(abs(leftToe(:,i) - targetToeL1')) + sum(abs(rightToe(:,i) -
targetToeR1'));
end

figure

```

```
plot(toeDeviation)
title('Toe Cumulative error')

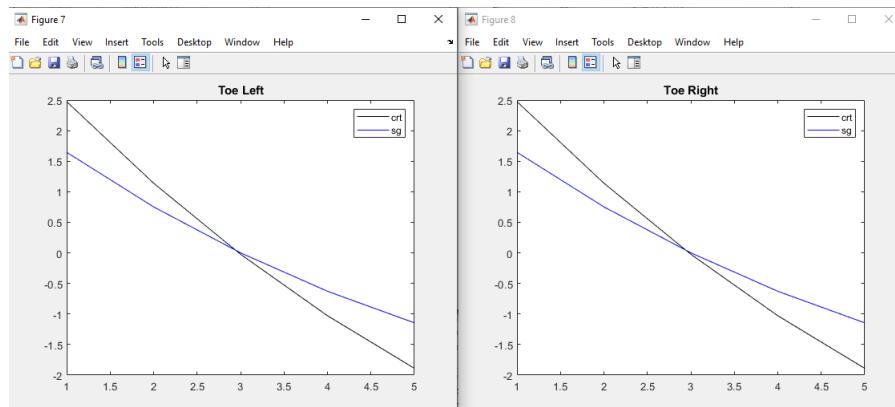
%%% retrieve ID of minimum error %%%
[sgMinError, sgMinId]=min(toeDeviation);
```



Then a comparison between toe splines will be shown:

```
figure
plot(targetToeL1, 'k');
hold on
plot(leftToe(:,sgMinId), 'b');
legend('crt','sg');
title('Toe Left');

figure
plot(targetToeR1, 'k');
hold on
plot(rightToe(:,sgMinId), 'b');
legend('crt','sg');
title('Toe Right');
sgMinId;
```



The last part consists in the computation of optimized suspension configuration running VI-SuspensionGen in batch mode in order to export the files in xml format. These files will replace the front suspension and steering subsystem in VI-CarRealTime model.

```
%% final comparison %%
outputPrefix='mySusp';
suspGenWorkingDir='final';
```

```
[successFlag, suspensionGenStruct]= runViSg(suspGenStructCollector{sgMinId},
outputPrefix,suspGenWorkingDir);

workingDir ='final';

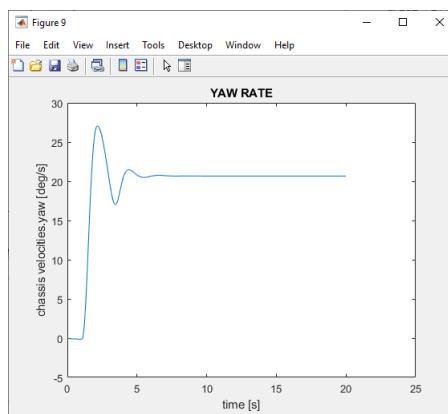
%% update system with new suspension file
systemStruct.frontSuspension.file =suspensionGenStruct.output.suspensionFile;
systemStruct.steering.file =suspensionGenStruct.output.steeringFile;

%% Steering Subsystem Modifications %%
keys = cell(2,1);
keys{1,1}='CRTsteeringSystem';
keys{2,1}='steering_gear';
attributes= cell(1,1);
attributes{1}='method';
values=cell(1,1);
values{1}='use_toe';
systemStruct = steeringSetValue(systemStruct,keys,attributes,values);
```

Finally, a VI-CarRealTime simulation will be performed using the updated subsystems and the plot for yaw rate will appear, together with the new value of the overshoot.

```
%% run vicrt
[successFlag, outputList]=runViCrt(baseFingerprint,systemStruct,'last',workingDir);

%% output plot
output=sprintf('%s.mat',outputList{1});
figure;
out=load(output);
plot(out.time,out.chassis_velocities_yaw*(180/pi));
title('YAW RATE');
xlabel('time [s]');
ylabel('chassis velocities.yaw [deg/s]');
overShootF=(max(abs(out.chassis_velocities_yaw))- ...
out.chassis_velocities_yaw(end))/out.chassis_velocities_yaw(end) *100
clear out;
hold on;
sgMinId;
```



The computed overshoot is **30.8347**, quite the same as the [target](#) value (31).

1.4.3 Using FMI Master

The tutorials in the following section are intended to make users more familiar with the FMI Master cosimulation interface, in order to use VI-CarRealTime models together with external components using the FMI interface.

Below is a list of the exercises in this tutorial section:

- [FMI: External Driveline](#)
- [FMI: External Suspension](#)

FMI: External Driveline

This tutorial shows how to use [FMI Master](#) available as auxiliary subsystem to replace the internal driveline of the vehicle model with an FMU (Functional Mockup Unit) driveline.

The FMU used for this tutorial has been generated from a Driveline system modeled using Modelica Standard Library (MSL 3.2).

The FMU includes the engine flywheel, the clutch, the gear box, and the differential. The Driveline model has the following I/O channels and parameters:

output channels

```
'transmission_ratio': Gear ratio of the selected gear  
'engine_omega': Engine angular speed (clutch input shaft)  
'clutch_torque': Clutch torque  
'LW_tau': Left driving wheel torque  
'RW_tau': Right driving wheel torque  
'clutch_slip': Clutch slip  
'clutch_slip_der': Clutch slip derivative  
'transimission_omega': Transmission angular velocity (clutch output shaft)  
'clutch_flag': Boolean flag to deactivate internal clutch model  
'gearbox_flag': Boolean flag to deactivate internal gearbox model
```

input channels

```
'clutch_demand': Clutch pedal (0-1)  
'engine_torque': Engine mean torque calculated from the engine rpm-torque characteristic curve  
'gear_ID': Current gear  
'LW_omega': Left driving wheel angular speed  
'RW_omega': Right driving wheel angular speed
```

clutch parameters

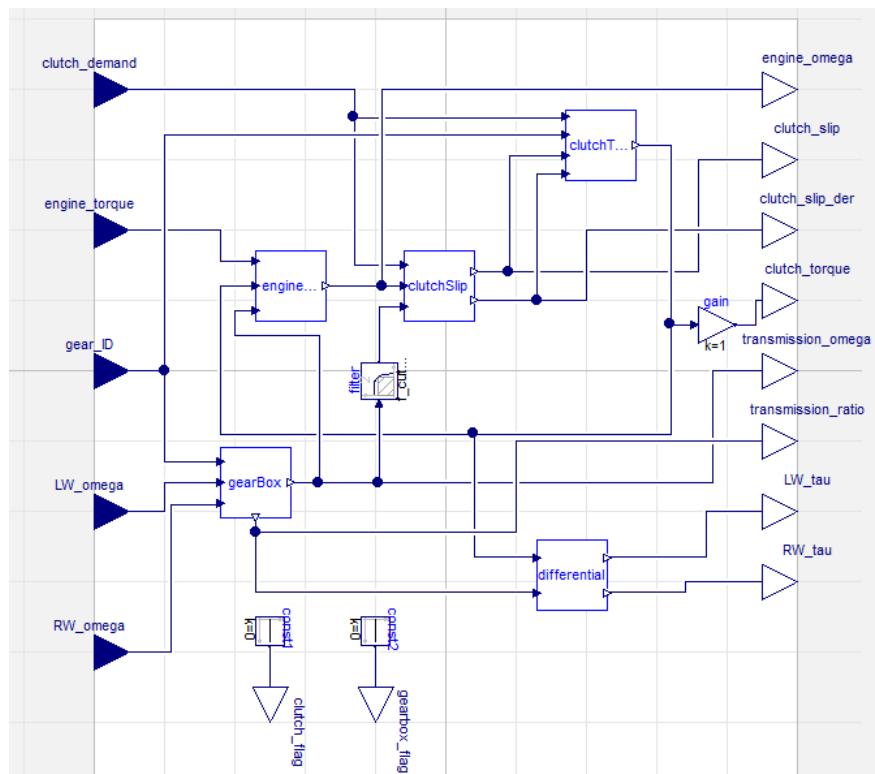
```
'clutch_stiffness': K (default = 1e3)  
'clutch_capacity': clutch torque limit (default = 1e3)  
'clutch_damping': C (default = 45.8366236105)  
'clutch_open': clutch demand value at which the clutch is totally open (<=1.0, default = 0.75)  
'clutch_close': clutch demand value at which the clutch is totally closed (>=0.0, default = 0.25)  
'clutch_tau': clutch time constant(default = 0.1e-1)
```

driveline inertia

```
'engine_inertia': engine inertia moment (default = 0.5e-1)
```

transmission ratios

```
'bevel_gear': ratio of differential bevel gear 1 (default = 1.0)
'gear_ratio_1': gear ratio of gear 1 (default = 3.50)
'gear_ratio_2': gear ratio of gear 2 (default = 1.94)
'gear_ratio_3': gear ratio of gear 3 (default = 1.26)
'gear_ratio_4': gear ratio of gear 4 (default = 0.93)
'gear_ratio_5': gear ratio of gear 5 (default = 0.76)
'gear_ratio_6': gear ratio of gear 6 (default = 0.675)
'gear_ratio_7': gear ratio of gear 7 (default = 0.55)
'gear_ratio_8': gear ratio of gear 8 (default = 0.0)
'gear_ratio_null': gear ratio of null gear (default = 0.0)
'gear_ratio_reverse': gear ratio of reverse gear (default = -3.0)
'final_drive': differential drive ratio (default = 3.7)
'drop_gear': crank shaft/main shaft ratio (default = 1.00)
```



Note: Before starting the following operations please make a copy of the FMU archive **Driveline2WD.fmu** placed under **\$VI-CarRealTime install dir\acarr\examples\fmi\fmi2\\$** into the working directory.

To couple VI-CarRealTime to the driveline modeled by the FMU, the following step are required:

1. create the FMI Master configuration file

After including the header and units block, according to the [FMI Master configuration file documentation](#), the [FMI_MASTER_PARAMETERS] block has to be set as follows.

```
$-----
FMI_MASTER_PARAMETERS
[FMI_MASTER_PARAMETERS]
(FMU_TABLE)
{ fmu_block }
'FMU_1'
```

In that way, the FMI Master is configured to manage the FMU specified by the block FMU_1. Now, add the block [FMU_1] to the configuration file,

```
$-----FMU_1
[FMU_1]
```

specifying the FMU archive full path

```
FMU_PATH='.\Driveline2WD.fmu'
```

the fmus instance name

```
INSTANCE_NAME='FMU_Driveline'
```

the execution id

```
EXECUTION_ID=1
```

the communication step between VI-CarRealTime and the FMU

```
COMMUNICATION_STEP=1e-3
```

the log level to 'ERROR' in order to show only the error messages coming from the FMU.

```
LOG_LEVEL='WARNING'
```

Regarding the input connections map, it has to be configured writing in the first column the set of VI-CarRealTime output channels used to feed the FMU input channels placed in the second column.

The channels are identified by their names, retrievable from the [VI-CarRealTime output channels](#) list, and from the FMU documentation placed in the directory documentation of the fmus archive.

```
(INPUT_CONNECTIONS_MAP)
{vicrt_output_chl fmu_input_chl}
'Wheel.Omega.L2'          'LW_omega'
'Wheel.Omega.R2'          'RW_omega'
'driver_demands.gear'    'gear_ID'
'driver_demands.clutch'  'clutch_demand'
'engine.torque'           'engine_torque'
```

Regarding the output connections map, it has to be configured writing in the first column the set of FMU output channels used to feed the set of VI-CarRealTime input channels placed in the second column.

The channels are identified by their names, retrievable from the [VI-CarRealTime input channels](#) list, and from the FMU documentation placed in the directory documentation of the FMU archive.

```
(OUTPUT_CONNECTIONS_MAP)
{fmu_output_chl vicrt_input_chl}
'LW_tau'                  'Wheel_driving.moment.L2'
'RW_tau'                  'Wheel_driving.moment.R2'
'transmission_ratio'      'GearBox.transmission_ratio'
'engine_omega'             'Engine.engine_omega'
'clutch_flag'              'Subsystem_Activation.Internal_Clutch_Activity'
'gearbox_flag'              'Subsystem_Activation.Internal_Gearbox_Activity'
```

VI-CarRealTime

Note that clutch output torque is not connected to the corresponding VI-CarRealTime input. This is because the clutch torque and the wheel torques are additive and if the external model provides both, their values will be summed internally. Since the example external models contains also a differential model, the torque are injected in VI-CarRealTime directly at wheel.

Regarding the initial conditions map, it has to be configured writing in the first column the FMU state name that has to be initialized and in the second column the name of VI-CarRealTime initial condition related to it.

Note: in this case there is no FMU state that needs to be initialized.

```
(INITIAL_CONDITIONS_MAP)
{fmu_initial_condition_name          vicrt_initial_condition_name}
```

After configuring the connections between VI-CarRealTime and the FMU, the last step is configure the driveline parameters (transmission ratios and inertia), specifying the names of the FMU parameters set that have to be changed (see FMU documentation) in the first column and their value in the second one.

```
(PARAMETERS_MAP)
{fmu_parameter_name    fmu_parameter_value}
'bevel_gear'           1
'clutch_capacity'     1000
'clutch_close'         0.25
'clutch_damping'       45.8366236105
'clutch_open'          1.0
'clutch_stiffness'    114.5915590262
'clutch_tau'           0.05
'drop_gear'            1
'engine_inertia'       0.07
'final_drive'          3.36
'gear_ratio_1'          2.88
'gear_ratio_2'          1.77
'gear_ratio_3'          1.27
'gear_ratio_4'          1.0
'gear_ratio_5'          0.83
'gear_ratio_6'          0.61
'gear_ratio_7'          0.0
'gear_ratio_8'          0.0
'gear_ratio_null'       0.0
'gear_ratio_reverse'    -3
```

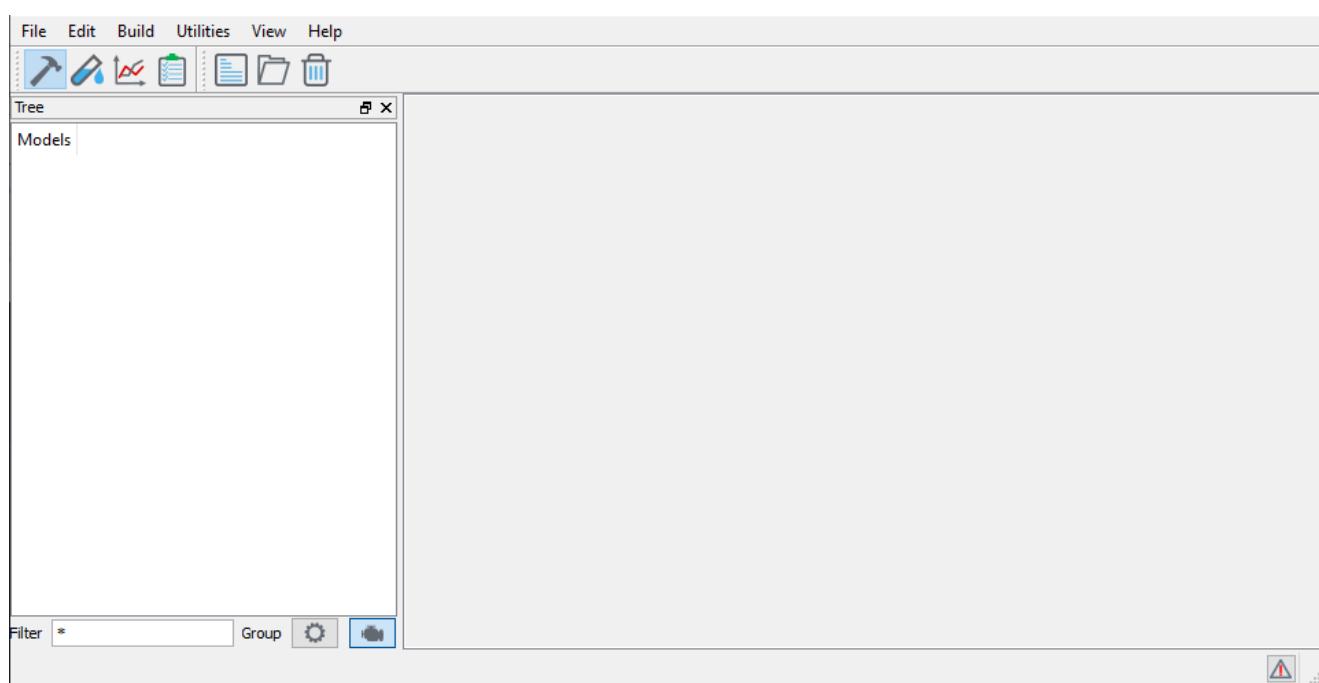
Note: In this tutorial the FMU documentation was reported at the beginning of the chapter together with its description.

The FMI Master configuration file is now completed. Save the file in the working directory as **Driveline2WD.vfm**.

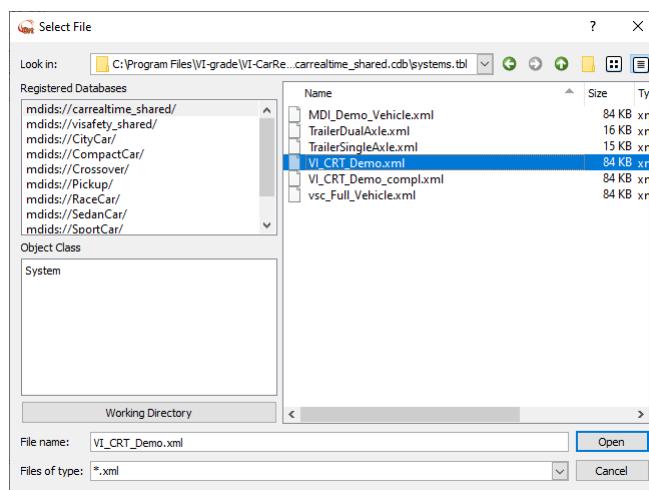
Note: An example Driveline2WD.vfm configuration file is also available under **\$VI-CarRealTime install dir\acarrt\examples\fmi\fmi2\\$**.

2. configure the FMI Master auxiliary subsystem

Start a new VI-CarRealTime from the current working directory typing **vicrt20** from a dos shell.

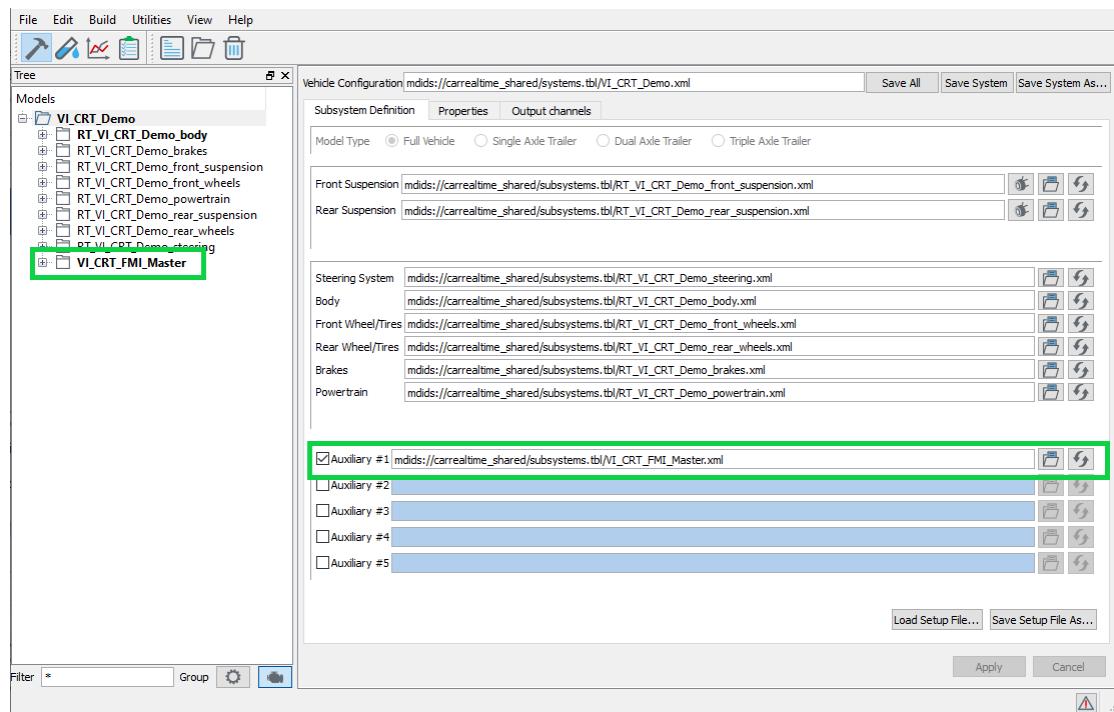


Open **VI-CRT_Demo.xml** by clicking on icon and browsing to **carrealtime_shared** database.

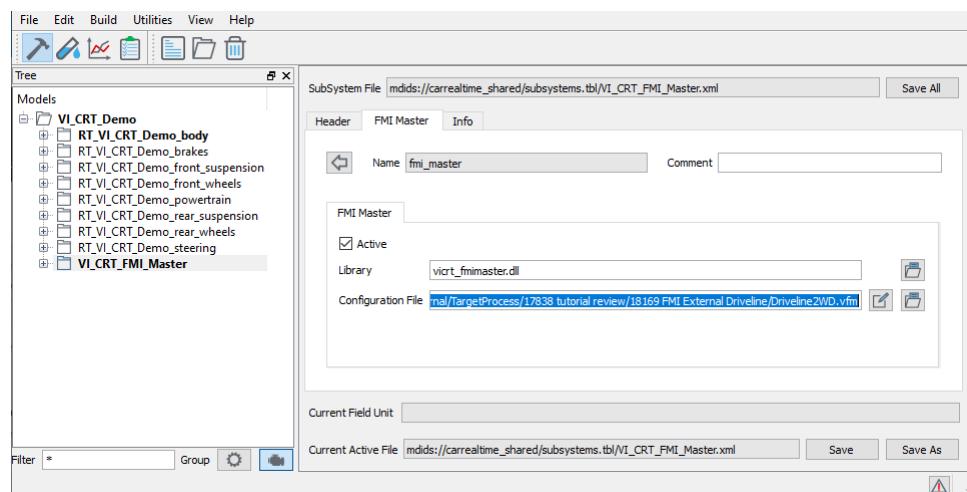


Enable the **Auxiliary** subsystem field, select **VI_CRT_FMI_Master** auxiliary subsystem from the **carrealtime_shared** database by clicking on and apply the modification.

VI-CarRealTime



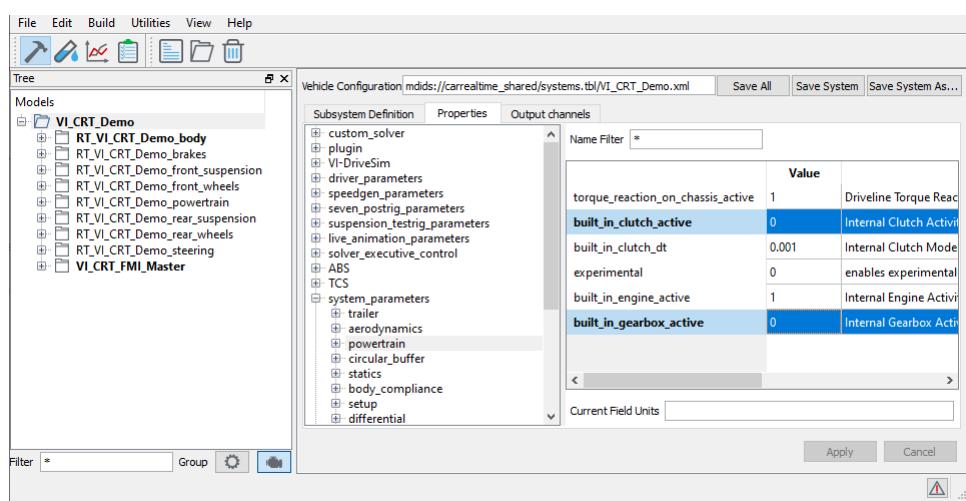
Edit the `VI_CRT_FMI_Master` subsystem from the model tree, set the FMI Master tab, enable the **Active** flag, and select the configuration file created previously.



3. disable the internal driveline

Before running the simulation, the internal driveline components replaced with the parts modeled by the FMU, have to be disabled.

Select the `VI_CRT_Demo` system, set to 0 the `built_in_clutch_active` flag and the `built_in_gearbox_active` flag from the property tree (`system_parameters>powertrain`) and apply the modification.

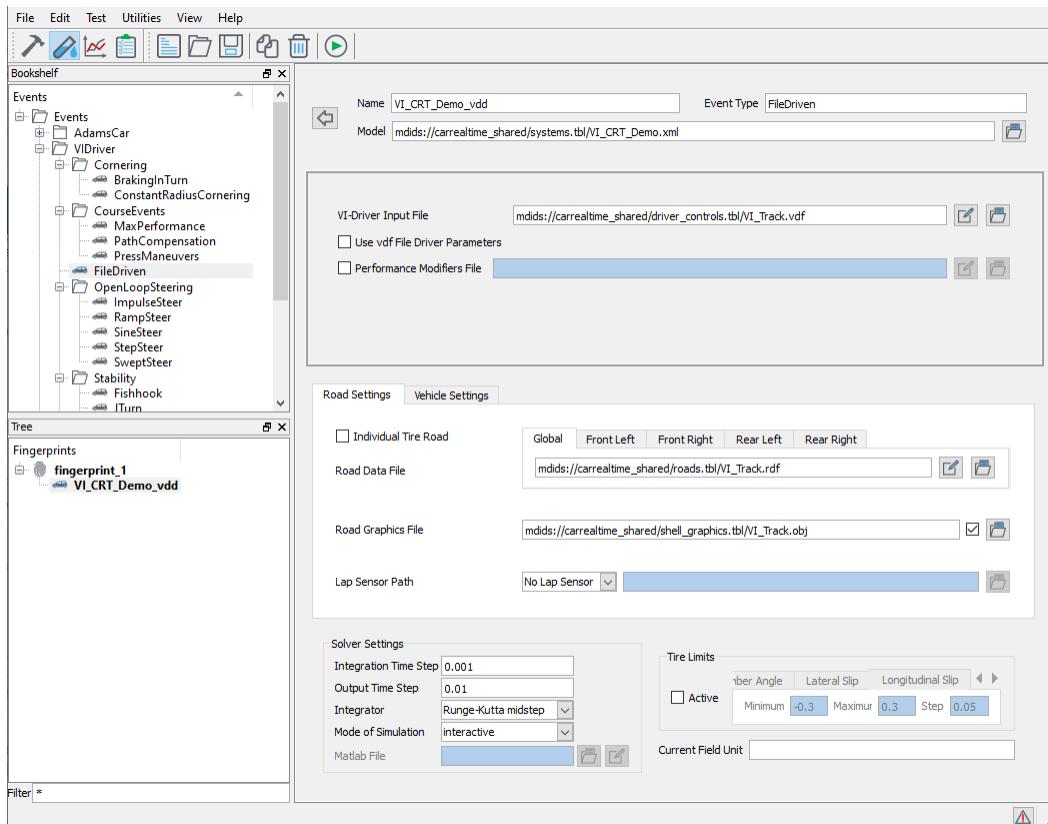


Note: The internal clutch and the internal gear box could be disabled also connecting the related VI-CarRealTime subsystem_activation inputs to an activation signal (set to 0.0) generated by the FMU. Using this option has the advantage that the same system can be simulated with or without the FMI subsystem simply changing its' activity flag, without any additional modification.

4. run the simulation

To run a simulation using the FMI Master, create a new event using the VI_CRT_Demo vehicle, select the VI-

Driver Input File VI_Track.vdf, the Road Data File VI_Track.rdf and click on the button to start the simulation.



After initializing the simulation, the VI-CarRealTime solver shows an additional message reporting the file used to configure the FMI Master and the status of its initialization.

VI-CarRealTime

```

>>> run ()
=====
= VI-CarRealTime =
=====
VERSION : 20.0
REVISION: 46401_RC0//branches/20/Main/software
BUILT ON: VG13-W10/16-Jun-20
=====

Event data is being read from input XML file...

Initializing FMI Master using C:/Work/Internal/TargetProcess/17838 t
utorial review/18169 FMI External Driveline/Driveline2WD.vfm...
Done.

Performing the simulation: VI_CRT_Demo_vdd.

Initializing powertrain / driveline...
Done.

Initializing aero force...
Done.

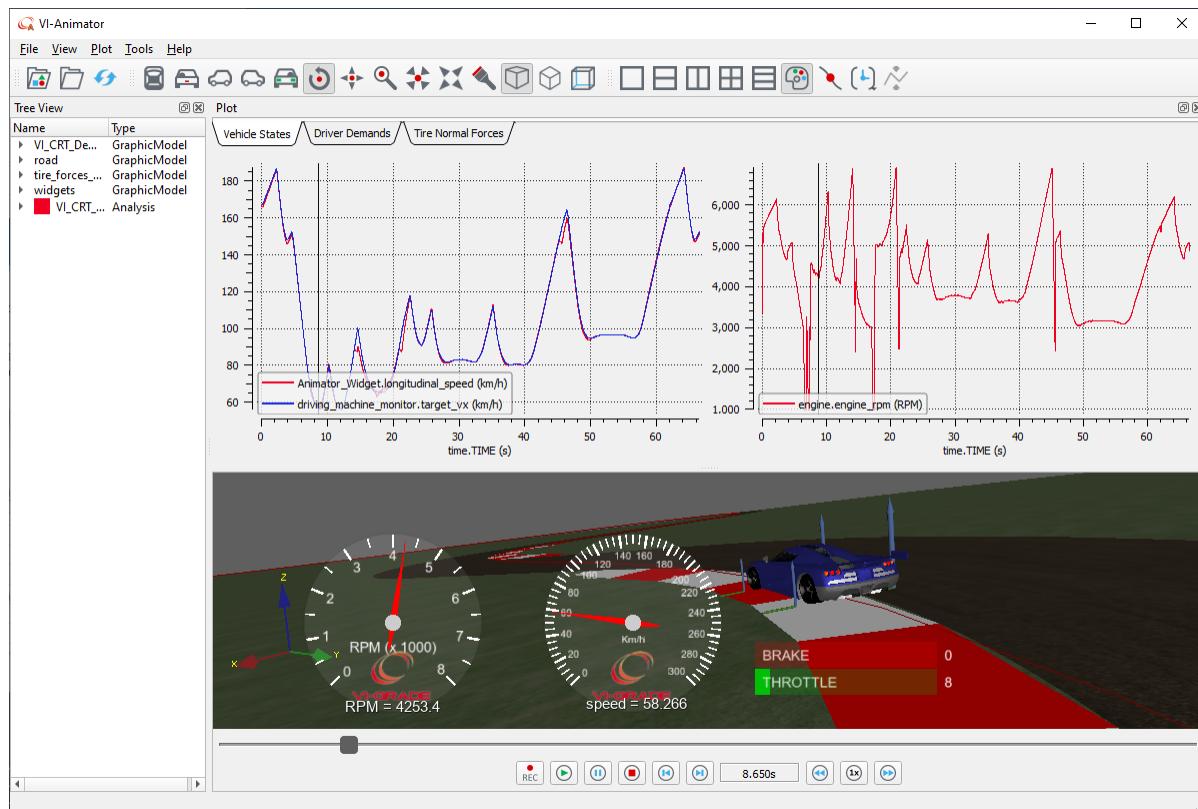
Initializing tires...
Loading road data file... (file=C:/PROGRAM~1/VI-grade/VI-CAR~3/acarrrt
/CARREA~1.CDB/roads.tbl/VI_Track.rdf)

```

At the end of the simulation, switch to **Review Mode** and select **VI_CRT_Demo_vdd** events from tree view; click button.

VI-Animator will start and animation can be played.

When a simulation is performed including FMI Master, VI-CarRealTime automatically creates a result set in the output file for each of the FMUs, containing all the output variables (Real, Integer and Boolean), defined in the FMU model description.



FMI: External Suspension

This tutorial shows how to entirely replace the rear suspension of a VI-CarRealTime vehicle model with a suspension modeled using an external tool and packed in FMU format.

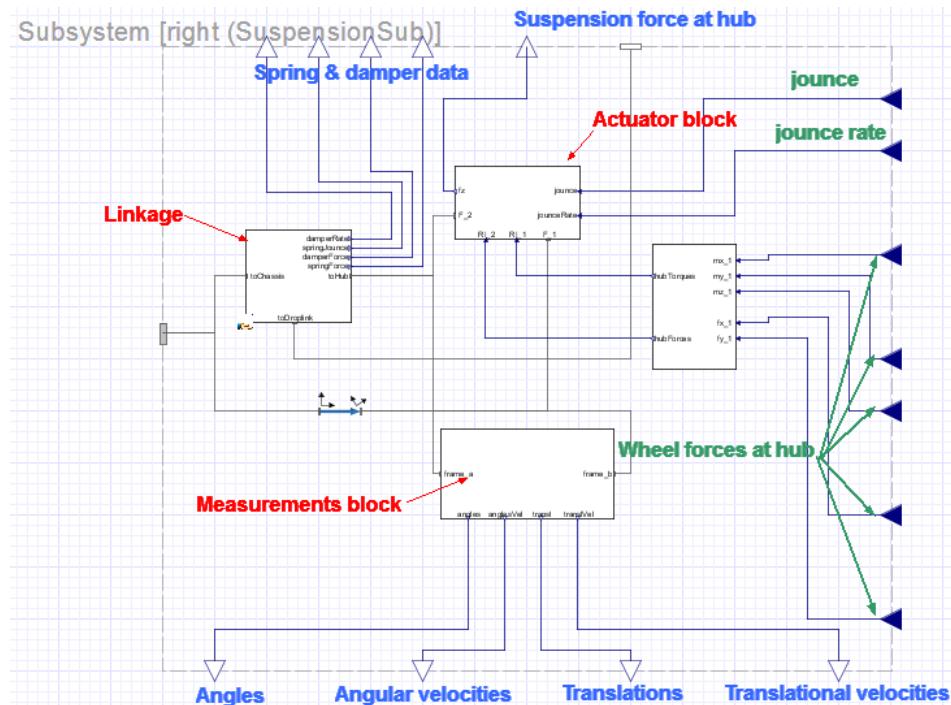
The example external suspension model consists of a double wishbone kinematic suspension with antirollbar.

The model is actuated through a motion driver that takes in input the jounce from the VI-CarRealTime model as well as the tire forces at hub (except for the normal force). The force produced by the motion driver to actuate the suspension is the resultant of a series of contributions:

- spring force at wheel
- damper force at wheel
- arb force at wheel
- kinematic effects

Gravitational and inertial effects are removed by setting the masses of the model to very low value and removing the gravitational field.

The following picture give a short overview of some of the components of the model in the original modeling tool:



The interface between the external suspension model and VI-CarRealTime vehicle model will be defined in the FMI master configuration file, and consists of the following channels:

Input (from VI-CarRealTime to the FMU suspension)

- Suspension.Jounce.L2
- Suspension.Jounce_Rate.L2
- Tire.Hub_Carrier_Moment_Local_XL2
- Brake.Brake_Moment.L2
- Tire.Hub_Carrier_Moment_Local_ZL2
- Tire.Hub_Carrier_Force_Local_XL2
- Tire.Hub_Carrier_Force_Local_YL2

- Suspension.Jounce.R2
- Suspension.Jounce_Rate.R2
- Tire.Hub_Carrier_Moment_Local_XR2
- Brake.Brake_Moment.R2
- Tire.Hub_Carrier_Moment_Local_ZR2

VI-CarRealTime

- Tire.Hub_Carrier_Force_Local_X.R2
- Tire.Hub_Carrier_Force_Local_Y.R2

Output (from FMU suspension to VI-CarRealTime)

- Suspension.L2.baseKinematicInternalFlag
- Suspension.L2.trackKinematicInternalFlag
- Suspension.L2.svaKinematicInternalFlag
- Suspension.L2.toeKinematicInternalFlag
- Suspension.L2.camberKinematicInternalFlag
- Suspension.L2.baseDisplacement
- Suspension.L2.trackDisplacement
- Suspension.L2.svaDisplacement
- Suspension.L2.camberDisplacement
- Suspension.L2.toeDisplacement
- Suspension.L2.baseVelocity
- Suspension.L2.trackVelocity
- Suspension.L2.svaVelocity
- Suspension.L2.toeVelocity
- Suspension.L2.camberVelocity
- Main_Spring.activity_flag.L2
- Main_Damper.activity_flag.L2

- Suspension.R2.trackKinematicInternalFlag
- Suspension.R2.baseKinematicInternalFlag
- Suspension.R2.svaKinematicInternalFlag
- Suspension.R2.toeKinematicInternalFlag
- Suspension.R2.camberKinematicInternalFlag
- Suspension.R2.baseDisplacement
- Suspension.R2.trackDisplacement
- Suspension.R2.svaDisplacement
- Suspension.R2.toeDisplacement
- Suspension.R2.camberDisplacement
- Suspension.R2.baseVelocity
- Suspension.R2.trackVelocity
- Suspension.R2.svaVelocity
- Suspension.R2.toeVelocity
- Suspension.R2.camberVelocity
- Main_Spring.activity_flag.R2
- Main_Damper.activity_flag.R2

- Antiroll_Bar.activity_flag.rear

- Auxiliary_vertical.force.L2
- Auxiliary_vertical.force.R2

The above list contains the output of the FMU suspension that are "connected" to the vehicle model in VI-CarRealTime. Any other output defined in the FMU model but NOT connected to the VI-CarRealTime is automatically mapped to an output channel and will be available in the results file of the simulation.

Suspension Interface Details

The external suspension returns to VI-CarRealTime the kinematics and its time derivatives. Derivatives are needed because the kinematic states are used in the vehicle model to generate the inputs for the tire forces computation, and they have a non negligible effect on the computation of tire kinematics.

As far as the suspension forces is concerned it would have been possible to apply the contribution of each single component separately to the VI-CarRealTime model, but in this example we opted for a different approach. All the suspension forces due to different contribution (all internal elements force and kinematics "anti" effects) are all measured at hub and returned to the vehicle model as auxiliary vertical force at wheel.

Note: Before starting the following operations please make a copy of the FMU archive **dw_kin.fmu** placed under **\$VI-CarRealTime install dir\acarr\examples\fmi\fmi2\\$** into the working directory.

To couple VI-CarRealTime to the driveline modeled by the FMU, the following step are required:

1. create the FMI Master configuration file

After including the header and units block, according to the [FMI Master configuration file documentation](#), the [FMI_MASTER_PARAMETERS] block has to be set as follows.

```
$-----
FMI_MASTER_PARAMETERS
[FMI_MASTER_PARAMETERS]
(FMU_TABLE)
{ fmu_block }
'FMU_1'
```

In that way, the FMI Master is configured to manage the FMU specified by the block FMU_1.
Now, add the block [FMU_1] to the configuration file,

```
$-----FMU_1
[FMU_1]
```

specifying the FMU archive full path

```
FMU_PATH='.\dw_kin.fmu'
```

the fmw instance name

```
INSTANCE_NAME='ExternalSusp'
```

the execution id

```
EXECUTION_ID=1
```

the communication step between VI-CarRealTime and the FMU

```
COMMUNICATION_STEP=1e-3
```

the log level to 'ERROR' in order to show only the error messages coming from the FMU.

```
LOG_LEVEL='ERROR'
```

Regarding the input connections map, it has to be configured writing in the first column the set of VI-CarRealTime output channels used to feed the FMU input channels placed in the second column.

The channels are identified by their names, retrievable from the [VI-CarRealTime output channels](#) list, and from the FMU model documentation:

(INPUT_CONNECTIONS_MAP)	
{vicrt_output_chl fmu_input_chl}	
'Suspension.Jounce.L2'	'jounce_left'
'Suspension.Jounce_Rate.L2'	'jounceRate_left'
'Suspension.Jounce.R2'	'jounce_right'
'Suspension.Jounce_Rate.R2'	'jounceRate_right'
'Brake.Brake_Moment.L2'	'My_left'
'Tire.Hub_Carrier_Force_Local_X.L2'	'Fx_left'
'Tire.Hub_Carrier_Force_Local_Y.L2'	'Fy_left'
'Brake.Brake_Moment.R2'	'My_right'
'Tire.Hub_Carrier_Force_Local_X.R2'	'Fx_right'
'Tire.Hub_Carrier_Force_Local_Y.R2'	'Fy_right'
'Tire.Hub_Carrier_Moment_Local_Z.L2'	'Mz_left'
'Tire.Hub_Carrier_Moment_Local_Z.R2'	'Mz_right'
'Tire.Hub_Carrier_Moment_Local_X.L2'	'Mx_left'
'Tire.Hub_Carrier_Moment_Local_X.R2'	'Mx_right'

Regarding the output connections map, it has to be configured writing in the first column the set of FMU output channels used to feed the set of VI-CarRealTime input channels placed in the second column.

The channels are identified by their names, retrievable from the [VI-CarRealTime input channels](#) list, and from the FMU model documentation::

```
(OUTPUT_CONNECTIONS_MAP)
{fmu_output_chl vicrt_input_chl}
```

```

'svaLeft'           'Suspension.L2.svaDisplacement'
'camberLeft'        'Suspension.L2.camberDisplacement'
'toeLeft'           'Suspension.L2.toeDisplacement'
'baseLeft'          'Suspension.L2.baseDisplacement'
'svaRight'          'Suspension.R2.svaDisplacement'
'camberRight'       'Suspension.R2.camberDisplacement'
'trackLeft'         'Suspension.L2.trackDisplacement'
'toeRight'          'Suspension.R2.toeDisplacement'
'baseRight'         'Suspension.R2.baseDisplacement'
'trackRight'        'Suspension.R2.trackDisplacement'
'svaVelLeft'        'Suspension.L2.svaVelocity'
'camberVelLeft'     'Suspension.L2.camberVelocity'
'toeVelLeft'        'Suspension.L2.toeVelocity'
'baseVelLeft'       'Suspension.L2.baseVelocity'
'svaVelRight'        'Suspension.R2.svaVelocity'
'camberVelRight'    'Suspension.R2.camberVelocity'
'trackVelLeft'      'Suspension.L2.trackVelocity'
'toeVelRight'       'Suspension.R2.toeVelocity'
'baseVelRight'      'Suspension.R2.baseVelocity'
'trackVelRight'     'Suspension.R2.trackVelocity'
'sva_active_right'  'Suspension.R2.svaKinematicInternalFlag'
'sva_active_left'   'Suspension.L2.svaKinematicInternalFlag'
'camber_active_right'  'Suspension.R2.camberKinematicInternalFlag'
'camber_active_left' 'Suspension.L2.camberKinematicInternalFlag'
'toe_active_left'   'Suspension.L2.toeKinematicInternalFlag'
'toe_active_right'  'Suspension.R2.toeKinematicInternalFlag'
'track_active_left' 'Suspension.L2.trackKinematicInternalFlag'
'track_active_right' 'Suspension.R2.trackKinematicInternalFlag'
'base_active_left'  'Suspension.L2.baseKinematicInternalFlag'
'base_active_right' 'Suspension.R2.baseKinematicInternalFlag'
'left_spring_active' 'Main_Spring.activity_flag.L2'
'right_spring_active' 'Main_Spring.activity_flag.R2'
'left_damper_active' 'Main_Damper.activity_flag.L2'
'right_damper_active' 'Main_Damper.activity_flag.R2'
'antirollbar_active' 'Antiroll_Bar.activity_flag.rear'
'fz_left'           'Auxiliary_vertical.force.L2'
'fz_right'          'Auxiliary_vertical.force.R2'

```

Once the connections between VI-CarRealTime and the FMU are configured it is also possible to configure the FMU parameters. The set of parameters available at FMU interface depends on the model contained in the FMU. For the model used in this example only a reduced number of parameters has been published to the FMU interface:

```

(PARAMETERS_MAP)
{fmu_parameter_name fmu_parameter_value}
'dw_kin_arbActive'      1
'dw_kin_springActive'   1
'dw_kin_damperActive'   1
'dw_kin_damperC'        15000
'dw_kin_arbK'           3437.75
'dw_kin_springK'        150000.0
'dw_kin_springPreload'  5800.0

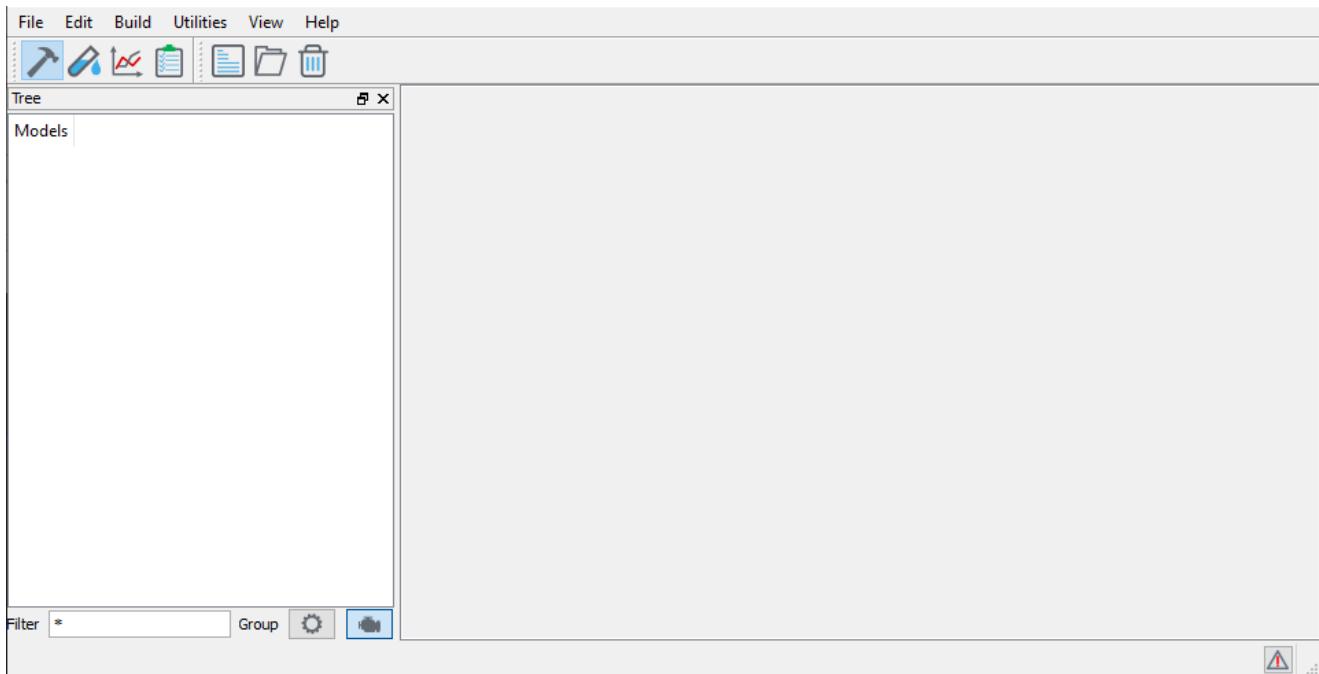
```

The FMI Master configuration file is now completed. Save the file in the working directory as **dw_kin.vfm**.

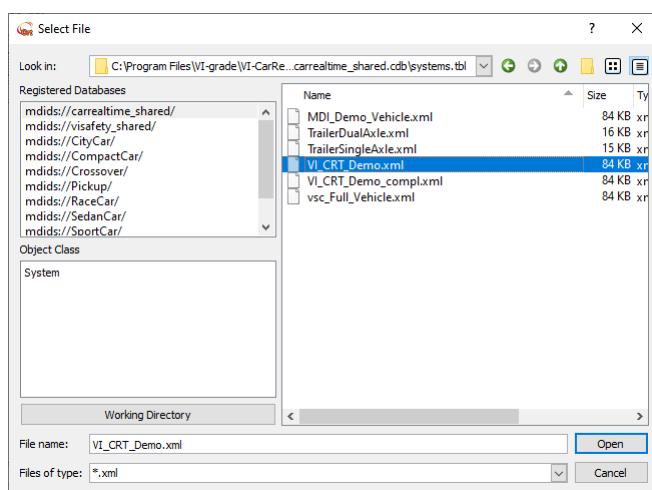
Note: An example dw_kin.vfm configuration file is also available under **\$VI-CarRealTime install dir\acarr\examples\fmi\fmi2\\$**.

2. configure the FMI Master auxiliary subsystem

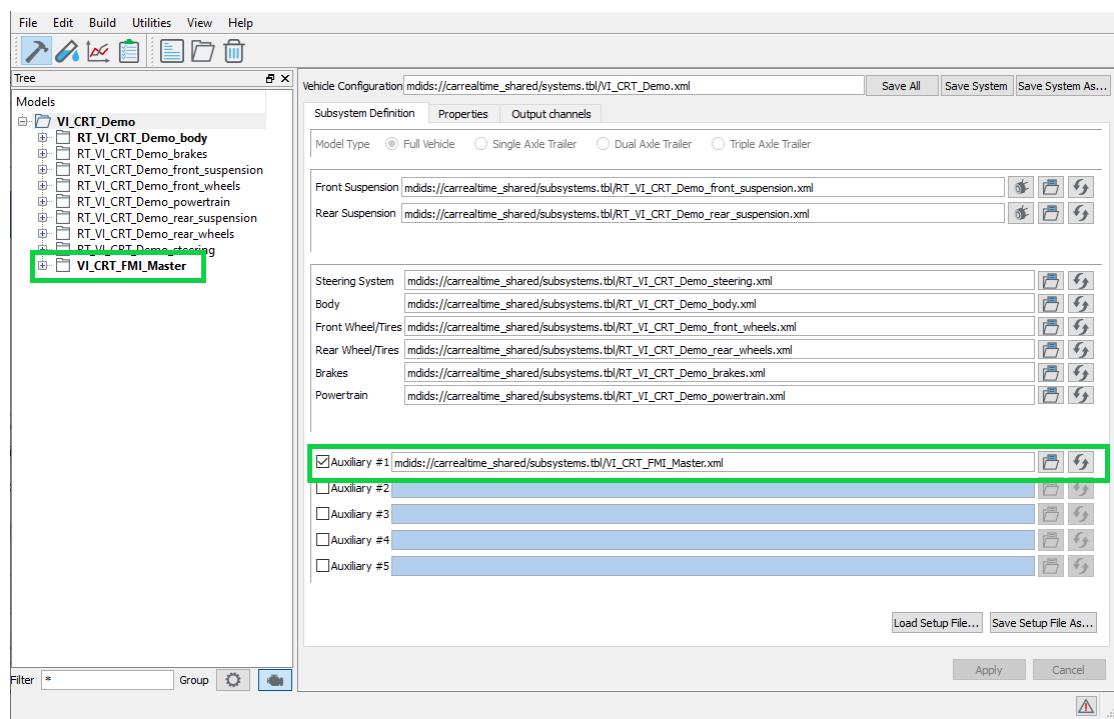
Start a new VI-CarRealTime from the current working directory typing **vicrt20** from a dos shell.



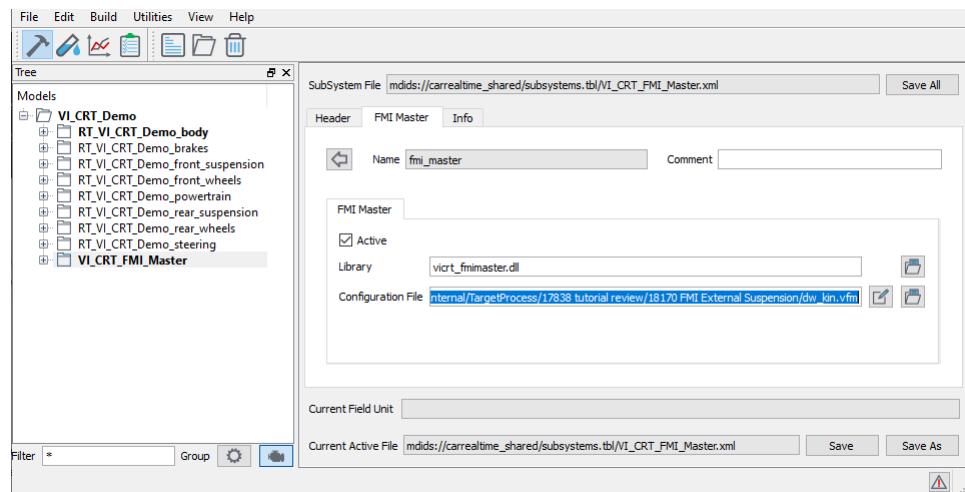
Open **VI-CRT_Demo.xml** by clicking on icon and browsing to **carrealtime_shared** database.



Enable the **Auxiliary** subsystem field, select **VI_CRT_FMI_Master** auxiliary subsystem from the **carrealtime_shared** database by clicking on and apply the modification.



Edit the `VI_CRT_FMI_Master` subsystem from the model tree, set the FMI Master tab, enable the **Active** flag, and select the configuration file created previously.

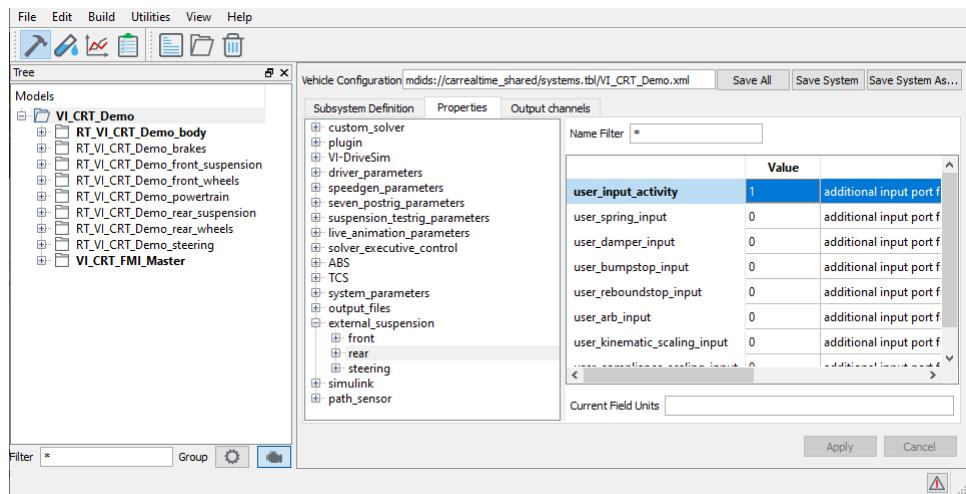


3. enable external suspension input

Before running the simulation, the external suspension input for both kinematic and compliance must be activated in the system tree.

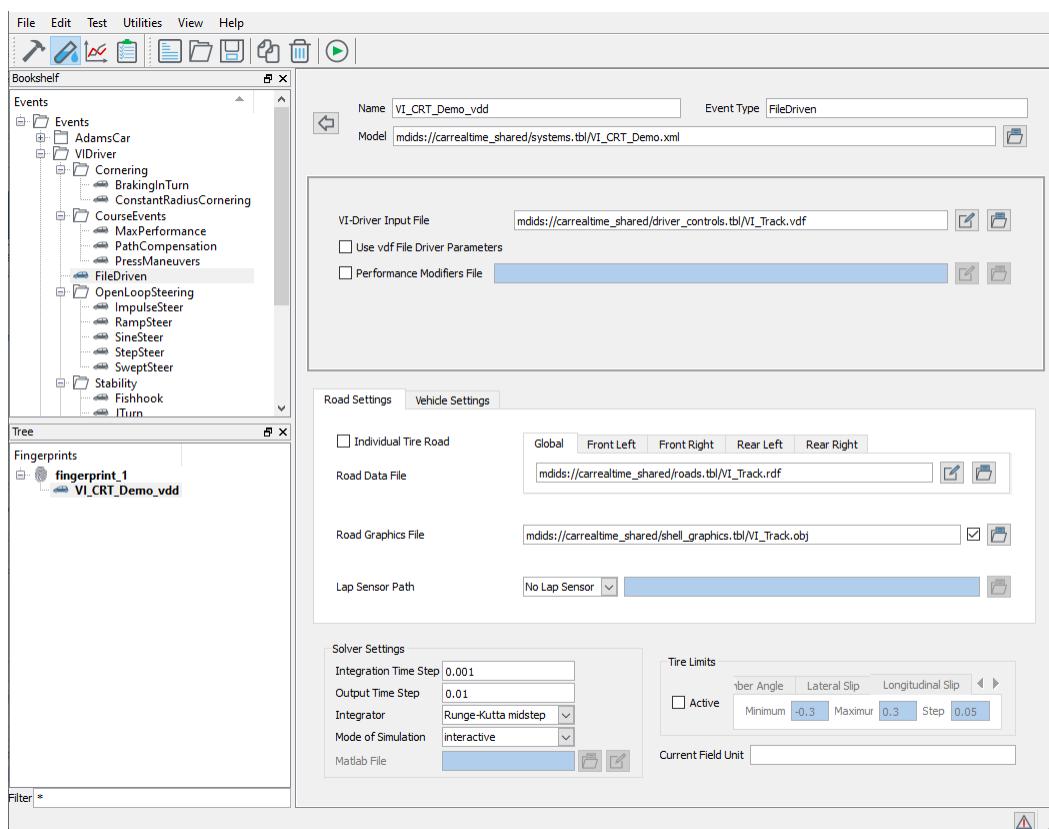
Select the `VI_CRT_Demo` system, set to 1 the `user_input_activity` flag related to the rear suspension from the property tree (`external_suspension>rear`) and **Apply** the modification.

VI-CarRealTime

**4. run the simulation**

Running a simulation using the FMI Master doesn't present any difference with respect to a VI-CarRealTime event performed with any other model.

Create a new event using the VI_CRT_Demo vehicle, select the VI-Driver Input File VI_Track.vdf, the Road Data File VI_Track.rdf and click on the button to start the simulation.

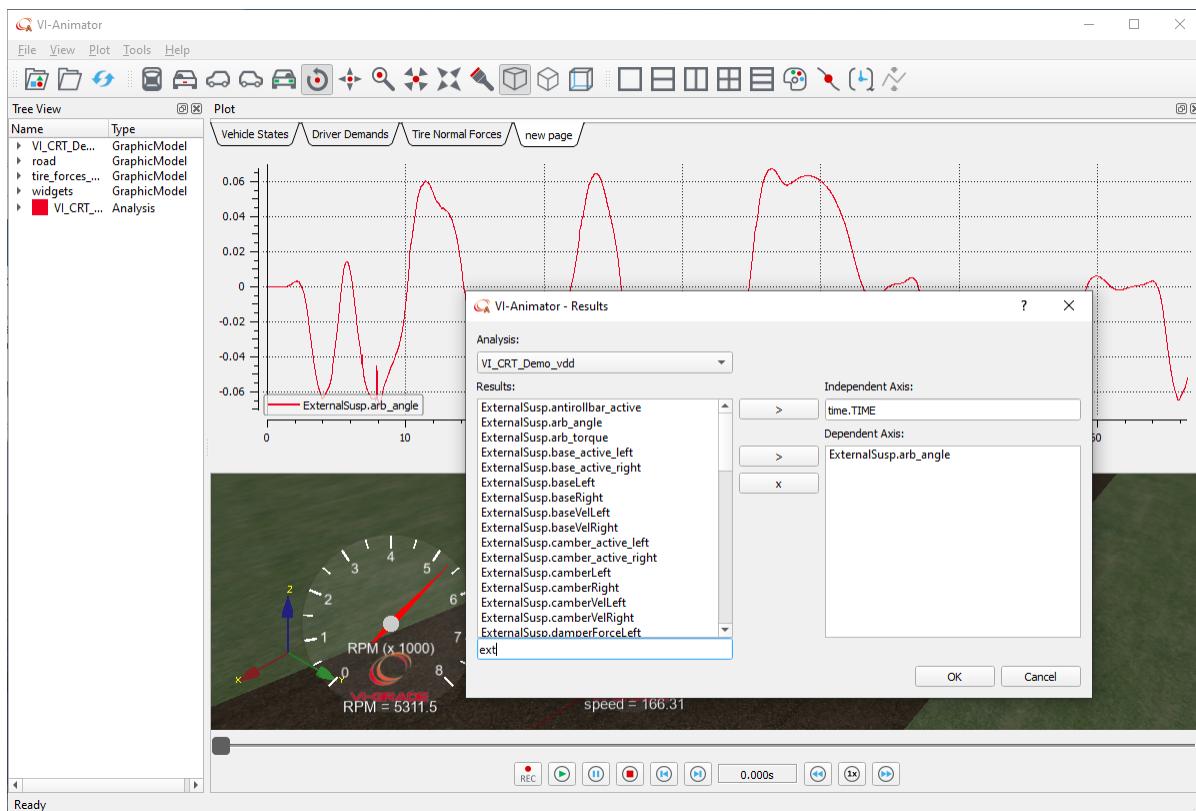


After initializing the simulation, the VI-CarRealTime solver shows an additional message reporting the file used to configure the FMI Master and the status of its initialization.

At the end of the simulation, switch to **Review** mode and select **VI_CRT_Demo_vdd** events from tree view; click button.

VI-Animator will start and animation can be played.

When a simulation is performed including FMI Master, VI-CarRealTime automatically creates a result set in the output file for each of the FMUs, containing all the output variables (Real, Integer and Boolean), defined in the FMU model description.



1.4.4 VI-CarRealTime Customization

In the following tutorial you will learn how to create custom version of VI-CarRealTime.

[Creating Solver Plugin](#)

[Creating Custom Tire Library](#)

[Creating Custom Aerodynamics Force Library](#)

[Creating Custom Event](#)

Creating Solver Plugin

In this tutorial you'll learn how to build and run a customized VI-CarRealTime solver plugin library.

To run this tutorial you need to copy the files `simplePowertrain.cpp`, `simplePowertrain.hpp` and `solverPlugin_powertrain.cpp` from the directory `$VI-CarRealTime$Installationdir\acarrt\examples\SolverPlugin` to an empty directory of your choice, that we will call `$WorkingDir` from now on.

Note: To successfully perform this tutorial you need the Microsoft Visual C++ .NET (Microsoft Visual Studio .NET 2017) installed on your computer.

This tutorial is composed by the following sections:

- [Building the Custom Library](#)

VI-CarRealTime

- [Running Driving Machine event with Custom Solver \(Standalone\)](#)
- [Running Driving Machine event with Custom Solver \(Matlab Interface\)](#)

Note: please refer to [Environment Variable](#) topic for an overview about the environment variables that can be set for the building process.

Building Custom Library

Building the Custom Library

- Open a command prompt shell of your compiler and change directory to **\$WorkingDir**.
- Type the following command:
vig20 acreal cr-plugin
The VI-CarRealTime selection menu will appear and will start the plugin creation wizard.
- First of all you will have to choose if you want to perform a debug link:
Would you like to link in Debug mode? (CR=n) or EXIT:
Press "**Enter**", to accept the default.
- You are now asked to enter the object file, source file, or source list file to be included in the custom library:
Enter name of first object or source file or EXIT:
Type each of the example source file names (*simplePowertrain.cpp*, *solverPlugin_powertrain.cpp*) followed by "**Enter**"
- The wizard asks for additional source files or for a carriage return if no more sources/objects are needed.
Enter name of next object or source file (<CR>=none), or EXIT:
Press "**Enter**" to end the list.
- You now have to specify the name of the custom shared library:
Enter name of your VI-CRT Solver Plugin DLL or EXIT::
Type **simplePowertrainPlugin.dll**.
The procedure will now proceed automatically and the custom plugin library "*simplePowertrainPlugin.dll*" will be created in **\$WorkingDir**.

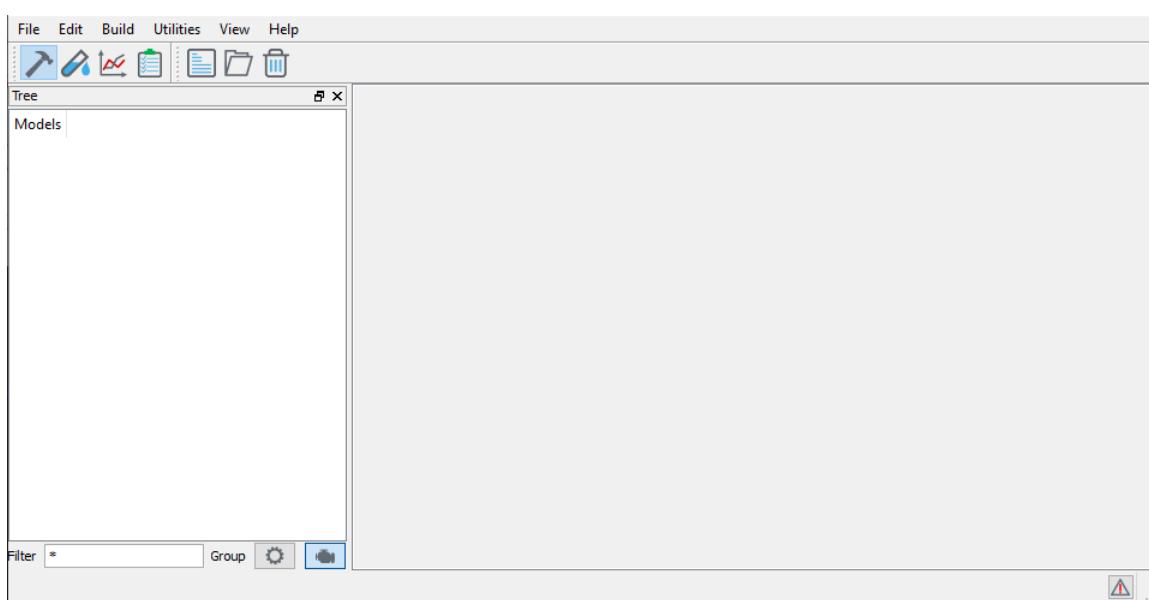
Note: Instead of going interactively through the wizard steps, you can specify all the parameters in one command line and the procedure will be automatically executed. Remember that the end of the source and objects list is identified by the -n separator. In our example the full command line would be:

```
vig20 acreal cr-plugin n simplePowertrain.cpp solverPlugin_powertrain.cpp -n  
simplePowertrainPlugin.dll
```

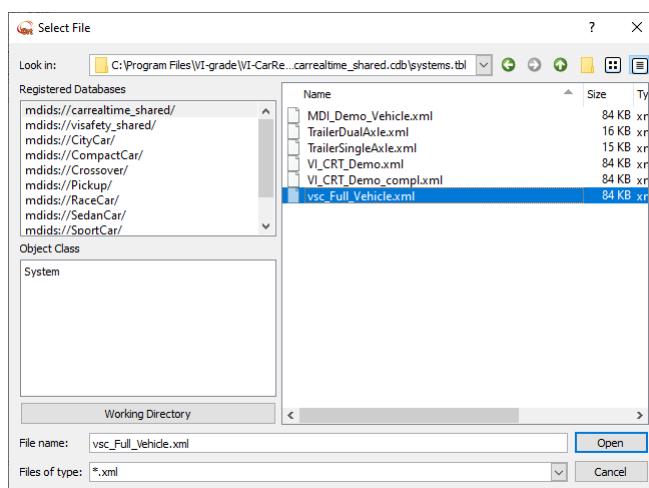
Running Event With Custom Solver (Standalone)

Note: in order to perform this tutorial, the tutorial [Building Custom Library](#) must have been completed.

Start a new VI-CarRealTime session in your **\$WorkingDir** typing **vicrt20** from a dos shell.



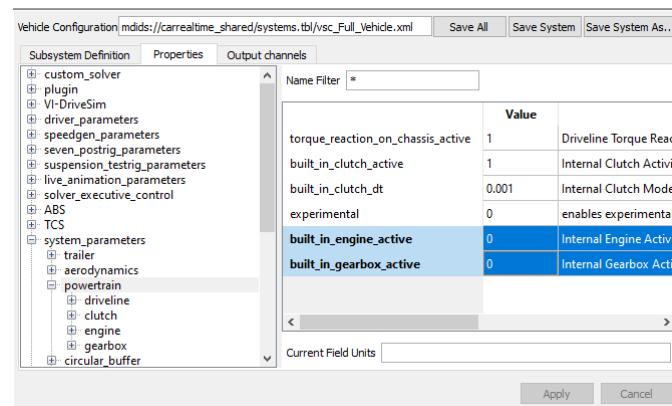
Open **vsc_Full_Vehicle.xml** by clicking on icon and browsing to **carrealtime_shared** database



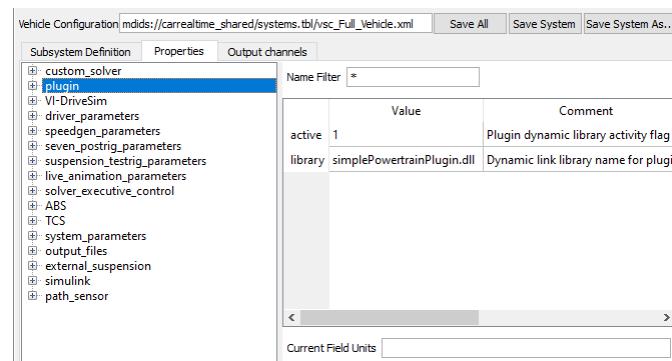
The custom solver plugin dll we've just created contains a simple engine and transmission model. In order to correctly use the custom powertrain model in place of the built-in model, we need to "deactivate" the internal model. This can be done by setting to "0" one or more of the flags available in the [System Properties Tab](#), at the section **system_parameters>powertrain**.

For this simple example we will deactivate only *built-in engine* and *gearbox*, leaving the *built-in clutch* model active.

Press **Apply** button to set the changes in the model.



Custom solver plugin can be accessed running VI-CarRealTime normally after specifying solver plugin library name and activity at the section Plugin:



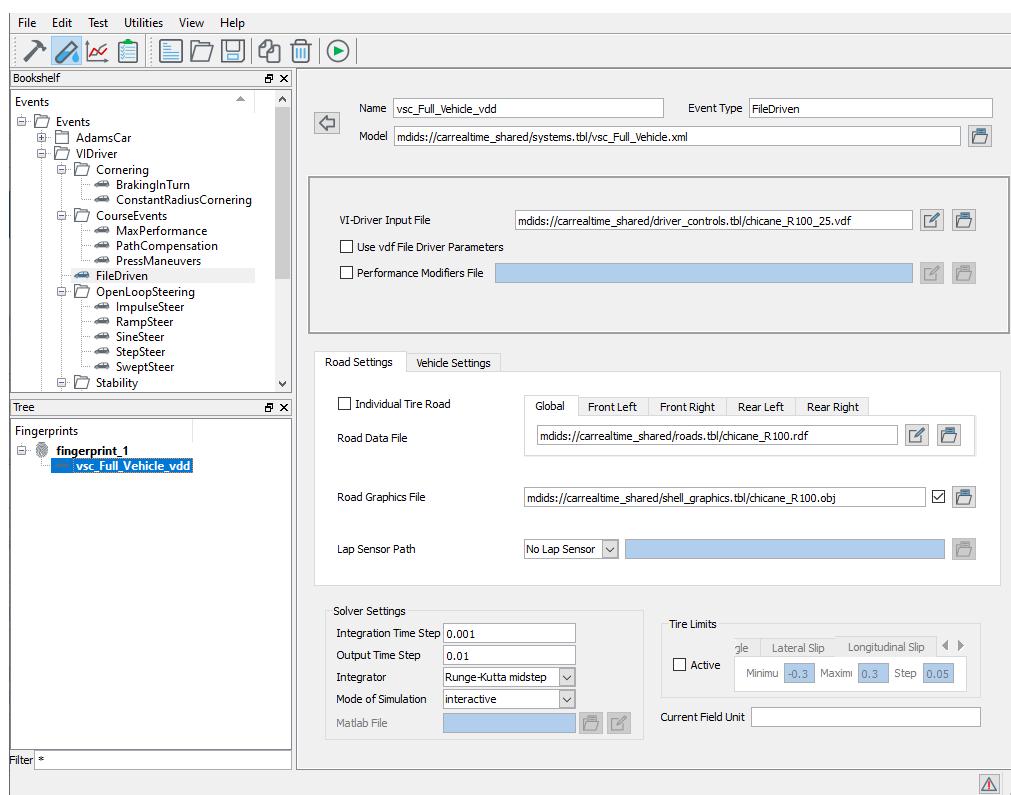
Note: the custom solver plugin can be also accessed running VI-CarRealTime solver from the command line, and adding the **-e** switch to specify the name of the dll. In such case the dll specified in the command line will override the one set in the System tree.

For example, to run the model with the just created custom plugin, you need to start the simulation from the dos prompt, as follows:

```
$VI-CarRealTimeInstallationdir\carrt\win64\vicrt.exe -f <input_data_file.xml> -e simplePowertrainPlugin.dll
```

where `<input_data_file.xml>` is the name of the `*_send.xml` input file generated by the VI-CarRealTime GUI when running a dynamic event.

Switch to **Test Mode**  and submit a **File Driven** event, as shown in the picture below:



Run the event by clicking button.

```
VI.PTW (VI-CarRealTime Python Task Window)
-- INFO -- Maneuver 1 (MANEUVER_1) end of path reached
...
-- INFO -- (maneuver time = 20.88s / simulation time =
20.88s)
-- INFO --
-- INFO --
-- INFO --
-- INFO -- <<< VI-Driver MANEUVER INFO >>>
-- INFO -- == LAST MANEUVER COMPLETED ==
-- INFO -- == DRIVER HAS BEEN DEACTIVATED ==
-- INFO --
-- INFO --

A Driving-Machine end condition was triggered: simulation
terminated prior to simulation abort time.

=====
= VI-CarRealtime =
= VEHICLE SIMULATION STATISTICS =
=====

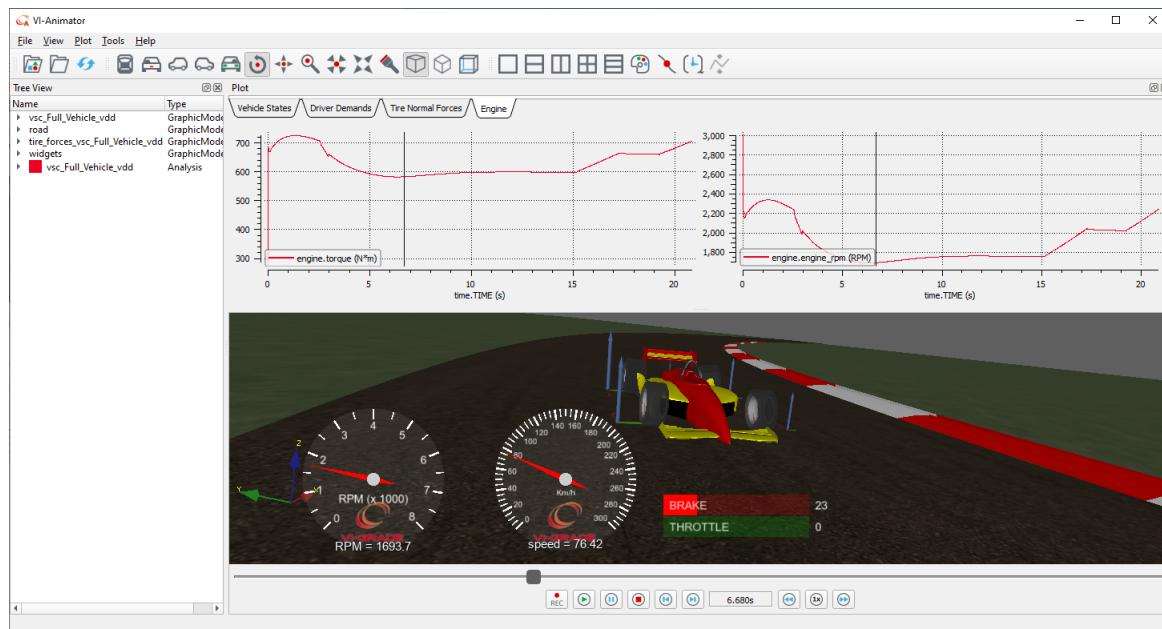
--- Simulation Statistics ---
>> LAP TIME      = 20.881000 sec
>> LAT ACCEL MAX = 0.645981 G
>> LAT ACCEL MIN = -0.511852 G
>> LON ACCEL MAX = 0.849252 G
>> LON ACCEL MIN = -0.677655 G
>> LON SPEED MAX = 104.903583 km/h
>> LON SPEED MIN = 76.363985 km/h
>> LON SPEED AVG = 85.399740 km/h
>> STEERING MAX = 0.680550 rad
>> STEERING MIN = -0.647891 rad

Writing output to file. Please Wait ...

Elapsed Simulation Time: 20.881 (sec).
Elapsed Clock Time: 3.379 (sec).
Computational Efficiency: 6.180 (sim. sec)/(sec).

Simulation is complete.
```

Using VI-Animator it is possible to review analysis results in terms of plots and animation.



Running Event with Custom Solver (Matlab)

The custom solver plugin created in [Building Custom Library](#) tutorial can be also used in conjunction with the VI-CarRealTime [Matlab Interface](#). In order to correctly use the custom powertrain model in place of the built-in model, we need to "deactivate" the internal (VI-CarRealTime) model.

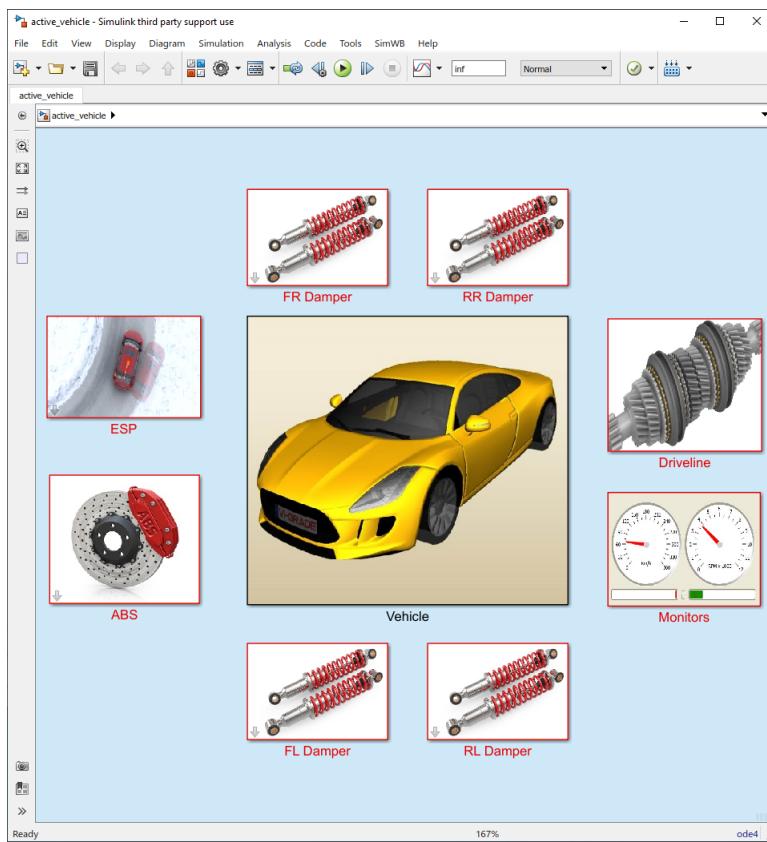
Note: in order to perform this tutorial, the tutorials [Building Custom Library](#) and [Running Event With Custom Solver \(Standalone\)](#) must have been completed.

Open a MATLAB session and change the working directory in order to match with [\\$WorkingDir](#).

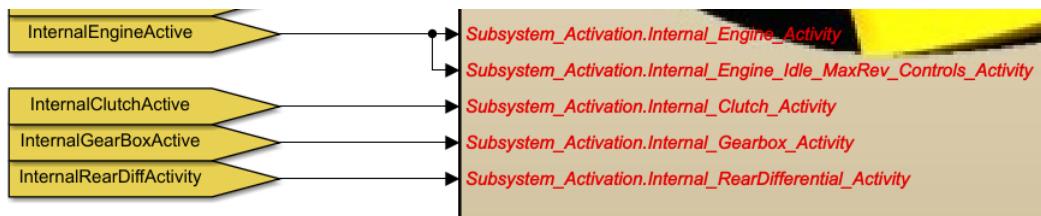
Type the following command in MATLAB Command Window in order to add the paths needed to retrieve VI-CarRealTime stuff:

```
Command Window
>> addpath_vicrt_20
fx >> |
```

Open the `$VI-CarRealTime$Installationdir\acarrt\examples\Simulink\active_vehicle.mdl` model.



Open the **Vehicle** subsystem and then the **VI-CarRealTime** block. The following inputs will be set to 1 as default.



In this way the internal (VI-CarRealTime) clutch engine and gearbox will be used; but we want to use the engine and gearbox modeled externally in the plug-in dll. In order to do this we have to tell VI-CarRealTime solver to not use the internal (VI-CarRealTime) engine and gearbox but the external (plug-in dll) ones.

We can feed the solver with this information through the plug-in dll, so we must implement this in the source files. Open `solverPlugin_powertrain.cpp` stored in the [\\$WorkingDir](#) and add the following lines (highlighted) in the `extcall_set_input` function:

```

67     PUBLIC EXPORT int extcall_set_input( double *inputvec, int inputvecSize )
68     // INPUT
69     //   double* inputvec: pointer to the VI-CarRealTime input array, as described
70     //     in svm_ii_FV_io_map.h and
71     //     int inputvecSize: size of the input array
72     // OUTPUT
73     //   int status: if 0 error else success
74
75     static double *outputArray = NULL;
76     if (NULL == outputArray)
77         outputArray = new double[evOutputDIM];
78
79     // perform powertrain computation
80     pwr.Compute();
81     // retrieve outputs
82     pwr.GetOutput(outputArray);
83     inputvec[crtInputIDs[evActualTorque]] = outputArray[evActualTorque];
84     inputvec[crtInputIDs[evMinTorque]] = outputArray[evMinTorque];
85     inputvec[crtInputIDs[evMaxTorque]] = outputArray[evMaxTorque];
86     inputvec[crtInputIDs[evEngineOmega]] = outputArray[evEngineOmega];
87     inputvec[crtInputIDs[evTransmissionRatio]] = outputArray[evTransmissionRatio];
88
89     // set outputs for differential torques to 0.0 (OPEN differential)
90     // This input is only used to return the torque split due to differential with respect
91     // to an open diff (50% split)
92     // inputvec[crtDifferentialInputIDs[0]] = 0;
93     // inputvec[crtDifferentialInputIDs[1]] = 0;
94     inputvec[INPUT_FV_Subsystem_Activation_Internal_Engine_Activity] = 0;
95     inputvec[INPUT_FV_Subsystem_Activation_Internal_Gearbox_Activity] = 0;
96
97
98     return 1;
99 }

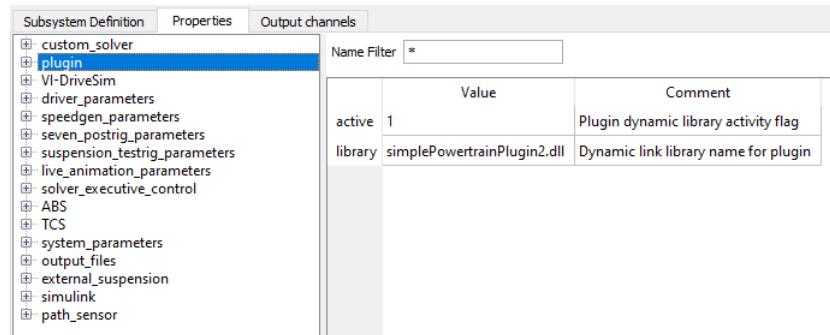
```

The name of these inputs can be retrieved in the `svm_ii_FV_io_map.h` stored in **\$VI-CarRealTime\Installationdir\acarr\user_api** folder.

Save the modification and repeat the [procedure](#) for creating the new dll (rename it **simplePowertrainPlugin2.dll**).

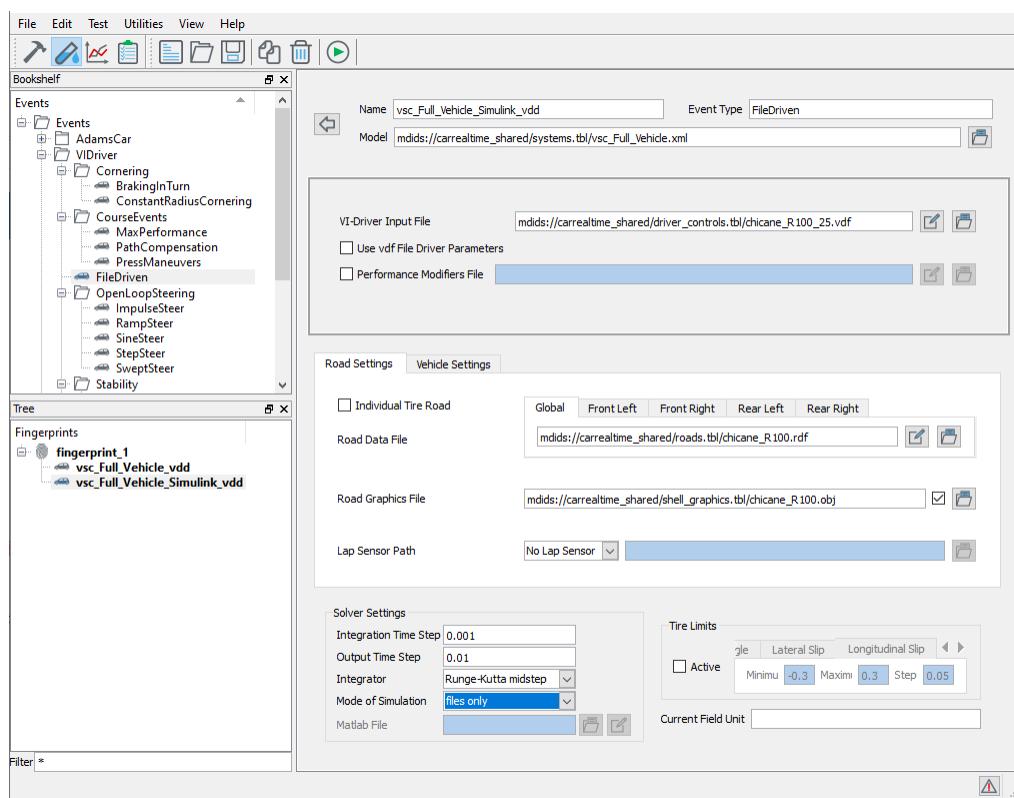
Switch to VI-CarRealTime Interface, and browse to **Build Mode**.

Select **vsc_Full_Vehicle** assembly, browse to **Properties** panel and change the **library** value in order to point to the new dll (**simplePowertrainPlugin2.dll**).



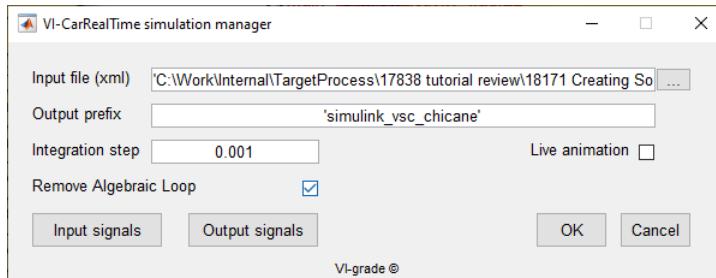
Switch to **Test Mode** ().

Copy the **vsc_Full_Vehicle_vdd** event and rename it **vsc_Full_Vehicle_simulink_vdd**.



Set **files only** mode in **Mode of Simulation** and run (▶) the event.

Switch back to **Simulink** Interface, open the **Vehicle** subsystem and then double click on **VI-CarRealTime** mask and set the **vsc_Full_Vehicle_simulink_vdd_send_svm.xml** file stored in **\$WorkingDir** in **Input File (xml)** field:



Type **simulink_vsc_chicane** in **Output prefix** field and press **OK** button to save the changes.
Run the co-simulation by pressing **Play** button in Simulink interface.

Note: the name of the solver plugin dll will be retrieved by send xml file.

Open VI-Animator and load **simulink_vsc_chicane.res** file to review the analysis results.



Creating Custom Tire Library

In this tutorial you'll learn how to build and run a customized VI-CarRealTime Tire Library.

To run this tutorial you need to copy the files in the directory **\$VI-CarRealTimeInstallationdir\acarr\examples\Custom_Tire** to an empty directory of your choice, that we will call **\$WorkingDir** from now on.

Note: To successfully perform this tutorial you need the Microsoft Visual Studio 2017 (C++) installed on your computer.

This tutorial is composed by the following sections:

- [Building the Custom Tire Library](#)
- [Running Driving Machine event with Custom Tire Model](#)

Note: please refer to [Environment Variable](#) topic for an overview about the environment variables that can be set for the building process.

Building Custom Tire Library

Building the Custom Tire Library

In the Custom_Tire folder two different tire interfaces can be found:

- **VI-Tire** interface (*tire_cpp_crt_user.cxx*);

- **STI** interface (*tire_c_sti_user.c*).

The following steps help the user to build up a custom tire dll with VI-Tire interface:

- Open a command prompt shell of your compiler and change directory to **\$WorkingDir**.
- Type the following command:
vig20 acreal cr-tire
The VI-CarRealTime selection menu will appear and will start the tire dynamic library creation wizard.
- First of all you will have to choose if you want to perform a debug link:
Would you like to link in Debug mode? (CR=n) or EXIT:
Press "**Enter**", to accept the default.
- You are now asked to enter the object file, source file, or source list file to be included in the custom library:
Enter name of first object or source file or EXIT:
Type **tire_cpp_crt_user.cxx** the name of the sample source file, containing a simplified tire model (vertical force only)
- The wizard asks for additional source files or for a carriage return if no more sources/objects are needed.
Enter name of next object or source file (<CR>=none), or EXIT:
Press "**Enter**" to end the list.
- You now have to specify the name of the custom executable:
Enter name of your VI-CRT UserTire-DLL or EXIT:
Type **mytire_crt.dll**.
The procedure will now proceed automatically and the custom tire library "mytire_crt.dll" will be created in **\$WorkingDir**.

The following steps tell the user how to build up a custom tire dll by using STI interface:

- Open a dos prompt and change directory to **\$WorkingDir**.
- Type the following command:
vig20 acreal
The VI-CarRealTime selection menu will appear.
- Select the Option **cr-tire**, to create a custom VI-CarRealTime tire library.
This will start the tire dynamic library creation wizard.
- First of all you will have to choose if you want to perform a debug link:
Would you like to link in Debug mode? (CR=n) or EXIT:
Press "**Enter**", to accept the default.
- You are now asked to enter the object file, source file, or source list file to be included in the custom library:
Enter name of first object or source file or EXIT:
Type **tire_c_sti_user.c** the name of the sample source file, containing a simplified tire model (vertical force only)
- The wizard asks for additional source files or for a carriage return if no more sources/objects are needed.
Enter name of next object or source file (<CR>=none), or EXIT:
Press "**Enter**" to end the list.
- You now have to specify the name of the custom executable:
Enter name of your VI-CRT UserTire-DLL or EXIT:
Type **mytire.dll**.
The procedure will now proceed automatically and the custom tire library "my_tire.dll" will be created in **\$WorkingDir**.

Note: Instead of going interactively through the wizard steps, you can specify all the parameters in one command line and the procedure will be automatically executed.

Remember that the end of the source and objects list is identified by the -n separator. In our example the full command line would be:

```
vig20 acreal cr-tire n <source_file> -n <dll_name>
```

Running Event With Custom Tire Model

Running Driving Machine event with Custom Tire Model

The custom tire we've just created contains a simple tire model, vertical only. The mechanism used in VI-CarRealTime to select the user tire model instead of the internal model is based on the *tire model* selection in the tire property file.

A tire property file has a global section at the beginning with information on the file type, units and model. The remaining portion of the files includes the tire parameter mapping, and it is strongly dependent on the tire model.

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE      = 'tir'
FILE_VERSION   = 7.1
FILE_FORMAT    = 'ASCII'
$-----UNITS
[UNITS]
LENGTH        = 'meter'
ANGLE         = 'radian'
FORCE          = 'newton'
MASS           = 'kg'
TIME           = 'second'
$-----MODEL
[MODEL]
PROPERTY_FILE_FORMAT = 'USER'
FUNCTION_NAME = 'my_tire::crtusertire'
INTERFACE_TYPE = 'CRT'
...
...
```

In order to use the custom tire model you need to specify the **PROPERTY_FILE_FORMAT (USER)** and the name of the **custom tire library (dll)** and the **function** where the custom algorithm is implemented. As an example, for the *mytire.dll* created in the [Building Custom Tire Library](#) tutorial, the **MODEL** block of the tire property file is the following:

```
[MODEL]
PROPERTY_FILE_FORMAT = 'USER'
FUNCTION_NAME        = 'mytire::tire_cpp_crt_user'
INTERFACE_TYPE       = 'CRT'
```

where:

mytire is the name of the dynamic link library (*mytire.dll*);

tire_cpp_crt_user is the name of the interface function of VI-Tire model, defined in the *tire_cpp_crt_user.cxx* source file.

The key **interface type** defines which calling interface the custom subroutine expects: supported values are STI and CRT.

You can specify the tire property file name in the model as explained in [Wheel Subsystem](#). Once you've specified the correct property file in the model, you can run a [Driving Machine](#) event.

Note: Make sure that the custom tire model library is in the *working directory* or in the *system search path*.

Creating Custom Aerodynamics Force Library

In this tutorial you'll learn how to build and run a customized VI-CarRealTime Aerodynamic Forces Library.

To run this tutorial you need to copy the files in the directory **\$VI-CarRealTime\Installationdir\acarrt\examples\Custom_Aerodynamics** to an empty directory of your choice, that we will call **\$WorkingDir** from now on.

Note: To successfully perform this tutorial you need the Microsoft Visual Studio 2017 (C++) installed on your computer.

This tutorial is composed by the following sections:

- [Building the Custom Aerodynamics Force Library](#)
- [Running Driving Machine event with Custom Aerodynamic Forces Model](#)

Note: please refer to [Environment Variable](#) topic for an overview about the environment variables that can be set for the building process.

Building Custom Aerodynamics Force Library

Building the Custom Aerodynamics Force Library

- Open a command prompt shell of your compiler and change directory to **\$WorkingDir**.
- Type the following command:
vig20 acreal cr-aero
The VI-CarRealTime selection menu will appear and will start the aerodynamic forces dynamic library creation wizard.
- First of all you will have to choose if you want to perform a debug link:
Would you like to link in Debug mode? (CR=n) or EXIT:
Press "**Enter**", to accept the default.
- You are now asked to enter the object file, source file, or source list file to be included in the custom library:
Enter name of first object or source file or EXIT:
There are two example files provided as example in VI-CarRealTime (one is written using C language, the other using C++ language). In this tutorial we will use the C++ language sample file (**aero_cpp_user.cxx**), but all the steps apply also to the C language.
Type the name of the source file **aero_cpp_user.cxx**, containing a simplified aerodynamic forces model.
- The wizard asks for additional source files or for a carriage return if no more sources/objects are needed.
Enter name of next object or source file (<CR>=none), or EXIT:

Press "**Enter**" to end the list.

- You now have to specify the name of the custom executable:

Enter name of your VI-CRT UserAeroForce-DLL or EXIT:

Type **user_aero.dll**.

The procedure will now proceed automatically and the custom aerodynamic forces library "user_aero.dll" will be created in **\$WorkingDir**.

Note: Instead of going interactively through the wizard steps, you can specify all the parameters in one command line and the procedure will be automatically executed.

Remember that the end of the source and objects list is identified by the -n separator. In our example the full command line would be:

```
vig20 acreal cr-aero n aero_cpp_user.hxx -n user_aero.dll
```

Running Event With Custom Aero Forces Model

Running Driving Machine event with Custom Aerodynamic Forces Model

The custom aerodynamic forces library we just created contains a simple model. The mechanism used in VI-CarRealTime to select the user aerodynamic forces model instead of the internal model is based on the model selection in the aerodynamic forces property file (*.aer). An aerodynamic forces property file has a global section at the beginning with information on the file type, units and model. The remaining portion of the files includes the tire parameter mapping, and it is strongly dependent on the tire model.

```
$-----MDI_HEADER
[VIGRADE_HEADER]
FILE_TYPE      = 'aer'
FILE_VERSION   = 1.0
FILE_FORMAT    = 'ASCII'
(COMMENTS)
{comment_string}
'VI-grade user aerodynamics sample data file'
$-----UNITS
[UNITS]
LENGTH          = 'FOOT'
FORCE           = 'POUND'
ANGLE           = 'RADIAN'
MASS            = 'LBF'
TIME            = 'SECOND'
$-----MODEL
[MODEL]
PROPERTY_FILE_FORMAT = 'USER'
FUNCTION_NAME     = 'user_aero::aero_cpp_usersample'
$-----USER_DEFINED
[USER_DEFINED]
(COMMENTS)
{comment_string}
'Place here parameters for the user defined model'
```

In order to use the custom aerodynamic forces model you need to specify the **PROPERTY_FILE_FORMAT (USER)** and the name of the custom library and function where the custom algorithm is implemented. An example file to test the newly created custom library is available in the **VI-CarRealTime** shared database (user_aero.aer).

You can specify the aerodynamic forces property file name in the model as explained in [Body Subsystem](#). Once you've specified the correct property file in the model, you can run a [Driving Machine](#) event.

Note: Make sure that the custom aerodynamic forces model library is in the working directory or in the system search path.

Creating Custom Event

The aim of the present tutorial is to show the user how to use the environment to set up, modify and run a custom event.

The tutorial is composed by four sections:

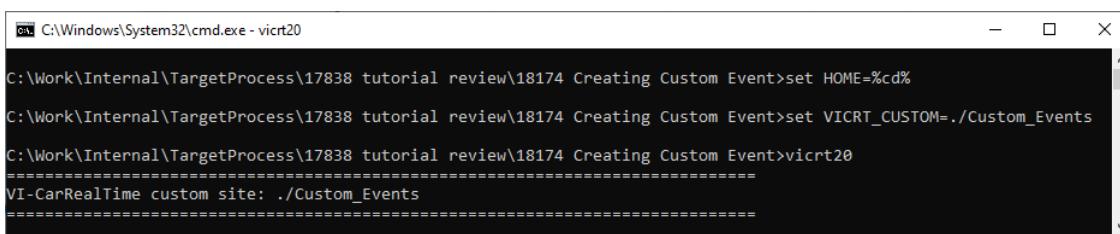
- [Build Up Environment](#)
- [Add New Event](#)
- [Run Custom Event](#)
- [Review Results](#)

Build Up Environment

Before opening a new VI-CarRealTime session, the custom environment has to be set up.

In order to do this, the [VICRT_CUSTOM](#) environment variable must be set to the [root_folder](#) of the environment.

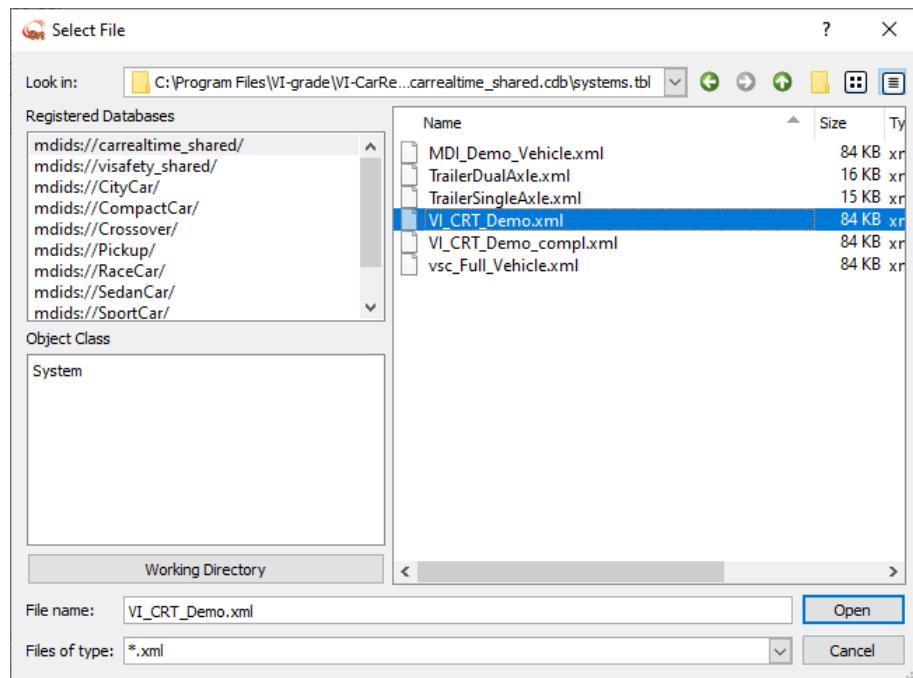
- Copy the Custom_Events folder located in <installation_dir>/acarrt/examples/Custom_Events in your working directory.
- Open a new dos shell and browse to your working folder;
- Set the environment variable to the environment root_folder (for this tutorial the root_folder name is Custom_Events and it is stored in the working directory);
- run VI-CarRealTime by typing *vicrt20*;



```
C:\Windows\System32\cmd.exe - vicrt20
C:\Work\Internal\TargetProcess\17838 tutorial review\18174 Creating Custom Event>set HOME=%cd%
C:\Work\Internal\TargetProcess\17838 tutorial review\18174 Creating Custom Event>set VICRT_CUSTOM=./Custom_Events
C:\Work\Internal\TargetProcess\17838 tutorial review\18174 Creating Custom Event>vicrt20
=====
VI-CarRealTime custom site: ./Custom_Events
=====
```

A message showing that a custom event module has been found is displayed in the shell.

- In the VI-CarRealTime main GUI, select **Build Mode**, open **VI-CRT_Demo.xml** by clicking on  icon and browsing to **carrealtime_shared** database:

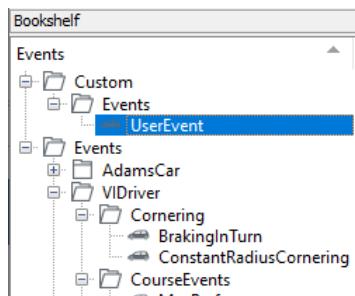


- Switch to **Test Mode** .

The event parameter tree will be composed by two main folders:

- **Events**
folder embedding the standard events;
- **Custom(*)**
folder embedding the custom events.

* : this is the name of the folder under events folder; its name can be changed by the user.



The name of the event is the name of the class defined in the python file under `Custom_Events/events/Custom/events`.

Add New Event

The next step consists in adding a new custom event in the parameter tree.

The event we are going to create consists in a pre-analysis and a main analysis: the pre-analysis is a step steer which computes the steering angle which gives a preset lateral acceleration of the chassis while the main analysis is a sine steer having a preset frequency and the steering amplitude previously computed.

The steps to add the new event are:

- create a python file (`my_sine_event.py`) embedding a new [event class](#) and store the file in the `Custom_Events/events/Custom/Events` folder.

```
█ my_sine_event.py
# Basic modules import
import sys,os,re,qt,string
from   math           import *
from   types          import *
from   mdi.afc.afckernel import Enum, Class
from   mdi.afc.afckernel import AfcKernel,Bibliography,Enum,Class,BoolType,Point
from   mdi.afc.afckernel import Units as AfcUnits
from   mdi.afc.afckernel import UnitSetting,UnitValue,UnitDimension,UnitSystem,UnitConv
from   mdi.afc          import vfc
from   mdi.visedit import vfc_files
import copy,types
import svm_event
# Other modules import
from vigrade import vicrt_event_utils
# Define local variables

DTOR = pi/180.0

if sys.version_info[0] > 2:
    ClassType = type
    ListType = list
    TupleType = tuple
    StringType = str
    FloatType = float
    IntType = int

class MySineEvent(svm_event.SVMEVENT):

    def __init__(self,parent):
        """
        __init__(self,parent):
            Class initialization method
            Setting class member defaults

            Input parameters: event parent
            Output      : -
        """
        svm_event.SVMEVENT.__init__(self,parent)
        # Exposing class names
        self.eventClass('UserEvent')
        # Setting default parameters
        self.initialVelocity(10000.0)
        self.initialGear(1)
        self.endTime(20.0)
        self.frequency(1)
        self.lateralAccelerationLimit(3000)
        self.showLapSensorWidgets(0)

    def register (self):
        """
        register (self):
            Registering method for UserEvent class members

            Input parameters: -
            Output      : -
        """
        c = Class (self)
        c.addMember("viDriverFile",           type=StringType)
        c.addMember("preEventViDriverFile",  type=StringType)
```

```

c.addMember("eventClass",                                     type=StringType)
c.addMember("endTime",                                         type=FloatType, units ='time')
c.addMember("initialVelocity",                                type=FloatType, units='velocity')
c.addMember("initialGear",                                    type=IntType)
c.addMember("frequency",                                     type=FloatType, units='frequency')
c.addMember("lateralAccelerationLimit",                     type=FloatType, units='acceleration')

def getSolverSystem(self):
    """
        getSolverSystem(self):
            Method assembling model and event data to be passed to
            VI-CarRealTime solver

            Input parameters: -
            Output          : container class instance for vehicle model and event data
    """
    # basic class instance
    top_class = svm_event.SVMEvent.getSolverSystem(self)
    system = top_class.child(type=vfc.System)
    # creation of parameter tree for event data
    partree=vfc.ParameterTree(system)
    partree.name("EventParameters")
    # event class
    p = vfc.StringParameter(partree)
    p.name("eventClass")
    p.value(self.eventClass())
    # local variables for VI-Driver input data files:
    # - actual manoeuvre
    vdfFileName = self.name()+' .vdf'
    # - pre-analysis manoeuvre
    vdfPreFileName = self.name()+'_pre.vdf'
    # creating event data dictionaries and vdf files for the analyses
    self.createPreVDF(vdfPreFileName)
    self.createVDF(vdfFileName)
    # setting vi-driver input file class member
    self.viDriverFile(vdfFileName)
    if not self.viDriverFile():
        self.errorInData = True
        print("Error:")
        print("      VI-Driver Input File not found")
        return top_class
    # setting preferences parameter tree
    parameterTree = system.child(type=vfc.ParameterTree, name="preferences")
    self.fullSDFFFileName = vfc_files.fullFileName(self.viDriverFile())
    p = vfc.StringParameter(parameterTree)
    p.name("smart_driver_file")
    p.value(self.fullSDFFFileName)
    return top_class

def createPreVDF(self, vdfFileName):
    """
        createPreVDF(self, vdfFileName):
            Creates VDF file for the pre-analysis replacing the placeholders
            in the vdf prototype with the parameters defined in data dictionary

            Input parameters: vdf file path
            Output          : Boolean (True=success, False=fail)
    """
    #
    # getting units conversion factors
    # solver input file are in SI Units and the may not match with the event ones (set
    #
    # retrieve event units
    eventUnits = self.parent().parent().child(type=AfcUnits)
    # create SI units instance ( AFC class )

```

```

SIUnits      = AfcUnits ()
SIUnits.setSystem ("si")
SIUnits.setDimension ('length', 'meter')
SIUnits.setDimension ('angle', 'degree')
# get conversion factors for
ucf = UnitConversionFactors (eventUnits, SIUnits)
# length
self.lenUcf = ucf.factor('length')
# velocity
self.velUcf = ucf.factor('velocity')

# define data dictionary for vdf file creation
preVdpData = {}
preVdpData['end_time']=self.endTime()
preVdpData['initial_speed']=self.initialVelocity()*self.velUcf
preVdpData['initial_gear']=self.initialGear()
preVdpData['duration'] = 15.0
preVdpData['max_swa'] = 3.1416
# set path for vdf prototype
vdpFile = os.path.join(os.environ['VICRT_CUSTOM'], 'vdf_prototypes', 'step_steer.vdp'
# convert prototype to vdf
vicrt_event_utils.vdpToVdf (vdpFile ,vdfFileName , preVdpData )
return True

def createVDF(self, vdfFileName, data = None):
    """
        createPreVDF(self, vdfFileName):
        Creates VDF file for the analysis replacing the placeholders
        in the vdf prototype with the parameters defined by data dictionary

        Input parameters: vdf file path, event data dictionary
        Output          : Boolean (True=success, False=fail)
    """

    #
    # getting units conversion factors
    # solver input file are in SI Units and the may not match with the event ones (set :
    #
    # retrieve event units
    eventUnits = self.parent().parent().child(type=AfcUnits)
    # create SI units instance ( AFC class )
    SIUnits      = AfcUnits ()
    SIUnits.setSystem ("si")
    SIUnits.setDimension ('length', 'meter')
    SIUnits.setDimension ('angle', 'degree')
    # get conversion factors for
    ucf = UnitConversionFactors (eventUnits, SIUnits)
    # length
    #self.lenUcf = ucf.factor('length')
    # velocity
    self.velUcf = ucf.factor('velocity')
    # angle
    #self.angUcf = ucf.factor('angle')
    # acceleration
    #self.accUcf = ucf.factor('acceleration')
    #
    # Creating data dictionary if not passed as method parameter
    if not data:
        data = {}
        data['end_time']=self.endTime()
        data['initial_speed']=self.initialVelocity()*self.velUcf
        data['initial_gear']=self.initialGear()
        data['amplitude']= 1.0
        data['frequency']= self.frequency()
        data['duration']=self.endTime()
    # set path for vdf prototype

```

VI-CarRealTime

```

vdpFile = os.path.join(os.environ['VICRT_CUSTOM'], 'vdf_prototypes', 'sine_steer.vdp'
# convert prototype to vdf
vicrt_event_utils.vdpToVdf (vdpFile ,vdfFileName , data )

def runPreSolver(self, solverExe):
    """
        runPreSolver(self, solverExe):
            Solver call to run the pre-analysis.
            The method is used the pre-analysis events,
            retrieving the results and generating the vdf
            file (updated with pre-analysis outputs) for
            the final analysis

        Input parameters: solver executable path
        Output          : Boolean (True=success, False=fail)

    """
    # create output channels map for the event (safe for controlling names and content
    # of pre-event result file
    preOCF = self.name() + '_omap.xml'
    success = self.createOutputChannelsFile (preOCF)
    # create send file for pre-analysis
    preVDF = self.name()+'_pre.vdf'
    preOutputPrefix = self.name() + '_pre'
    success = vicrt_event_utils.eventToSendFile( self, outputChannelsFile = preOCF, vdf
    # Remove old result files if existing
    if os.path.exists(self.name() + '_pre.res'):
        os.remove(self.name() + '_pre.res')
    # Run solver (pre-analysis)
    preSendFile = preOutputPrefix+'_send_svm.xml'
    self.runSASolver(solverExe, True,preSendFile )
    # Compute simulation indices
    results = self.simResultsCompute (resultFile = self.name() + '_pre.res')
    if results == None:
        print('\n Unable to retrieve a valid steering angle value.')
        return False
    # Update dictionary data with also the values retrieved by the pre-analysis
    # (results array)
    data ={}
    data['end_time']=self.endTime()
    data['initial_speed']=self.initialVelocity()/1000.0
    data['initial_gear']=self.initialGear()
    data['amplitude']= results[0]
    data['frequency']=self.frequency()
    data['duration']=self.endTime()
    # create vdf for the analysis with the updated data
    self.createVDF(self.name()+'_.vdf', data = data)
    # Remove Pre analysis files
    if os.path.isfile(self.name() + '_pre.res'):
        os.remove(self.name() + '_pre.res')
        os.remove(self.name() + '_pre_send_svm.xml')
        os.remove(self.name() + '_pre_solver_svm.log')
        os.remove(self.name() + '_pre.vdf')
        os.remove(self.name() + '_omap.xml')
    # runPreSolver method must return True in order for the svm_events class to run t!
    return True
    else:
        print('\n.... Pre Analysis Event Finished')
        return False

def simResultsCompute (self, resultFile):
    """ simResultsCompute (self, resultFile):
        Compute the simulation results.
        The method extracts desired vectors from result file

        Input parameters: result file path

```

```

        Output      : results array
"""

# getting units conversion factors
# solver input file are in SI Units and the may not match with the event ones (set :
#
# retrieve event units
eventUnits = self.parent().parent().child(type=AfcUnits)
# create SI units instance ( AFC class )
SIUnits   = AfcUnits ()
SIUnits.setSystem ("si")
SIUnits.setDimension ('length', 'meter')
SIUnits.setDimension ('angle', 'degree')
# get conversion factors for
ucf = UnitConversionFactors (eventUnits,SIUnits)
# acceleration in G's
self.accGUcf = ucf.factor('acceleration')/9.8065
if os.path.exists(resultFile):
    # load results
    print(.. Reading results file ...)
    # extract relevant components
    components = vicrt_event_utils.resultFileToComponent (resultFile, fullCmpNamLst =
vx      = components[0]
ay      = components[1]
swa     = components[2]
man_id = components[3]
[vx,ay,swa,man_id]
print(.. Extracting components ...)
# using absolute lateral acceleration as metrics
for i in range(0,len(ay)):
    ay[i]=abs(ay[i])
# sort steering wheel angle by lateral acceleration
[ayAsc,swaNew] =vicrt_event_utils.sortByComponent (ay,swa,maneuverID=man_id, id=
if not ayAsc:
    print(.. Failed!)
    return None
# setting lateral acceleration limit value for final event
accyLimit = self.lateralAccelerationLimit()*self.accGUcf
if (max(ayAsc) < accyLimit):
    print('warning: \nLateral Acceleration Limit (' + str(accyLimit) + ' g) is bigg
for i in range(3,len(ayAsc)):
    # getting steering wheel angle
    steeringVal=swaNew[i]
    if abs(ayAsc[i])>(accyLimit):
        break
    return [steeringVal]
else:
    return None

def createOutputChannelsFile(self, fileName ):
"""
    createOutputChannelsFile(self, fileName ):
    Creates a custom channels map file.
    The method generates an output map controlling the number
    and names of result set components

    Input parameters: result file path
    Output       : Boolean (True=success, False=fail)
"""

# Defining components and names to be included in pre event results
# Note: the three lists below must have the same number of elements
chilSt   = ['chassis_velocities_longitudinal',
            'chassis_accelerations_lateral',
            'driver_demands_steering',
            'driving_machine_monitor_maneuver_ID']

```

```

resSetLst = ['chassis_velocities', 'chassis_accelerations', 'driver_demands', 'driv.
cmpLst    = ['longitudinal', 'lateral', 'steering', 'maneuver_ID']
# create output channels map file
return vicrt_event_utils.createCustomOutputChannelsFile(fileName, chLst, resSetLst

```

- create a VI-Driver-Prototype (vdp) file called `sine_steer.vdp` and store it in the `Custom_Events/vdf_prototypes` folder

`sine_steer.vdp`

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'VDF'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----DEBUG_PARAMETERS
[DEBUG_PARAMETERS]
ACTIVATION = 'FALSE'
INITIALIZATION_DUMP = 'FALSE'
$-----GLOBAL_SETTINGS
[GLOBAL_SETTINGS]
MODEL_TYPE = 'CAR_STANDARD'
$-----CONTROLLER_STANDARD
[CONTROLLER_STANDARD]
PREVIEW_TIME = 0.5
MIN_PREVIEW_DISTANCE = 5
LAT_ANT_COMP_TIME = 0
LON_ANT_COMP_TIME = 0
LONGITUDINAL_SPEED_CONTROL = 'LONVEL'
PREVIEW_DISTANCE_CALC = 'STANDARD'
STEERING_PDC_ACTIVE = 'TRUE'
STEERING_YAW_PID_ACTIVE = 'TRUE'
THROTTLE_FEED_FWD_TRQ_ACTIVE = 'TRUE'
THROTTLE_SAT_TGTACCX_ACTIVE = 'TRUE'
$-----STEERING_STANDARD
[STEERING_STANDARD]
MAX_VALUE = 12.56637061435917
MIN_VALUE = -12.56637061435917
$-----THROTTLE_STANDARD
[THROTTLE_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
$-----BRAKING_STANDARD
[BRAKING_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
$-----CLUTCH_STANDARD
[CLUTCH_STANDARD]
MAX_VALUE = 1
MIN_VALUE = 0
SCALING_FACTOR = 1
MIN_RPM = 500
ACTIVATION = 'TRUE'
MODEL = 'STANDARD'
$-----STARTUP
[STARTUP]
STARTING_TIME = 0

```

```

INITIAL_SPEED = param(initial_speed)
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRAKING = 0
INITIAL_BRAKING2 = 0
INITIAL_GEAR = param(initial_gear)
INITIAL_CLUTCH = 0
INITIAL_SETUP = 'OFF'
$-----MANEUVERS_LIST
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'MANEUVER_1' param(duration) 0.01
$-----MANEUVER_1
[MANEUVER_1]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'
(STEERING)
METHOD = 'OPENLOOP'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0
SIGNAL = 'MANEUVER_1_STEERING_SIGNAL'
(THROTTLE)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0
(BRAKING)
METHOD = 'OPENLOOP'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0
SIGNAL = 'MAINTAIN_SIGNAL'
(GEAR)
METHOD = 'OPENLOOP'
SMOOTHING_ACTIVE = 'FALSE'
SMOOTHING_TIME = 0
SIGNAL = 'MAINTAIN_SIGNAL'
(CLUTCH)
METHOD = 'OPENLOOP'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0
SIGNAL = 'MAINTAIN_SIGNAL'
(BRAKING2)
METHOD = 'USER'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(MACHINE)
SPEED = 'MANEUVER_1_VELMAP_T'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'FALSE'
SPEED_MAP_ABS_TIME = 'FALSE'
SPEED_MAP_GLOBAL_S = 'FALSE'
CONNECT_PATH = 'FALSE'
$-----MAINTAIN_SIGNAL
[MAINTAIN_SIGNAL]
TYPE = 'MAINTAIN'
SCALING_FACTOR = 1
CALC_MODE = 'ABSOLUTE'
AUTO_SCALING = 'TRUE'
AUTO_SATURATION = 'TRUE'
$-----MANEUVER_1_STEERING_SIGNAL
[MANEUVER_1_STEERING_SIGNAL]
TYPE = 'SINE'
AMPLITUDE = param(amplitude)
INITIAL_PHASE = 0
FREQUENCY = param(frequency)
SCALING_FACTOR = 1
CALC_MODE = 'ABSOLUTE'

```

```

AUTO_SCALING = 'TRUE'
AUTO_SATURATION = 'TRUE'
$-----MANEUVER_1_VELMAP_T
[MANEUVER_1_VELMAP_T]
TYPE = 'MAP'
X_DATA = 'TIME'
Y_DATA = 'LONVEL'
(VALUES)
{ t Vx }
0 param(initial_speed)
0.1 param(initial_speed)
param(duration) param(initial_speed)

```

- create a new [event editor](#) class and store the file (*my_sine_event_editor.py*) in the *Custom_Events/visedit* folder.

❑ **my_sine_event_editor.py**

```

#!/usr/bin/env python

from mdi.visedit.svm.user_event_layout import UserEventLayout

from mdi.visedit.editors import *
from mdi.afc      import svm_paths
from mdi.afc.afckernel import Units, UnitSetting, UnitValue, UnitDimension, UnitSystem, U
from mdi.visedit import chassis_qt_extensions
from mdi.visedit import vfc_files
from mdi.afc      import dcf

import os, string
import qt
from vigrade import vicrt_pth

#####
# this class enables the communication between user defined class event and the
# event GUI;
class MySineEventDialog (UserEventLayout):

    def __init__(self, parent = None, name = None, modal = 0, fl = 0):
        UserEventLayout.__init__(self, parent)
        self.afcConnect()
        self.initialGear_spin.setMaximum(7)
        self.hideAll()
        self.renameItems()
        self.showItems()
        self.show()

    def afcConnect (self):
        self.items = []
        FieldConnector      (self, self.endTime_field,           'endTime')
        FieldConnector      (self, self.initialVelocity_field, 'initialVelocity')
        SpinBoxConnector   (self, self.initialGear_spin,       'initialGear')
        CheckBoxConnector  (self,  self.userCheck1_check,      'cruiseControl')
        FieldConnector      (self, self.userField1_field,     'frequency')
        FieldConnector      (self, self.userField2_field,     'lateralAccelerationLimit')

    def load(self, object):
        self.object = object
        for item in self.items:
            item.load(self.object)

    def renameItems(self):
        self.event_GroupBox.setTitle('User Event (Sine Steer)')
        #self.userCheck1_check.setText('Cruise control')

```

```

    self.userField1_label.setText('Frequency')
    self.userField2_label.setText('Lateral Acceleration Limit')

def hideAll (self):
    for child in self.event_GroupBox.children():
        if isinstance(child,QLineEdit) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
        if isinstance(child, QLabel) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
        if isinstance(child, QCheckBox) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
        if isinstance(child, QComboBox) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
        if isinstance(child, QSpinBox) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
        if isinstance(child, QPushButton) :
            child.hide()
            child.setToolTip(self.trUtf8(""))
def showItems(self):
    self.endTime_field.show()
    self.endTime_label.show()
    self.initialVelocity_field.show()
    self.initialVelocity_label.show()
    self.initialGear_spin.show()
    self.initialGear_label.show()
    self.userField1_label.show()
    self.userField1_field.show()
    self.userField2_label.show()
    self.userField2_field.show()

```

- update the [customEditorsRegister](#) function in order to load and connect the new event class and the related GUI.

custom_editors_register.py

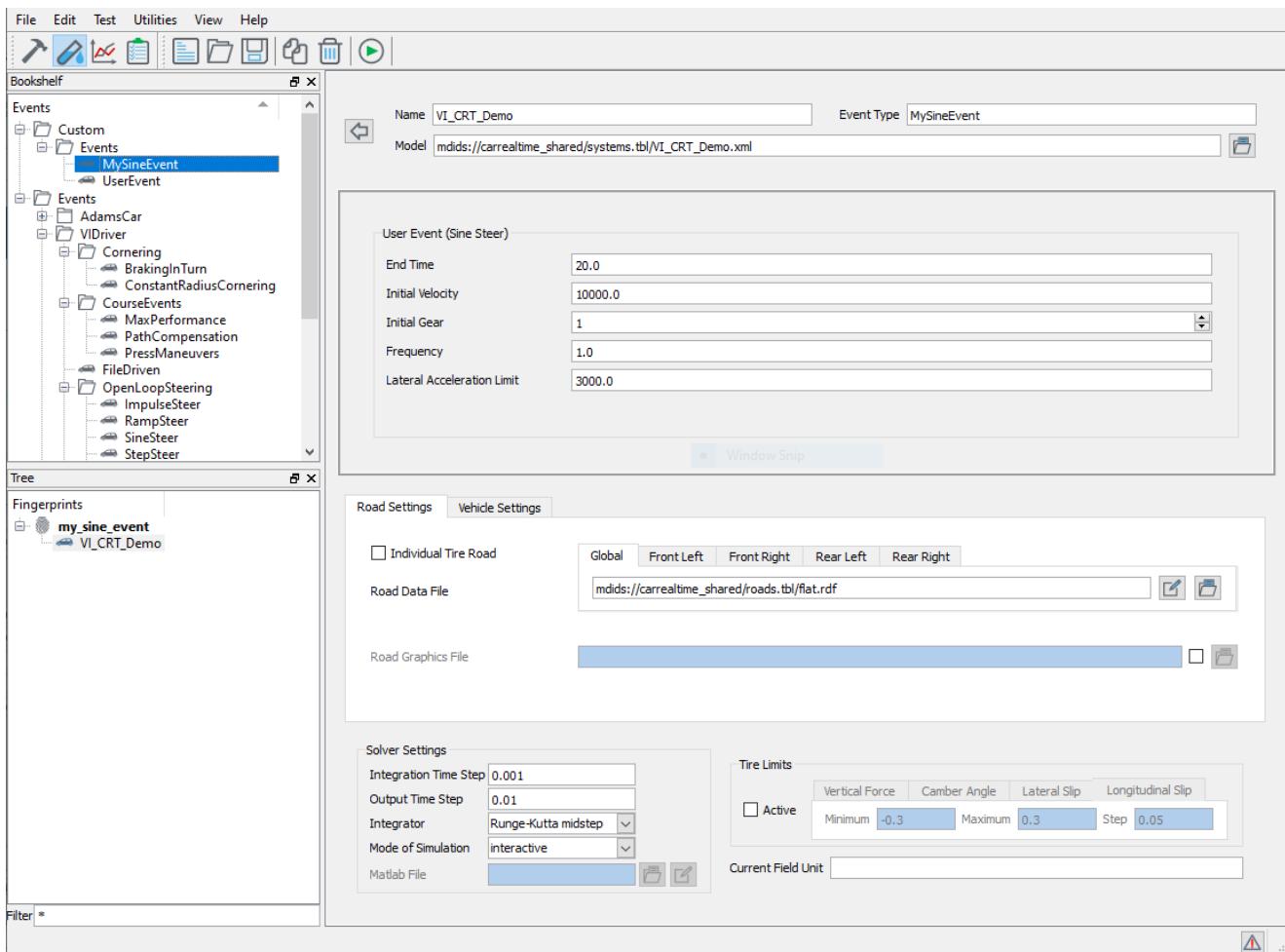
```

# import modules
import os,sys
from mdi.afc import afckernel
from mdi.afc import svm_paths
from mdi.visedit.editors import *
import qt
from mdi.visedit import vfc_files
from mdi.visedit.svm.event_gui_info import EventGuiInfo

def customEditorsRegister (guiInfos):
    # all the calls to user_event_editor (class connecting dialog box fields
    # to user event class members) have to be added here!
    if hasattr (svm_paths,"UserEvent"):
        g = EventGuiInfo ()
        g.editorClass ("UserEventDialog")
        g.editorFile ("user_event_editor")
        guiInfos[svm_paths.UserEvent] = g
    if hasattr (svm_paths,"MySineEvent"):
        g = EventGuiInfo ()
        g.editorClass ("MySineEventDialog")
        g.editorFile ("my_sine_event_editor")
        guiInfos[svm_paths.MySineEvent] = g
    return guiInfos

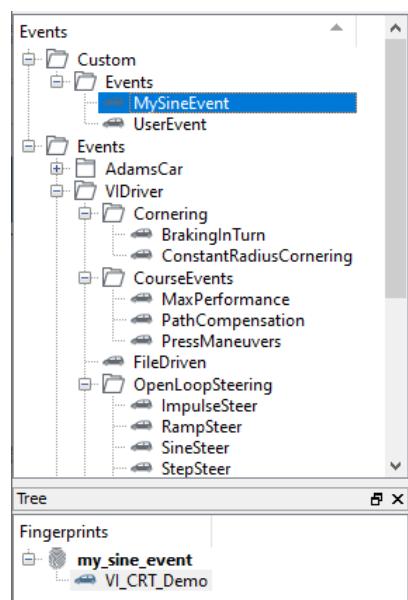
```

- close and open again VI-CarRealTime, so that the new event can be load in the parameter tree.

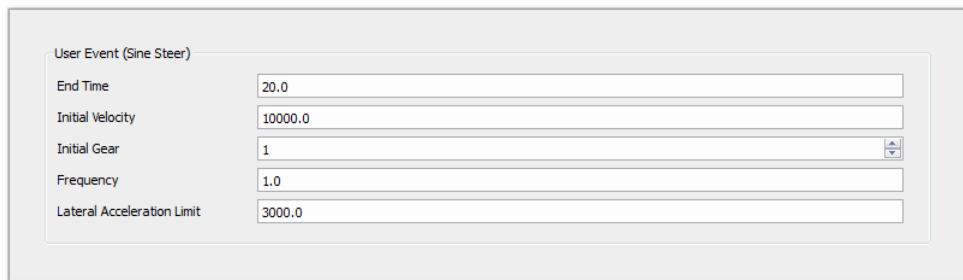


Run Custom Event

- Right click on **MySineEvent** event and select: **Add Event To New Fingerprint**.
- Rename the fingerprint to **my_sine_event**.



- Set the event values as shown in the picture below:



- Run the simulation using the button.

The maneuver is composed by a pre-analysis and a main analysis: the pre-event is a step_steer used to detect the steering value which correspond to an acceleration of 0.3 g. This steering value is used in the main analysis (sine steer) as steering amplitude.

```
VI.PTW (VI-CarRealTime Python Task Window) - X

Static equilibrium has been achieved.
Elapsed Simulation Time: 3.061 (sec).
Elapsed Clock Time: 0.409 (sec).

Performing dynamic simulation...

Simulation Progress (percent complete):
0 50 100
-----
=====
= VI-CarRealTime =
= VEHICLE SIMULATION STATISTICS =
=====

--- Simulation Statistics ---
>> LAP TIME = 20.000000 sec
>> LAT ACCEL MAX = 0.340016 G
>> LAT ACCEL MIN = -0.340057 G
>> LON ACCEL MAX = 0.009747 G
>> LON ACCEL MIN = -0.046181 G
>> LON SPEED MAX = 36.019002 km/h
>> LON SPEED MIN = 35.496214 km/h
>> LON SPEED AVG = 35.934361 km/h
>> STEERING MAX = 1.482573 rad
>> STEERING MIN = -1.482573 rad

Writing output to file. Please Wait ...

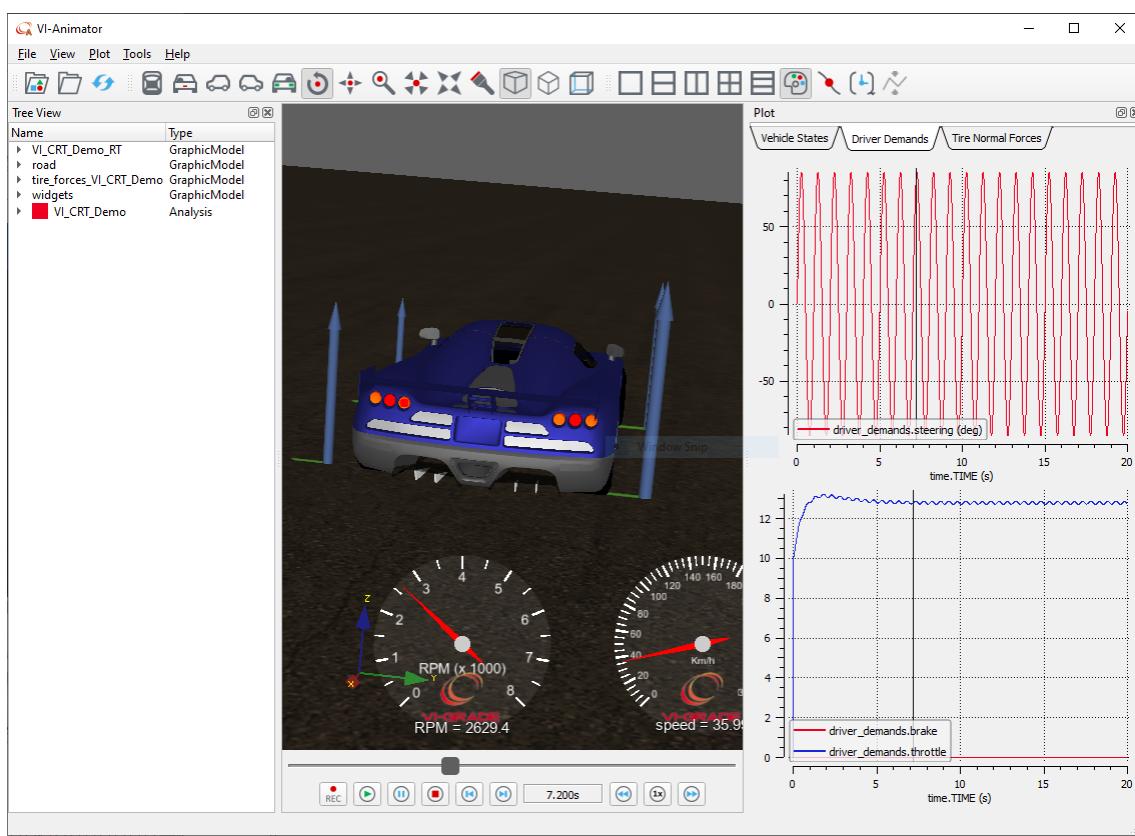
Elapsed Simulation Time: 20.000 (sec).
Elapsed Clock Time: 3.113 (sec).
Computational Efficiency: 6.425 (sim. sec)/(sec).

Simulation is complete.
```

Review Results

- Switch to Review mode using the  button.
 - Select the **VI_CRT_Demo** event using left mouse button and then click .

VI-Animator will start loading event results and animation.



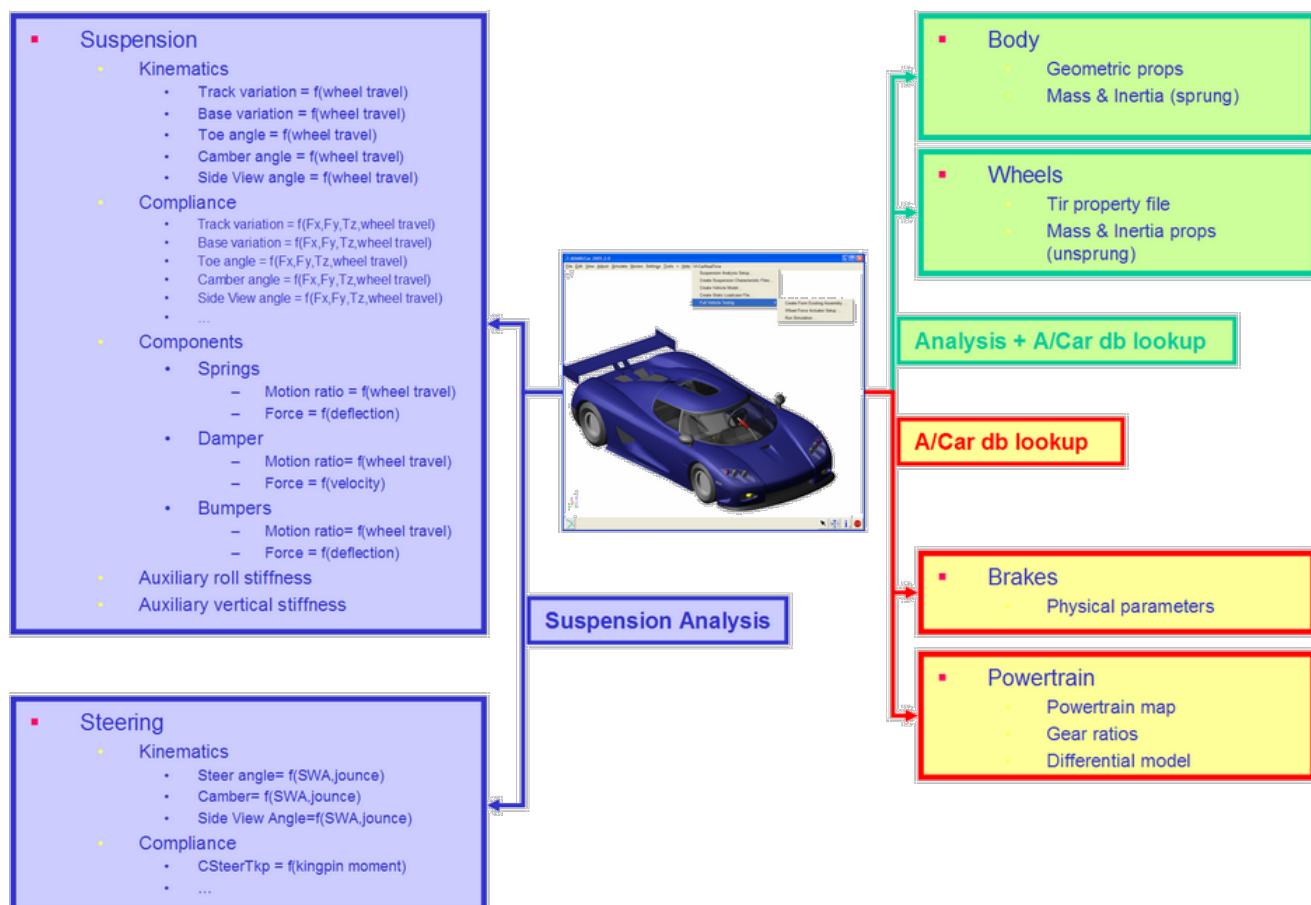
As you can see from the plots, the main event is a sine steer with the frequency of 1 Hz and an amplitude of **84.95 degrees**, which corresponds to the steering value that generates a chassis lateral acceleration on **0.3 g** in a step steer event.

2 VI-CarRealTime Adams Interface

VI-CarRealTime includes a plug-in for Adams Car for creating vehicle models from existing full vehicle assemblies. Several procedures for extracting characteristics and data that describe the Adams Car model are included.

The parameter extraction works through a set of analyses (suspension and full vehicle) and database lookups collecting all required data. The information is then saved into property files and organized in a database (shareable with Adams Car model).

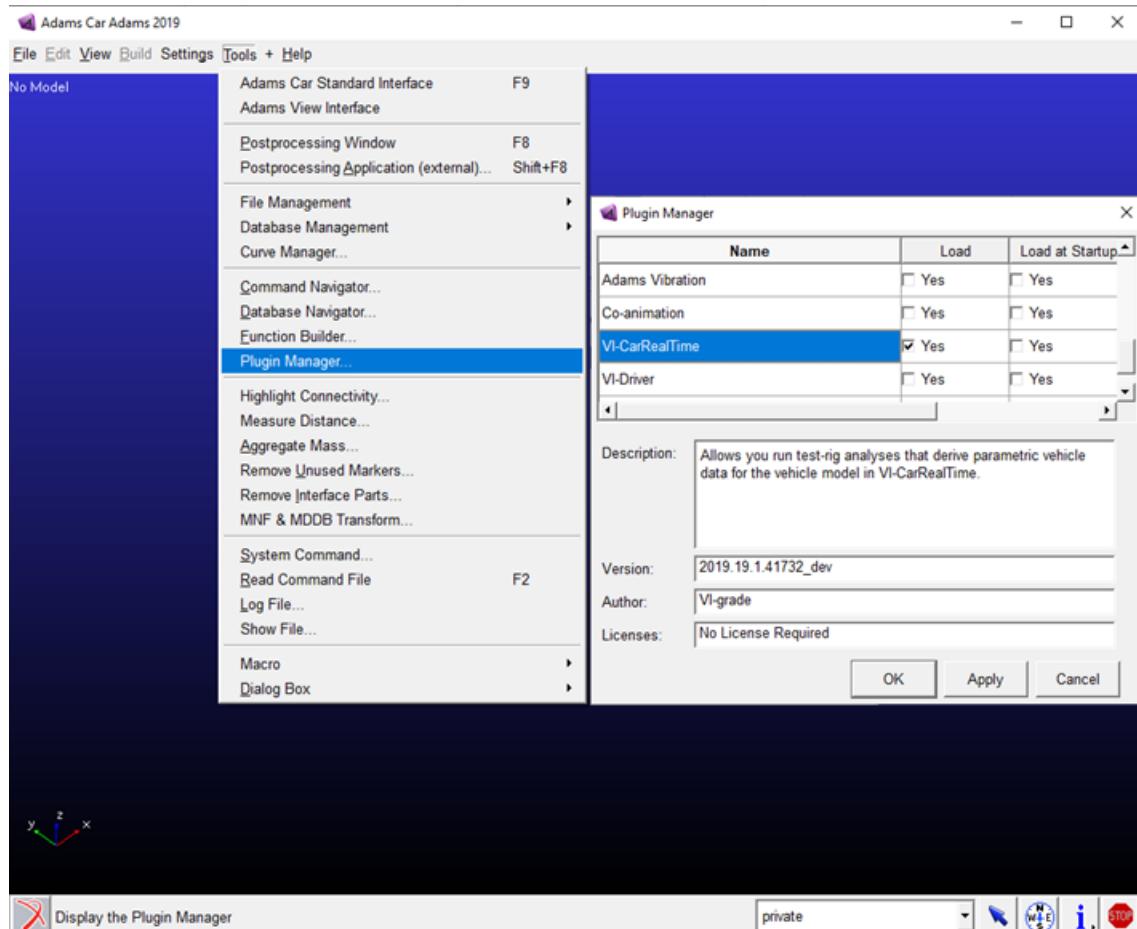
The diagram below shows the process flow of parameter extraction.



Loading the plugin

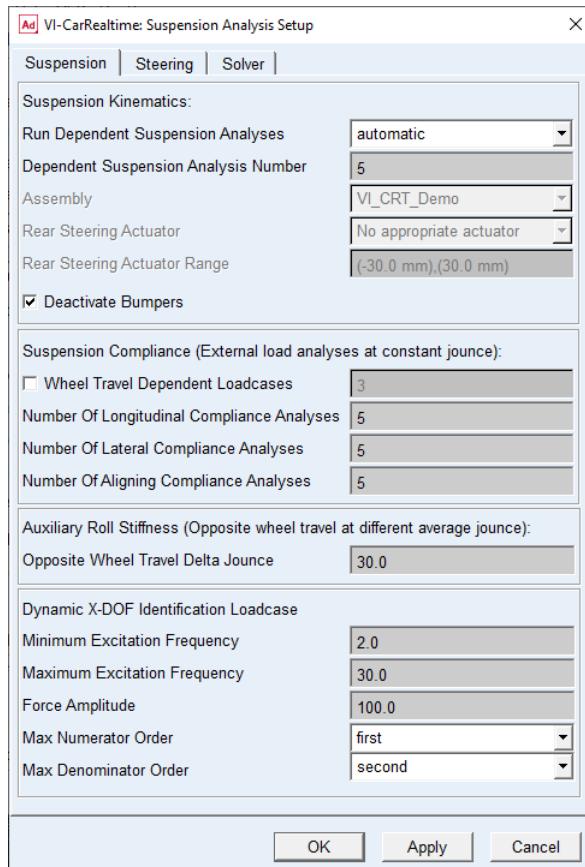
The plugin can be loaded using the Plugin Manager from the tools menu. In the Plugin Manager scroll down to the "VI-CarRealTime" Plugin and check the corresponding box in the the "Load" column. Finally select "OK".

The "VI-CarRealTime" menu will now appear in the menu bar containing all the tools needed to export the model from Adams Car to VI-CarRealTime.



2.1 Suspension Analysis Setup

In the Suspension Analysis Setup dialog box parameters are set for the suspension analyses that are included in the parameter extraction procedure from Adams Car to VI-CarRealTime.

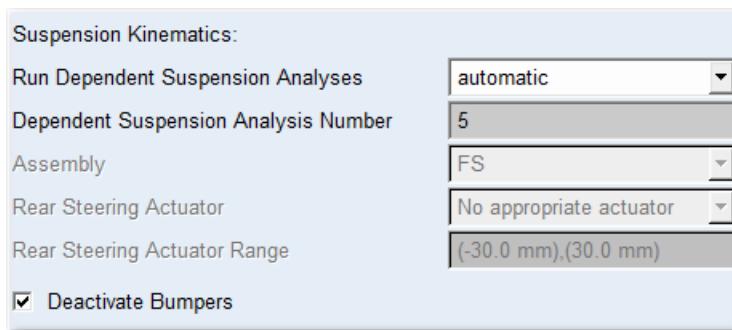


See in detail what can be set in the different tab containers:

- Suspension
 - [Suspension Kinematics](#)
 - [Suspension Compliance](#)
 - [Auxiliary Roll Stiffness](#)
 - [Dynamic X-DOF Identification Loadcase](#)
- [Steering](#)
- [Solver](#)

2.1.1 Suspension Kinematics

In the Suspension Kinematics container the type of dependency of kinematic curves on wheel jounce for the rear suspensions is defined; dependent suspensions will have kinematic curves depending on both left and right wheels jounce.



Run Dependent Suspension Analyses

Option menu which allows the user to choose how to extract the suspension kinematic curves both for front and rear suspension. Available options are:

- *no*
the suspension curves are forced to be extracted as independent
- *front only*
only the front suspension is forced to be extracted as dependent.
- *rear only*
only the rear suspension is forced to be extracted as dependent.
- *front and rear*
both front and rear suspensions are forced to be extracted as dependent.
- *automatic*
the choice is defined basing on *suspension type* variable defined in the *suspension parameter array*.
- *4 wheel steer*
enables the export of [Four wheel steering](#) models.

Dependent Suspension Analysis Number

Defines the number of analyses used to determine the suspension kinematics in dependent mode. It corresponds to the number of values of the second independent variable (Z) in the suspension curves.

Four wheel steering

VI-CarRealTime supports the possibility of modeling four wheel steered vehicles (in conjunction with MATLAB/Simulink).

In order to export Adams/Car model to VI-CarRealTime, including rear wheel steering option, some assumptions have to be done:

- rear suspension type is independent;
- the rear steering will be operated in Adams Car model by using joint motion actuator (*ac_joint_motion_actuator* UDE type);
- the actuator must have been implemented in a subsystem having *minor_role = "rear"* ;
- the actuator must be active when exporting the model;

A further step is needed to setup the model for the export, informing the plug-in that a specific joint motion actuator is associated to the steering.

When this procedure is completed the export phase is managed by the plug-in, by exercising the wheel suspension jounce for different joint motion actuator configurations, in order to get full suspension kinematics.

The resulting suspension curves will show a dependency of suspension kinematic characteristics (track and base variation, side view angle, toe angle, camber angle) on *wheel jounce* and *joint motion actuator position*.

The following parameters are defined when the 4 wheel steer option is active:

- **Assembly**
name of the full vehicle assembly to be exported using 4 wheel steer option;

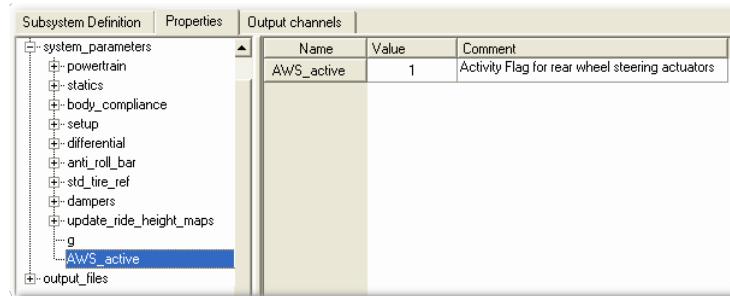
- **Rear Steering Actuator**

option menu storing all the actuators that have been found in the Assembly. It's up to the user to select the correct one.

- **Rear Steering Actuator Range**

range of suspension steering actuator exercised during suspension extraction.

Note: the vehicle assembly export procedure will add an integer parameter in the vehicle properties that can be used to deactivate steering dependency when using stand alone solver. In fact the rear steering actuators is supposed to be driven by [Rear Steering Actuator Disp](#) input channel. System parameter named **AWS_active** is used as activity flag.



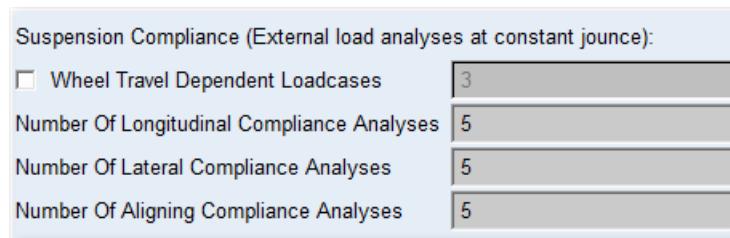
Deactivate bumpers

The check box gives to the user the possibility of activating/deactivating bumpers during suspension export procedure. By setting the bumpers activity to true it is possible to include the effect of bumper force (if engaging with suspension travel) on vehicle compliance.

Furthermore the flag has an effect on bumpers activity in [unsprung mass evaluation analysis](#): if the flag is checked the bumpers are active, otherwise they are switched off.

2.1.2 Suspension Compliance

In the Suspension Compliance container it is set the number of analyses that are performed during compliance analyses.



The values in the fields define the number compliance loadcases executed at constant values of wheel travel. They correspond to the columns entered in the compliance 3D splines (see picture below).

For each column, corresponding to a single value of left and right wheel travel, the longitudinal and lateral forces and the aligning moment are separately swept from their minimum to maximum value, as defined in the [main export panel](#).

Number Of Longitudinal Compliance Analyses

Corresponds to the number of jounce values in longitudinal force (driving, braking) compliance curves.

Number Of Lateral Compliance Analyses

Corresponds to the number of jounce values in lateral force compliance curves.

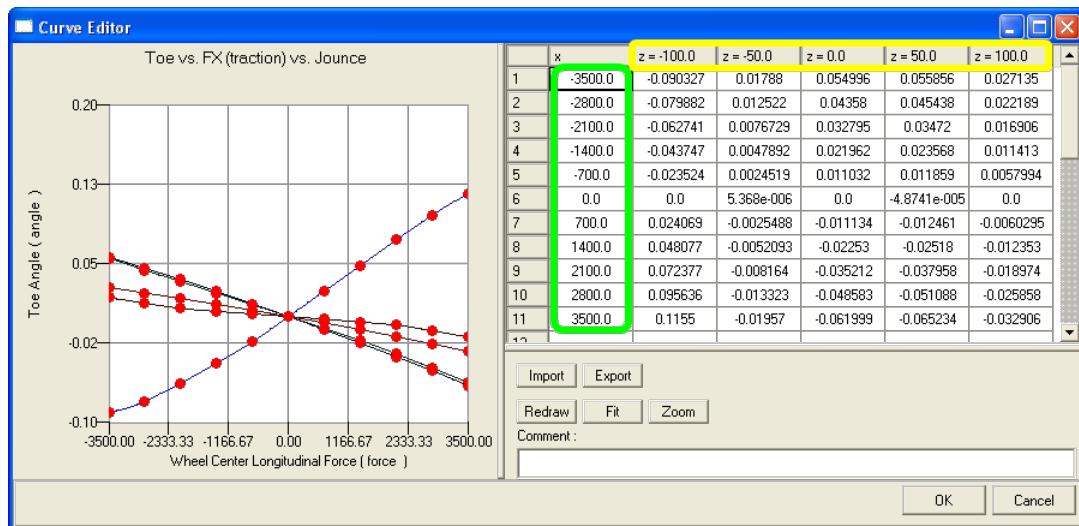
Number Of Aligning Compliance Analyses

Corresponds to the number of jounce values in aligning torque compliance curves.

The following figure shows how the compliance 3D spline are mapped.

X values (first independent variable, green in the figure) represent the values of the force from minimum to maximum value.

Z values (second independent variable, yellow in the figure) represent the values of the suspension jounce from minimum to maximum value and their number is defined by the number entered in the related Number of Compliance Analyses field.



Wheel Travel Dependent Loadcases (Experimental feature)

When the feature is not enabled the compliance loadcases are executed applying the forces in a kinematic configuration having left and right wheel travel set at the same value (parallel travel config).

When the toggle button is enabled, the computation mode of the suspension compliance is enhanced in order to take into account side wheel travels independently; the number entered in the field indicates the number of compliance sets extracted for each side (left and right).

For each set the wheel travel of one wheel is kept constant, while varying external force and wheel travel for the other wheel.

Example: setting the value of Wheel TravelDependent Loadcase field to 3, three sets of direct compliance curves will be extracted both for left and right as follows:

Left compliance

- the jounce of the *right* suspension is fixed to the **lowest** value (set in the [main export panel](#)) and the jounce of the *left* suspension is swept from the lowest to the highest value depending on the number of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.
- the jounce of the *right* suspension is fixed to the **intermediate** value and the jounce of the *left* suspension is swept from the lowest to the highest value depending on the number of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.
- the jounce of the *right* suspension is fixed to the **highest** value (set in the [main export panel](#)) and the jounce of the *left* suspension is swept from the lowest to the highest value depending on the number of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.

Right compliance

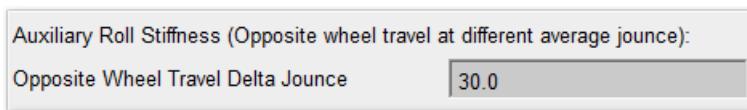
- the jounce of the *left* suspension is fixed to the **lowest** value (set in the [main export panel](#)) and the jounce of the *right* suspension is swept from the lowest to the highest value depending on the number

of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.

2. the jounce of the *left* suspension is fixed to the **Intermediate** value and the jounce of the *right* suspension is swept from the lowest to the highest value depending on the number of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.
3. the jounce of the *left* suspension is fixed to the **highest** value (set in the [main export panel](#)) and the jounce of the *right* suspension is swept from the lowest to the highest value depending on the number of steps defined. For each of these steps, while the suspension jounces are kept fixed, the longitudinal and lateral forces and the aligning torques are separately swept from minimum to maximum.

2.1.3 Auxiliary Roll Stiffness

In the Auxiliary Roll Stiffness (Opposite wheel travel at different average jounce) container the opposite wheel travel range for auxiliary roll curve is set.

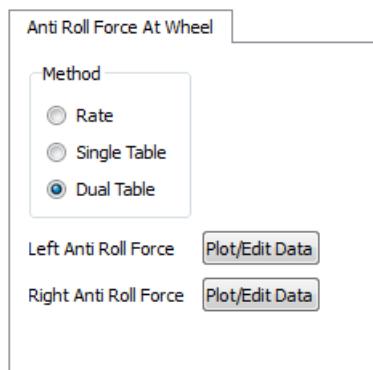


The export procedure computes the auxiliary roll forces by performing a set of opposite wheel travel analysis with and without the anti-roll bar.

For deactivating the anti-roll bar, VI-CarRealTime plug-in looks for a subsystem with major role set to `antirollbar` or for a parameter variable called `pvs_antirollbar_active` in the suspension subsystem.

The Auxiliary Roll Forces in the extracted model are mapped in two modes:

1. [Dual Table](#): Forces at Wheel as a function of left and right wheel travel (default option in VI-CarRealTime);
2. [Single Table](#): Forces at Wheel as a function of opposite jounce and average jounce.

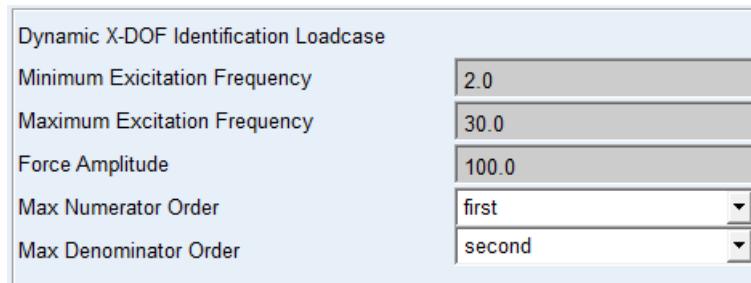


Opposite Wheel Travel Delta Jounce

Defines the delta jounce range for the auxiliary anti roll force applied during the analysis. The resulting curves are mapped in Single Table panel.

2.1.4 Dynamic X-DOF Identification Loadcase

In the Dynamic X-DOF Identification Loadcase container, the values for the dynamic analysis used to characterize the longitudinal behavior of the suspension are set.



The export procedure submits a dynamic analysis sweeping the longitudinal force from minimum to maximum frequency and then post-process the results characterizing the suspension x-dof by computing the coefficients of a [transfer function](#).

Both left and right wheels are excited with the same longitudinal force time history:

$$F_x^{[L,R]} = \text{SWEEP}(\text{time}, a, 0, f_0, \text{duration}, f_1, 1.0)$$

A constant vertical force is also applied to both left and right pads; such force corresponds to the static jacking force computed at jounce = 0.

Minimum Excitation Frequency

Minimum frequency (Hz) of the sweep function. It corresponds to f_0 parameter of the Adams/Solver SWEEP function.

Maximum Excitation Frequency

Maximum frequency (Hz) of the sweep function. It corresponds to f_1 parameter of the Adams/Solver SWEEP function.

Force Amplitude

Amplitude of the force (N) used in the sweep function. It corresponds to a parameter of the Adams/Solver SWEEP function.

Max Numerator Order

Order of the numerator of the transfer function used to characterize the longitudinal degree of freedom. Available choices are: zero, first, second .

Max Denominator Order

Order of the denominator of the transfer function used to characterize the longitudinal degree of freedom. Available choices are: second, third, fourth .

Note: for all the details about the SWEEP function, please refer to Adams documentation.

Note: the duration of the dynamic analysis that is submitted is computed as $\text{duration} = 0.2 * \text{Maximum Excitation Frequency}$.

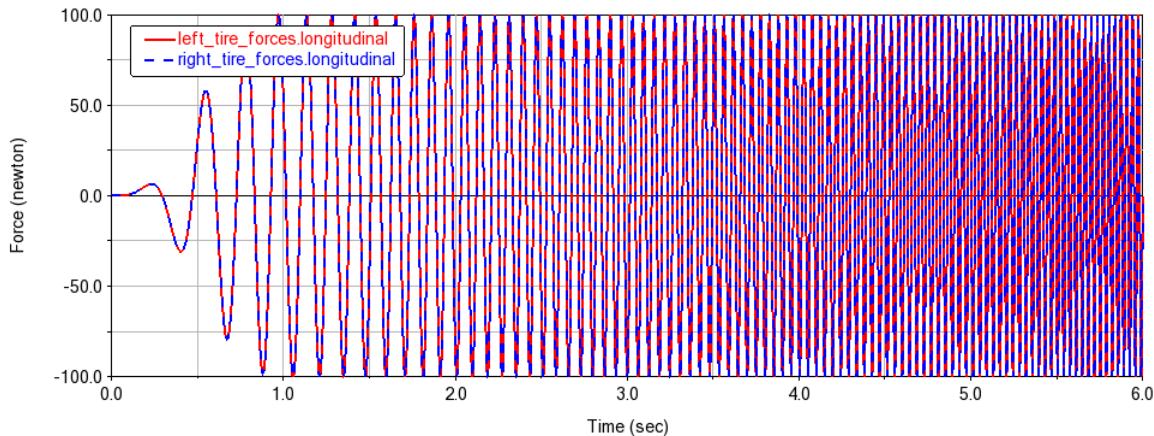
Here an example of the excitation force with:

Minimum Excitation Frequency = 1 Hz

Maximum Excitation Frequency = 30 Hz

Force Amplitude = 100 Hz

(duration = $0.2 \times 30\text{Hz} = 6\text{s}$)



Longitudinal force and wheel travel base displacement result set components are used during post-processing phase to compute the transfer function coefficients.

In particular:

1. wheel travel base displacement vs. longitudinal force curve is created
2. the FFT of such curve is computed
3. such FFT is fitted in the frequency range specified by the user (Minimum Excitation Frequency - Maximum Excitation Frequency)
4. the numerator and denominator coefficients are extracted depending on the order specified by the user (Max Numerator Order - Max Denominator Order)

2.1.5 Steering Kinematics

In the Steering Kinematics (SWA sweeps at constant jounce) container the number of analyses performed to define the steering system kinematics is set.

Steering Kinematics (SWA sweeps at constant jounce):	
Number Of Analyses	<input type="text" value="5"/>
Maximum Steering Wheel Angle	<input type="text" value="180.0"/>
Maximum Aligning Torque	<input type="text" value="5.0E+005"/>
<input checked="" type="checkbox"/> Steering Rack Analysis	

Number of Analyses

Corresponds to the number of jounce values at which each kinematic curve for steering system is evaluated.

Maximum Steering Wheel Angle

Sets the range of steering wheel angles.

Maximum Aligning Torque

Sets the range of aligning moment applied to wheels in the loadcases used to compute steering wheel torque and rack force maps.

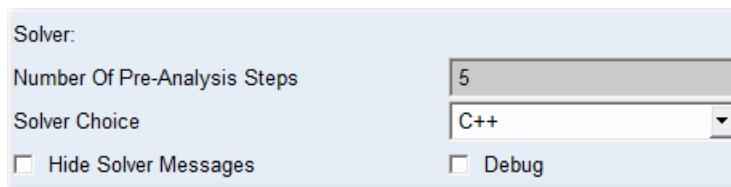
Steering rack analyses

Executes suspension analyses for populating the rack forces maps.

For subsystem not including a rack component, the steering rack (i.e. parallel link) analysis will produce dummy results (a warning will be issued by the plug-in in such cases).

2.1.6 Solver

In the Solver container it is possible to change the solver or to enable error debug.



Number Of Pre-Analysis Steps

Corresponds to the number of static analysis steps that are performed to pass from a suspension jounce configuration to the following one in a static load-case analysis. Increasing such number means that more steps are performed to pass from one configuration to another one and this can help the solver to achieve the static equilibrium when the suspension layout of the last sub-loadcase configuration is quite different with respect to the first configuration of the sequent sub-loadcase.

Solver Choice

Option menu which allows to choose the Adams solver to use to run the suspension analyses: C++ (default) or FORTRAN.

Hide solver messages

The check box disables the print out of the messages coming from A/Solver resulting in a faster export process. It is suggested to disable the check box activity during debug.

Debug

The check box optionally activates a verbose export process including the possibility of keeping alive the suspension subsystems at the end full vehicle export.

2.2 Create Suspension Characteristic Files

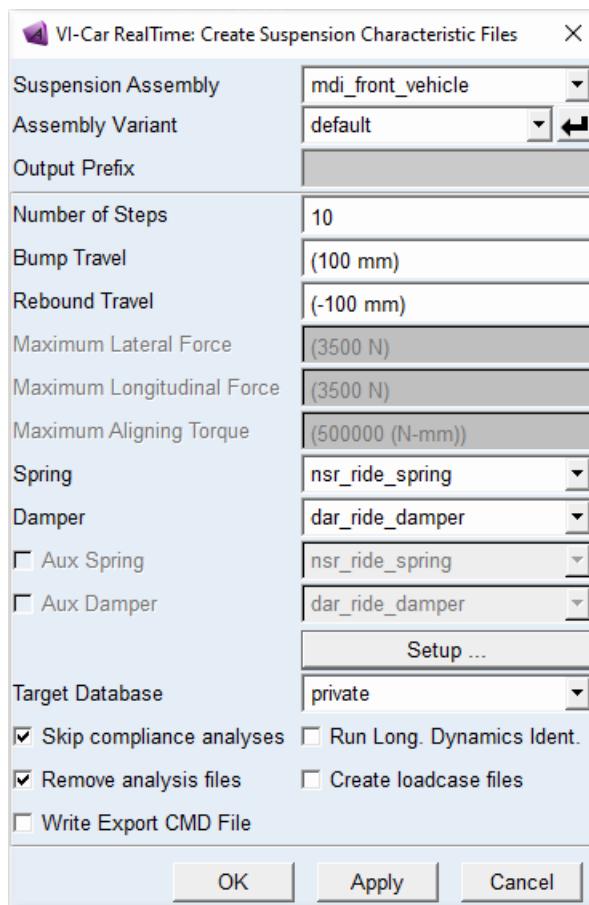
The Create Suspension Characteristic files dialog box is used to **create suspension data from an existing suspension assembly**.

If the assembly includes a steering system, a steering subsystem file is created as well.

Look at:

- [Parameters](#)
- [Sequence of analyses performed](#)

2.2.1 Parameters



- **Suspension Assembly**

suspension assembly to be exported.

- **Output Prefix**

prefix used for analysis names and for created subsystem files.

- **Number Of Steps**

number of steps for each analysis; it corresponds to the number of points for the first independent variable of the suspension curves.

- **Bump Travel**

jounce upper limit for suspension analyses.

- **Rebound Travel**

jounce lower limit for suspension analyses.

- **Maximum Lateral Force**

Maximum force value used for longitudinal compliance evaluation (force ranges from -MaxForce to +MaxForce).

- **Maximum Longitudinal Force**

Maximum force value used for lateral compliance evaluation (force ranges from -MaxForce to +MaxForce).

- **Maximum Aligning torque**

Maximum torque value used for aligning compliance evaluation (torque ranges from -MaxTorque to +MaxTorque).

- **Spring**

main spring component to be exported in VI-CarRealTime subsystem.

- **Damper**

main damper component to be exported in VI-CarRealTime subsystem.

- **Aux Spring**

auxiliary spring component to be exported in VI-CarRealTime subsystem (up to two spring pair per suspension is supported). The component will be considered in parallel to other spring. The effects of other springs will be included in auxiliary vertical stiffness.

- **Aux Damper**

auxiliary damper component to be exported in VI-CarRealTime subsystem (up to two damper pair per suspension is supported) The component will be considered in parallel to other damper.

- **Target Database**

Database in which the exported model will be stored.

Note: Database alias in VI-CarRealTime are defined by .cdb directory name. In order to avoid problems when using models in VI-CarRealTime it is suggested to follow the same rule when registering Adams Car databases through acar.cfg files.

- **Setup**

it opens [suspension analysis setup](#).

- **Skip compliance analyses**

flag to skip compliance evaluation analyses.

- **Run Long. Dynamics Ident.**

flag to run the longitudinal DOF characterization analyses.

- **Remove Analysis files**

flag to remove the analysis files stored in working directory at the end of export process.

- **Create loadcase files**

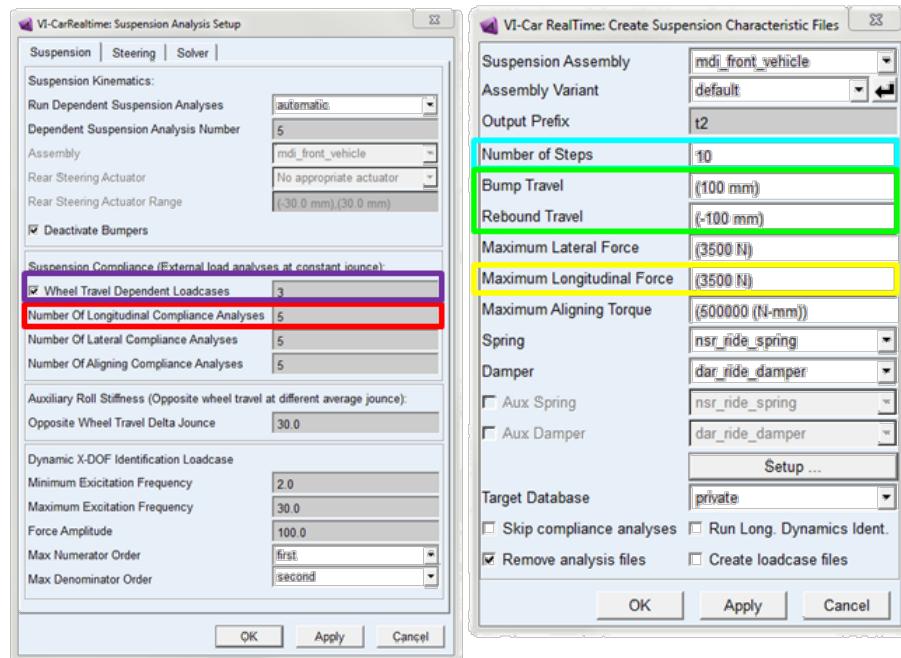
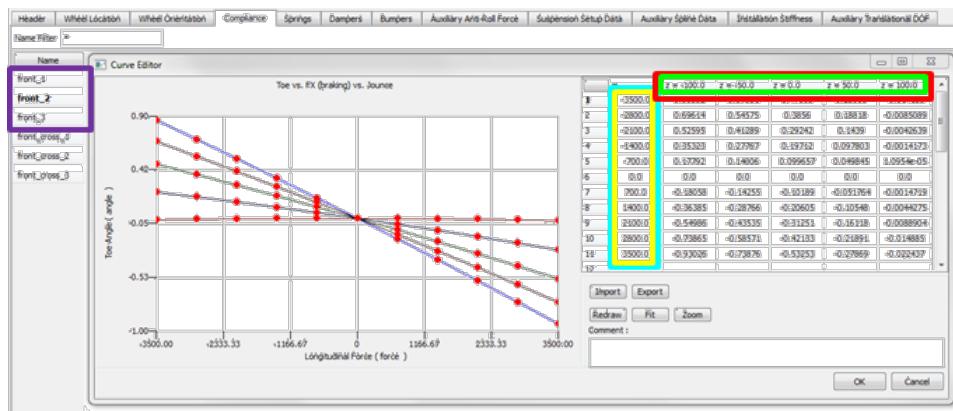
flag to create a loadcase file to be used in VI-CarRealTime Suspension Testrig to replicate Adams Car analyses (for model validation).

- **Write Export CMD File**

flag to write in the working directory the command file (its name is the Suspension Assembly name) with the macro commands used to export the suspension. The file can be used to repeat the export with exactly the same parameters that have been defined through the dialog box. To execute it in Adams session, just press F2 button and select the command file.

The interaction among chosen parameters and vehicle data is shown in the figure below:

VI-CarRealTime Adams Interface



2.2.2 Sequence of Analyses

The sequence of analyses performed to extract data from the suspension is:

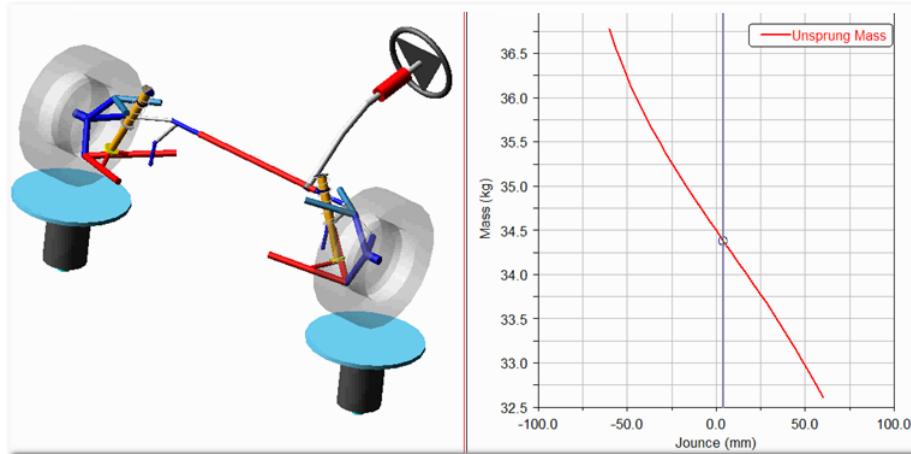
- [Unsprung Mass Evaluation](#)
- [Kinematic analyses](#)
- [Opposite Travel analyses](#)
- [Compliance analyses](#)
- [Dynamic analysis](#)

When the suspension minor role is front and there is a steering subsystem a further set of analyses is run:

- [Steering kinematics](#)
- [External Load analyses \(compliance\)](#)
- [External Load analyses \(steering\)](#)

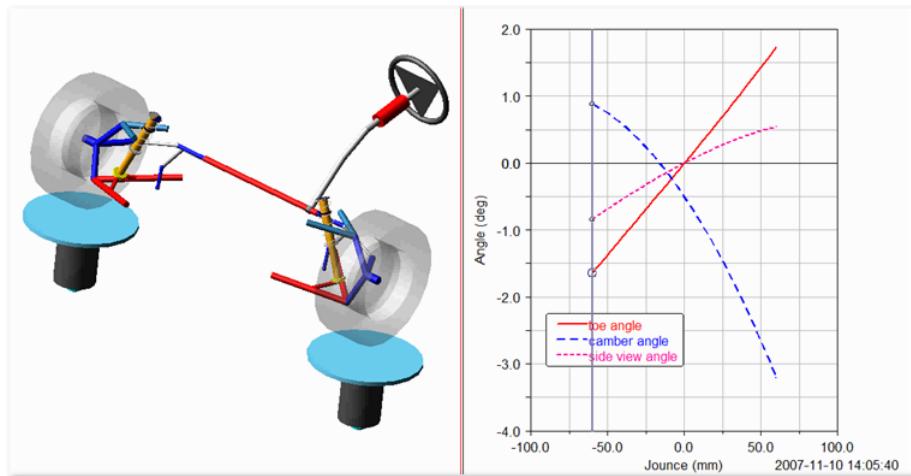
Unsprung Mass Evaluation

The analysis, that is needed to estimate the sprung mass, is a parallel wheel travel run with and without gravity. The difference in the wheel vertical load allows to create a curve with the dependency of unsprung mass on wheel jounce. These data are stored in the database and may be used when extracting a full vehicle (when a static analysis is run on the full vehicle to evaluate suspension jounce, used to get the correct value from the unsprung mass curve).



Kinematic analyses:

Parallel travel (for independent suspension) or set of single travel analyses at fixed jounce for the paired wheel.



Auxiliary stiffness analyses (roll and vertical):

Two sets of single wheel travel analyses (using a set of values for paired wheel travel are swept through the vertical travel range) are executed using active and inactive anti roll bar in order to populate relative splines in suspension subsystem. The deactivation of anti-roll bar is realized by using the first viable option among the ones listed here:

1. switching off the anti-roll bar subsystem (if existing);
2. setting the **pvs_antirollbar_active** variable (if existing) in the suspension subsystem to value = 0 under assumption that the state of the variable is toggling the anti-roll bar activity;
3. setting the variable referred by the **arb_stiffness** output communicator from suspension subsystem (if existing) to value = 0; original value will be restored at the end of the procedure.

Note: the export procedure evaluates in the order presented which is a viable option and performs whichever is feasible first.

Note: in case none of the conditions listed above applies, the auxiliary roll value extracted will be null and all the effects of auxiliary forces will be included in the [auxiliary vertical forces maps](#).

The auxiliary anti-roll force is computed as difference of the hub vertical forces measured in the corresponding load cases (active and inactive anti roll) according to the equation:

$$AuxRollForceZ = AuxForce_{ARBActive} - AuxForce_{ARBInactive}$$

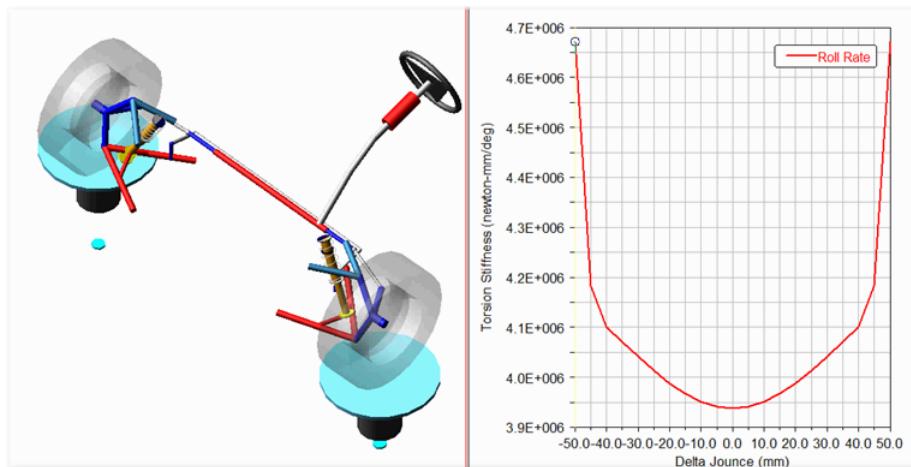
where the auxiliary force for each load case (and side) is computed as follows:

$$AuxForceZ = Hub_{FZ} - Jacking_{FZ} - UnsprungPart_{GravityFZ} - Spring_F * Spring_{MRatio} \\ - Bumpstop_F * Bumpstop_{MRatio} - Reboundstop_F * Reboundstop_{MRatio}$$

where the jacking force if retrieved from the following equation:

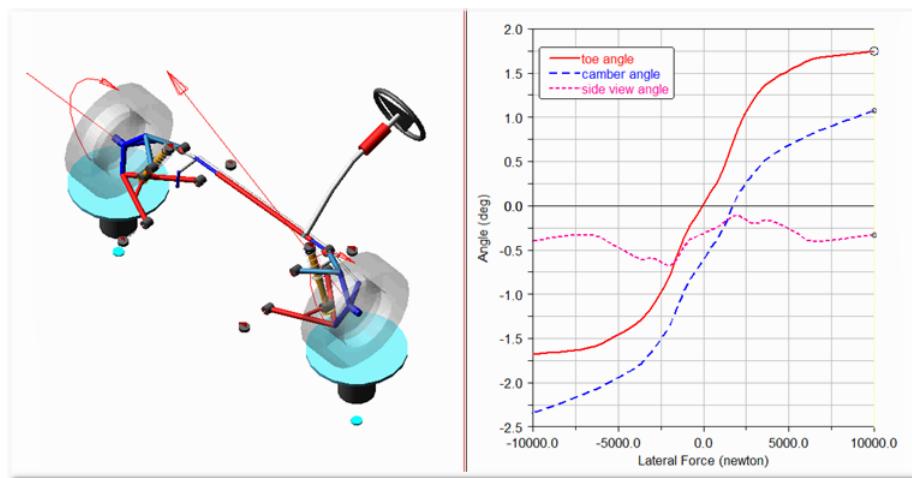
$$Jacking_{FZL,R} = \sum_{L,R} T_x * \frac{\partial Camber}{\partial Jounce} + \sum_{L,R} T_y * \frac{\partial SVA}{\partial Jounce} \dots$$

Note: the auxiliary anti roll force can be described either as a force on the wheel (default) or as rotational spring component by motion ratio and component properties (torsional stiffness or torque characteristic). In the latter case, in order to export the feature in VI-CarRealTime model the presence of a communicator, referring to the torque used to describe the ARB component, is needed (matching name=**cos_ARB_force**, type=force).



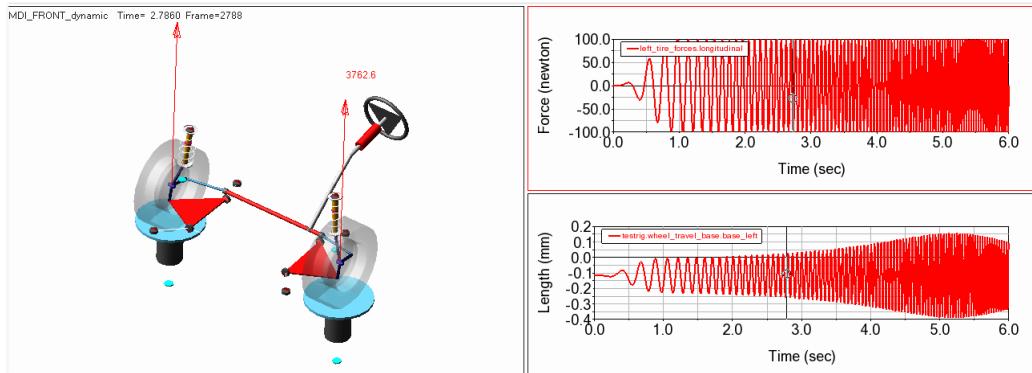
Compliance analyses:

A set of analyses to estimate suspension direct and cross compliance. See suspension compliance documentation to understand the kind of analyses run.



Dynamic analyses:

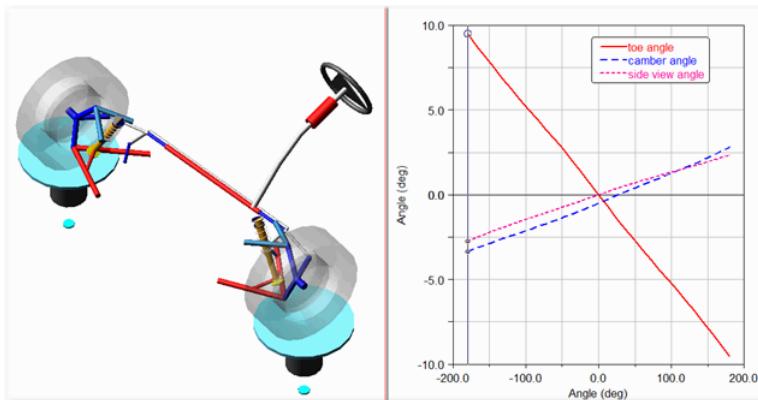
A dynamic analysis is run to characterize the longitudinal degree of freedom of the suspension. See [Dynamic X-DOF](#) topic to understand the kind of analysis run.



When the suspension minor role is front and there is a steering subsystem a further set of analyses is run:

Steering kinematics:

Steering analyses at different wheel jounce.

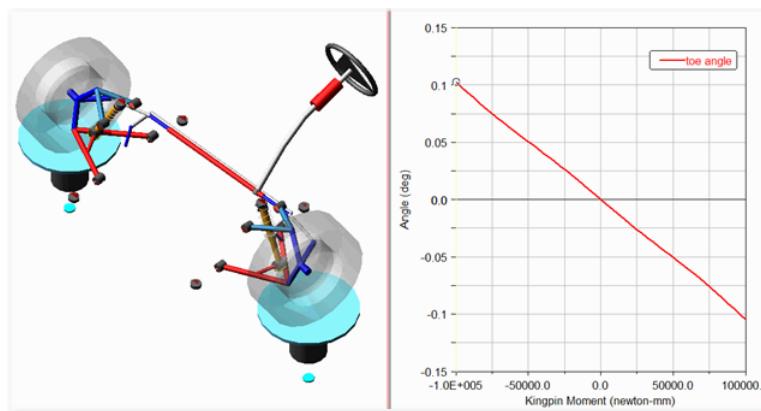


External Load analyses:

To evaluate steering wheel column compliance.

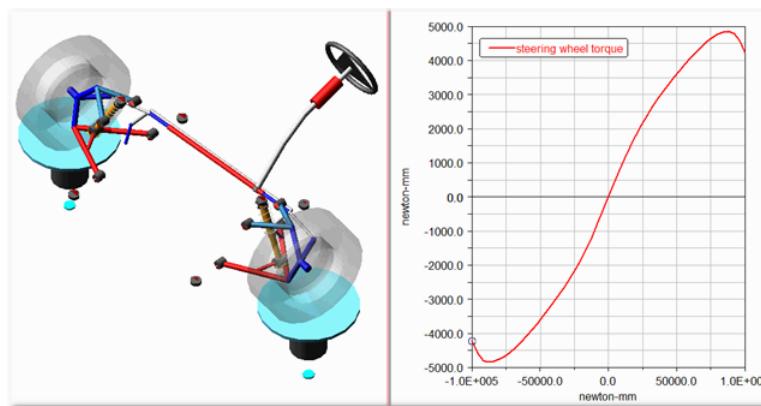
If a concentrated compliance element is present in the steering system, the compliance characteristic can be estimated and mapped in a specific lookup table expressing the steering column deformation as a function of total kingpin moment input in the steering system. In order to support such feature an output communicator that is referring to the force element for the steering column compliance (name=**cos_compliance_force**, type=force) must be present in the steering template. A second compliance element assumed to be in series to first one is optionally supported by the same communicator mechanism (name=**cos_aux_compliance_force**, type=force).

Note: a very good model characterization is obtained also without these communicators, since in such case the steering compliance contribute is automatically embedded in both steering and suspension splines. The usage of these communicators is recommended only when the aim is to have steering column deformation characterized into a specific spline ([steering compliance](#)).



External Load analysis

To evaluate steering wheel/rack travel effort:



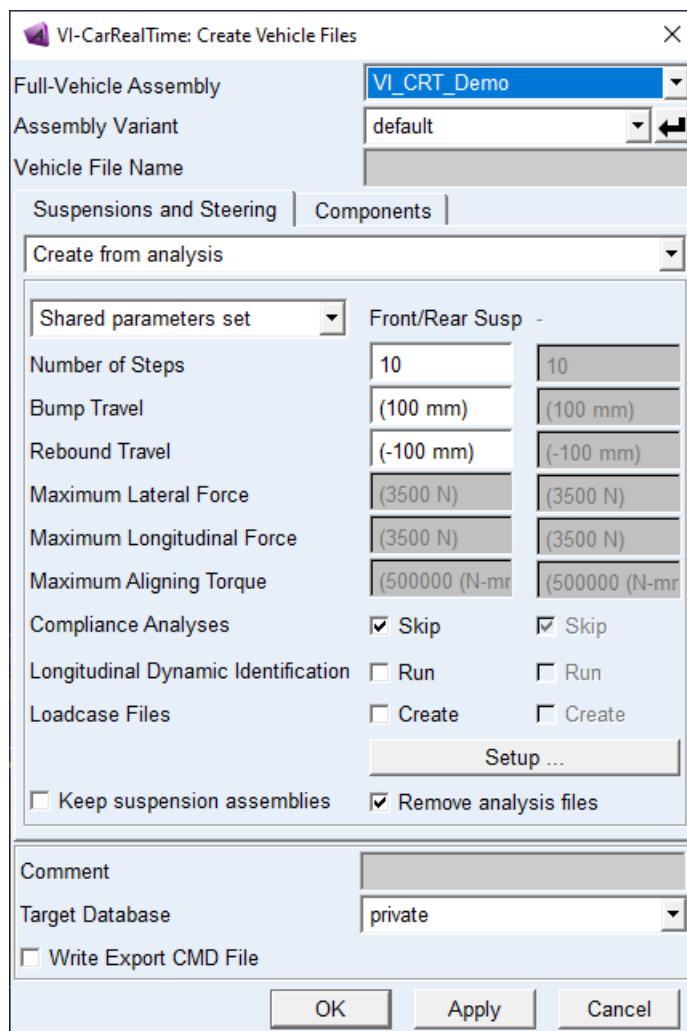
Note: The locations of the tie-rod inner and outer hard point can be taken into account by the export procedure in order to define the line of action of both tie-rod parts. These locations are used by VI-CarRealTime solver to compute run-time the X,Y,Z values of the [tie-rod force](#). Two symmetric (left/right) output communicators, the first couple referring to the inner location of the tie-rod parts (name=**co[l|r]_tierod_inner**, type=location) and the second couple referring to the outer location of the tie-rod parts (name=**co[l|r]_tierod_outer**, type=location) must be present in the suspension template.

As an alternative it is possible to create specific maps for tie-rod forces as function of rack travel and kingpin moment; the force is measured at kinematic joints or bushings connecting suspension tie-rods to steering subsystem so the Adams Car vehicle model needs to be prepared to pass the information to the VI-CarRealTime plugin; there are four options:

- tie-rod joints belonging to steering subsystem: a design variable of type object named **VICRT_tierod_joints** must exists in steering subsystem template; the variable will have two elements: the first is pointing to the left tie-rod joint the second to right one;
- tie-rod joints belonging to other subsystem (ex. suspension): an input communicator of type joint named **tierod_joint** must exist in steering subsystem and matched with a corresponding output one. The communicator will point to tie-rod joints.
- tie-rod bushings belonging to steering subsystem: a design variable of type object named **VICRT_tierod_bushings** must exists in steering subsystem template; the variable will have two elements: the first is pointing to the left tie-rod joint the second to right one;
- tie-rod bushings belonging to other subsystem (ex. suspension): an input communicator of type joint named **tierod_bushing** must exist in steering subsystem and matched with a corresponding output one. The communicator will point to tie-rod bushings.

2.3 Create Vehicle Model

The Create Vehicle Model dialog box is used to create vehicle data for VI-CarRealTime from an existing assembly.



Full Vehicle Assembly

Name of the full vehicle assembly to be exported.

Vehicle File Name

Name of the system file to be created.

Suspension and Steering Tab

It contains data to be used during suspension and steering parameters extraction analyses.

Model Creation Method

Supported options are:

- Create from Analyses: creates front and rear suspension assemblies and extracts data;
- No Suspension and Steering: creates an empty model;
- Use Existing: creates from existing suspension and steering subsystems and retrieves other vehicle data from database.

"Create from Analyses" Parameters

Available options are:

- Shared parameters set: same analysis parameters are used for front and rear suspension parameters.
- Independent parameters: different parameters are used for front and rear suspensions.

Number Of Steps

Number of steps for each analysis; corresponding to the number of values for the first independent variable in the suspension curves.

Bump Travel

Jounce upper limit for suspension analyses.

Rebound Travel

Jounce lower limit for suspension analyses.

Maximum Lateral Force

Maximum force value used for longitudinal compliance evaluation (force ranges from -MaxForce to +MaxForce)

Maximum Longitudinal Force

Maximum force value used for lateral compliance evaluation (force ranges from -MaxForce to +MaxForce).

Maximum Aligning torque

Maximum torque value used for aligning compliance evaluation (torque ranges from -MaxTorque to +MaxTorque).

Target Database

Database where the exported model will be stored.

Write Export CMD File

Flag to write in the working directory the command file (its name is the Full Vehicle Assembly name) with the macro commands used to export the suspension. The file can be used to repeat the export with exactly the same parameters that have been defined through the dialog box. To execute it in Adams session, just press F2 button and select the command file.

Note: Database alias in VI-CarRealTime are defined by .cdb directory name. In order to avoid problems when using models in VI-CarRealTime it is suggested to follow the same rule when registering Adams Car databases through acar.cfg files.

Setup

It opens [suspension analysis setup](#).

Keep suspension assemblies

Flag to retain into Adams Car session suspension assemblies created during full vehicle export.

Loadcase Files

Flag to create the loadcase files of the export analyses in XML format. The loadcases can be used in VI-CarRealTime Suspension Testrig to replicate Adams Car analyses (for model validation).

Compliance Analyses

When the flag is checked the compliance analyses are skipped. When it's unchecked, the user is allowed to set ranges for longitudinal and lateral forces and aligning moment.

Longitudinal Dynamic Identification

When the flag is checked, an additional dynamic analysis is performed to excite the longitudinal dynamic of the suspension in order to characterize its longitudinal behavior.

Components Tab

Lets you select spring and damper characteristics to be used in VI-CarRealTime suspension subsystems.

Spring(Front/Rear)

Spring component to be exported in VI-CarRealTime suspension subsystem.

Damper(Front/Rear)

Damper component to be exported in VI-CarRealTime suspension subsystem.

Aux Spring(Front/Rear)

Auxiliary spring component to be exported in VI-CarRealTime subsystem (up to two spring pair per suspension is supported). The component will be considered in parallel to other spring. The effects of other springs will be included in auxiliary vertical stiffness.

Aux Damper(Front/Rear)

Auxiliary damper component to be exported in VI-CarRealTime subsystem (up to two damper pair per suspension is supported) The component will be considered in parallel to other damper.

2.3.1 Parameter Extraction

During the parameter extraction, when using the create from analysis option (which is the most general case) the following operations are performed:

1. Creation of front suspension subsystem (including steering);
2. Front suspension parameter extraction;
3. Creation of rear suspension subsystem;
4. Body subsystem parameter extraction (including full vehicle static analysis and sprung mass evaluation);
5. Front and Rear Wheels subsystems parameter extraction (including unsprung mass and inertia values);
6. Powertrain subsystem parameter extraction;
7. Brakes subsystem parameter extraction.

Please refer to the [suspension data export](#) procedure for information about how it is possible to instrument the suspension and steering templates to transfer a complete set of information from Adams Car model into VI-CarRealTime.

Sprung Mass Evaluation

Sprung mass is evaluated using the following equation:

$$M_s = M_t - m_{l1} - m_{l2} - m_{r1} - m_{r2}$$

where:

- M_s is the sprung mass
- M_t is the vehicle total mass
- $m_{l1}, m_{l2}, m_{r1}, m_{r2}$ are corners unsprung masses, whose dependency on jounce has been evaluated during suspension analyses and final value has been determined by full vehicle static analysis.

The longitudinal position of sprung mass CG is computed as:

$$X_g = [(F_{tl2} + F_{tr2}) - (m_{l2} + m_{r2}) \cdot g] \cdot \frac{b}{M_s \cdot g}$$

where:

- X_g is longitudinal position of sprung mass CG;
- F_{tl2}, F_{tr2} are rear tire vertical forces;
- b is vehicle wheelbase;
- g is gravity.

Vertical position of sprung mass CG is obtained by the equivalence of vehicle CG for Adams Car model (obtained through aggregate mass utility) and the vehicle CG of simplified model (5 parts). The following equation is used:

$$Z_s = \frac{(M_s \cdot z_{sagg} + (m_{l1agg} \cdot z_{l1agg} + m_{l2agg} \cdot z_{l2agg} - m_{l1} \cdot z_{l1} - m_{l2} \cdot z_{l2} + m_{r1agg} \cdot z_{r1agg} + m_{r2agg} \cdot z_{r2agg} - m_{r1} \cdot z_{r1} - m_{r2} \cdot z_{r2}))}{M_s}$$

where the subscript s stands for "sprung" and agg stands for "aggregate".

Inertia moments are computed from the values obtained from aggregate mass for sprung parts (body and powertrain) and then scaled by a factor equal to the ratio between sprung mass value and aggregate mass value.

When the body is split for the torsional compliance implementation the same evaluation is done for front and rear body halves. To do it during the full vehicle static analysis the vertical reaction is evaluated on the constraint splitting the body part.

The following equations are applied:

$$M_{s1} = \frac{(F_{tl1} + F_{tr1} + F_c - (m_{l1} + m_{r1}) \cdot g)}{g}$$

$$x_1 = \frac{(F_c \cdot X_c + T_c)}{M_{s1} \cdot g}$$

$$M_{s2} = \frac{(F_{tl2} + F_{tr2} + F_c - (m_{l2} + m_{r2}) \cdot g)}{g}$$

$$x_2 = \frac{(F_c \cdot (b - X_c) + T_c)}{M_{s2} \cdot g}$$

where:

- F_c and T_c are the constraint vertical force and moment;

- M_{s1} and M_{s2} are front and rear sprung masses;
- x_1 and x_2 are front and rear sprung mass CG longitudinal positions;
- X_c is longitudinal position of split constraint.

Vertical positions are assumed to be coincident with full sprung mass CG.
Inertia are scaled consequently to mass distribution.

Note: Adams Car uses high stiffness springs to lock the wheels during the static analysis for better solver convergence, which may lead to altered values of the tire vertical loads in the sprung mass evaluation. Please check carefully for the presence of such forces in your models and reduce the stiffness values to the minimum sufficient to obtain statics convergence when needed.

Powertrain parts

It is possible to instruct VI-CarRealTime plug-in to export the powertrain parts by including in the powertrain template a group named `powertrain_parts` and having as objects the list of the parts constituting the powertrain. The plugin will detect automatically the bushings connecting the powertrain to the chassis and will create equivalent instances in VI-CarRealTime.

Powertrain rods

It is possible to instruct VI-CarRealTime plug-in to export the powertrain stabilizer bars (rods) by including in the powertrain template a group named `engine_rods` and having as objects the list of the parts constituting the powertrain.

Front Body parts / Rear Body parts

VI-CarRealTime plug-in can identify the front body parts by including in the body template (template having role="body") a group named `front_body_parts` and having as objects the list of the parts constituting the front body block. The plugin automatically computes the aggregate of such parts, characterizing a unique rigid part having mass, cm and inertia properties equivalent to all the parts stored in the group.

Such part becomes the Front [Rigid Part](#) in VI-CarRealTime body subsystem.

The same concept applies for the rear body parts when a group named `rear_body_parts` exists in the body template. The equivalent rigid body characterized becomes the Rear [Rigid Part](#).

Front Chassis parts / Rear Chassis parts

VI-CarRealTime plug-in can identify the front chassis parts by including in the body template (template having role="body") a group named `front_chassis_parts` and having as objects the list of the parts constituting the front chassis block. The plugin automatically computes the aggregate of such parts, characterizing a unique rigid part having mass, cm and inertia properties equivalent to all the parts stored in the group.

Such part becomes the Front [Sprung Mass](#) part in VI-CarRealTime body subsystem.

The same concept applies for the rear chassis parts when a group named `rear_chassis_parts` exists in the body template. The equivalent rigid body characterized becomes the Rear [Sprung Mass](#) part.

Furthermore, the plugin automatically detects the bushings connecting the front chassis parts to the Front Body parts (and the rear chassis parts to the Rear Body parts) and creates the equivalent instances in VI-CarRealTime. The plug-in places all the identified bushing instances in the [Bushings](#) panel of VI-CarRealTime body subsystem.

Kingpin Moment Elements Contribute

VI-CarRealTime plug-in automatically identifies the I and J marker of the elastic elements (spring, damper, bumpstop, reboundstop) in order to export their location in [Kingpin Moment Contribute](#) section of the steering subsystem.

When the `active` flag is checked in the steering component panel, the line of action between I and J marker will be used by VI-CarRealTime solver to compute the kingpin moment contribute due to the element force.

Since in VI-CarRealTime the suspension is conceptual, in case the user would like to specify a different line of action for the element force (for example for suspension with pushrod and rocker, the element force passes through pushrod before reaching the upright, so the pushrod outer and inner markers have to be considered respectively as element I and J marker), it's possible to rely on the output communicators below. The same concept applies for anti-roll bar elements: the user can define, by using communicators, the two locations (upright side and frame side) of left/right droplinks.

- **co[lr]_spring_upright_marker:** matching_name = *spring_upright_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_spring_frame_marker:** matching_name = *spring_frame_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_damper_upright_marker:** matching_name = *damper_upright_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_damper_frame_marker:** matching_name = *damper_frame_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_bumpstop_upright_marker:** matching_name = *bumpstop_upright_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_bumpstop_frame_marker:** matching_name = *bumpstop_frame_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_reboundstop_upright_marker:** matching_name = *reboundstop_upright_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_reboundstop_frame_marker:** matching_name = *reboundstop_frame_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_antirollbar_upright_marker:** matching_name = *antirollbar_upright_marker* , type = *marker* , symmetry = *left/right*
- **co[lr]_antirollbar_frame_marker:** matching_name = *antirollbar_frame_marker* , type = *marker* , symmetry = *left/right*

Note: when such communicators are defined for a component, the plugin automatically assumes that the component acts on the upright, so the [active](#) flag will be automatically checked in the exported model and the element force will contribute to generate kingpin moment. Otherwise, the splines will be anyway exported but the active flag will be set to *off*, assuming that the elastic element does not contribute in generating kingpin moment.

Differential models

Information about differential type adopted in Adams Car is exported automatically for models created in VI-Automotive. For standard Adams Car assemblies the same data can be communicated using a parameter variable named **pvs_differential_type** in powertrain subsystem. Available options recognized for parameter variable string value are:

- open
- LSD
- locked

In case of viscous differential the presence of a general spline named *gss_differential* in the powertrain template is used to retrieve the values of differential torque expressed by the differential.

In order for VI-CarRealTime solver to compute [reaction torques](#) on suspension and chassis part generated when drive-shaft is not aligned with spindle axis, the location of both inner and outer tripods must be defined. Such locations are characterized by means of output communicators:

- output driveshaft location
type: **location**
matching name: **driveshaft_to_spindle**
symmetry: **left/right**
- inner driveshaft location
type: **location**
matching name: **tripot_to_differential**
symmetry: **left/right**

Note: these output communicators are looked for in templates having as major role: "engine", "powertrain", "driveline", "suspension".

Cab part

For vehicle having a cab part connected to the chassis with bushings and/or other elastic elements (i.e. beam), Adams Car Plug-in is able to extract the cab part in an auxiliary subsystem and set-up the VI-CarRealTime model in order to manage the connection between the cab and the chassis part. Mass distribution of the extracted VI-CarRealTime model is then updated according to the presence of the new element.

This feature is automatically switched on when the Adams assembly has a subsystem with *major role* = **cab**.

VI-CarRealTime plug-in supports a single type of cab suspension: for further details about the cab model please refer to the [cab auxiliary subsystem](#) topic.

Rack-Pinion Steering

The plug-in can export a properly instrumented steering template and characterize its parameters into a [rack-pinion steering](#) XML subsystem for VI-CarRealTime. An example of such template is shipped with VI-CarRealTime installation package: \$VICRT_install_dir/acarrt/examples/SharedModels/Crossover.cdb/_rack_pinion_pfeffer.tpl

The template defines a special variable called *CRT_AdvSteer_Verified* which activates the rack-pinion steering export during the model conversion process. In particular, the rack-pinion export procedure looks for specific objects and variables defined in the template in order to identify and characterize the steering parameters (kinematic hardpoints, parts mass and inertia, steering column compliances, cardan angles, steering assist, frictions, etc).

Brakes

Brake parameters are extracted by looking for particular parameter variables in brake template. If that particular parameter variable does exist, its value is used to set the related value in VI-CarRealTime brake xml subsystem, otherwise a default value is set and a warning is raised.

Here a table showing all the acceptable pvs names for each brake parameter and the default value set when the pvs is not found. Additional information can be found in export report log file, automatically created in the working directory.

brake parameter	acceptable parameter variable name	default value
front brake mu	pvs_front_brake_mu, pvs_Reibwert_vorne	0.4
front effective	pvs_front_effective_piston_radius, pvs_Radius_effektiv_vorne, pvs_Wirkradius_vorne,	130 mm
piston radius	pvs_Kolbenflaeche_vorne	
front brake bias	pvs_front_brake_bias	0.5
front piston area	pvs_front_piston_area, pvs_Kolbenflaeche_vorne, pvs_wirksame_kolbenflaeche_einseitig_vorne	2500 mm ²
rear brake mu	pvs_rear_brake_mu, pvs_Reibwert_hinten	0.4
rear effective	pvs_rear_effective_piston_radius, pvs_Radius_effektiv_hinten,	130 mm
piston radius	pvs_Wirkradius_hinten, pvs_Kolbenflaeche_hinten	
rear piston area	pvs_rear_piston_area, pvs_Kolbenflaeche_hinten, pvs_wirksame_kolbenflaeche_einseitig_hinten	2500 mm ²
master cylinder	force_to_pressure_cnvt, pvs_max_brake_line_pressure,	0.1
pressure gain	pvs_bremsdruck_verstaerkungsfaktor	

Note: for additional insights about VI-CarRealTime brake model, please refer to [About External Brake Model](#) topic.

2.3.2 Creating Multiple Body Sensors

Multiple [sensor](#) instances can be created, when exporting a vehicle model by setting the environment variable *VICRT_GYRO_SENSORS=1*.

In Adams you can do that by executing the following command in the command line:

```
variable set variable_name = tmpvar integer_value =(eval(PUTENV("VICRT_GYRO_SENSORS","1")))
```

The vehicle will then contain the following additional sensors:

- gyro
- driver
- sensor_point_1
- sensor_point_2
- sensor_point_3

Outputs

The main sensor (sensor_point_1) output is written into the Sensor result set, with the following components:

s stand for sensor marker.

b stands for body chassis cm marker.

g stands for ground.

gyr stands for [gyro marker](#).

Global_X = DX(s,g,g)

measures the x component of s displacement relative to ground expressed in ground/global coordinates

Global_Y = DY(s,g,g)

measures the y component of s displacement relative to ground expressed in ground/global coordinates

Global_Z = DZ(s,g,g)

measures the z component of s displacement relative to ground expressed in ground/global coordinates

VX = VX (s,g,s)

measures the x component of s velocity relative to ground expressed in s

VY = VY (s,g,s)

measures the y component of s velocity relative to ground expressed in s

VZ = VZ (s,g,s)

measures the z component of s velocity relative to ground expressed in s

AX = ACCX(s,g,s)

measures the x component of s acceleration (including gravity) relative to ground expressed in s

AY = ACCY(s,g,s)

measures the y component of s acceleration (including gravity) relative to ground expressed in s

AZ = ACCZ(s,g,s)

measures the z component of s acceleration (including gravity) relative to ground expressed in s

AX_without_gravity = ACCX(s,g,gyr)

measures the x component of s acceleration relative to ground expressed in gyr

Ay_without_gravity = ACCY(s,g,gyr)

measures the y component of s acceleration relative to ground expressed in gyr

Az_without_gravity = ACCZ(s,g,gyr)

measures the z component of s acceleration relative to ground expressed in gyr

AccX_gyro = AX_without_gravity

(redundant value)

AccY_gyro = AY_without_gravity

(redundant value)

VX_gyro = VX (s,g,gyr)

measures the x component of s velocity relative to ground expressed in gyr

YawRate_gyro = WX (s,g,gyr)

measures the z component of s angular velocity relative to ground expressed in gyr

In the case of multiple sensors the extra sensor instances output is stored under the Gyro result set; the components are:

gyro_X = DX(gyr,g,g)

measures the x component of gyr displacement relative to ground expressed in ground/global coordinates

gyro_Y = DY(gyr,g,g)

measures the y component of gyr displacement relative to ground expressed in ground/global coordinates

gyro_VX = VX(gyr,g,g)

measures the x component of gyr velocity relative to ground expressed in gyr

```

gyro_ACCX = ACCX(gyr,g,g)
gyro_ACCY = ACCY(gyr,g,g)
Driver_X = DX(driver_sensor,g,g)
Driver_Z = DZ(driver_sensor,g,g)
sensor_1_Z = DZ(sensor_1,g,gyr)
sensor_2_Z = DZ(sensor_2,g,gyr)
sensor_3_Z = DZ(sensor_3,g,gyr)

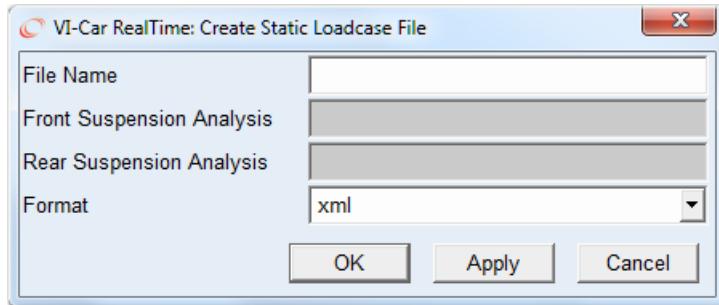
```

measures the x component of gyr acceleration relative to ground expressed in gyr
measures the y component of gyr acceleration relative to ground expressed in gyr
measures the x component of driver marker displacement relative to ground expressed in gyr
measures the z component of driver marker displacement relative to ground expressed in gyr
measures the z component of sensor #1 marker displacement relative to ground expressed in gyr
measures the z component of sensor #2 marker displacement relative to ground expressed in gyr
measures the z component of sensor #3 marker displacement relative to ground expressed in gyr

The gyro reference system (gyro marker) is located on the body and is oriented with the Z axis coincident with global Z during the simulation while the X axis is aligned to global X but follows the body movement, so it follows the vehicle displacement, follows the yaw but not the pitch of the vehicle.

2.4 Create Static Loadcase File

With the Create Static Loadcase File dialog box you can create the input file for Suspension Testrig analysis in VI-CarRealTime from the results of a suspension analyses performed in Adams Car.



File Name

Name of the output file.

Front Suspension Analysis

Analysis to be exported for front suspension.

Rear Suspension Analysis

Analysis to be exported for rear suspension.

Format

Available options are xml and ascii.

2.5 Load VI-Driver plugin

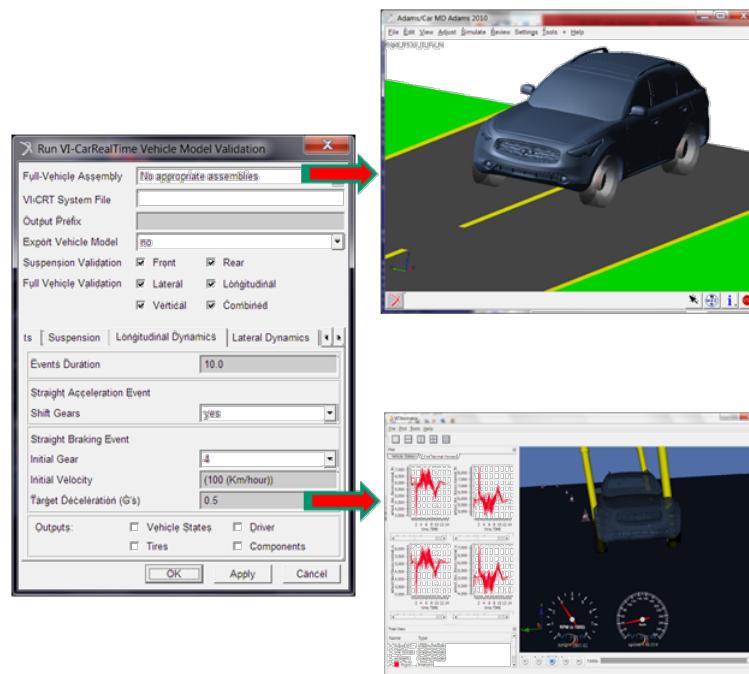
The Load VI-Driver plugin push button enables the use of VI-Driver plugin in Adams/ACar.

VI-Driver plugin makes available to Adams Car users the same driver technology implemented in VI-CarRealTime VI-Driver events; the use is encouraged specially for cross validation purposes, nevertheless Adams Car user will find VI-Driver useful for several full vehicle simulation activities.

2.6 Run Automatic Model Validation

The Automatic Model Validation procedure is a tool allowing to compare side by side the results obtained by a vehicle model in Adams Car and a VI-CarRealTime one.

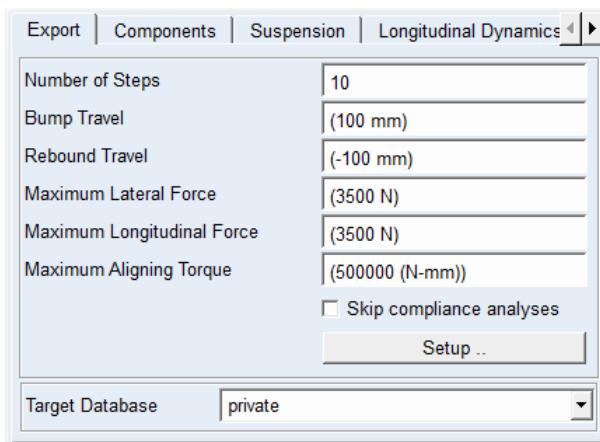
The procedure consists in running equivalent set of analysis in Adams/Car and VI-CarRealTime and comparing results by generation of an automatic report.



The process is modular and can be used to explore the quality of results in different fields:

- [Suspension elasto-kinematics](#)
- [Full vehicle longitudinal dynamics](#)
- [Full vehicle lateral dynamics](#)
- [Full vehicle combined dynamics](#)
- [Full vehicle vertical dynamics](#)

In case the VI-CarRealTime is not existing it can be [created](#) in the context of the procedure.



The following parameters are common to all validation tasks:

- **Full-Vehicle Assembly:** name of the full vehicle assembly present in Adams Car session to be used for comparison;
- **VI-CRT System File:** path to system file of VI-CarRealTime model to be used in the validation process;
- **Output Prefix:** name prefix for the output files generated by the procedure;
- **Export Vehicle Model:** Combo box for optionally activating the Adams Car model export in the validation context;
- **Suspension Validation (Front, Rear):** check boxes for suspension analysis validation;
- **Full Vehicle Validation (Lateral, Longitudinal, Vertical, Combined):** check boxes for full vehicle analysis validation.

At the end of the procedure, after all analyses have been run in Adams Car and VI-CarRealTime an html report is created including the most relevant indexes and al plot comparisons.

Model Type	File Path
A/Car assembly	mdids://carrealtime_shared/assemblies.tbl/VI_CRT_Demo.asy
VI-CarRealTime System	mdids://private/systems.tbl/VI_CRT_Demo_20120203.xml

Validation Task	Executed
Front Suspension	Yes
Rear Suspension	Yes
Steering	Yes
Full Vehicle Dynamics	No

Index	A/Car	VI-CarRealTime
Vehicle Mass	1279.0 kg	1278.9952 kg
Vehicle CG Longitudinal Position	-1567.0197642005 mm	-1566.97451468 mm
Vehicle CG Lateral Position	0.0 mm	0.0 mm
Vehicle CG Vertical Position	408.0860507616 mm	406.970356136 mm

2.6.1 Export

The Export tab of the Run VI-CarRealTime Vehicle Model Validation dialog box is active when the Export Vehicle Model option menu is set to yes.

The following parameters can be defined:

- **Number Of Steps**

number of steps for each analysis; it corresponds to the number of points for the first independent variable of the suspension curves.

- **Bump Travel**

jounce upper limit for suspension analyses.

- **Rebound Travel**

jounce lower limit for suspension analyses.

- **Maximum Lateral Force**

Maximum force value used for longitudinal compliance evaluation (force ranges from -MaxForce to +MaxForce).

- **Maximum Longitudinal Force**

Maximum force value used for lateral compliance evaluation (force ranges from -MaxForce to +MaxForce).

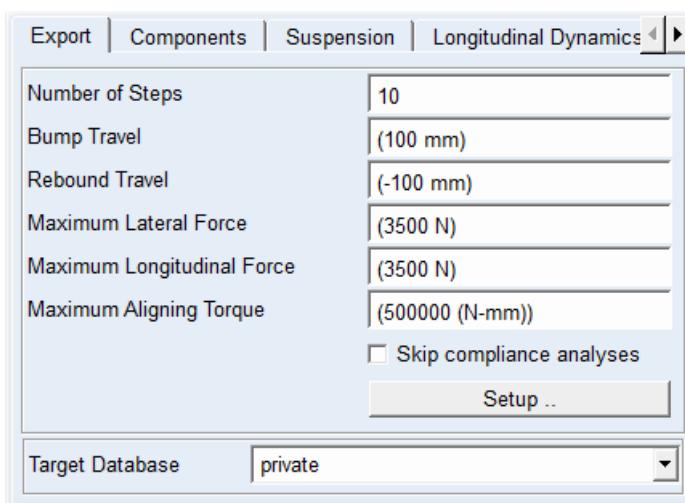
- **Maximum Aligning torque**

Maximum torque value used for aligning compliance evaluation (torque ranges from -MaxTorque to +MaxTorque).

- **Skip compliance analyses**

flag to skip compliance evaluation analyses.

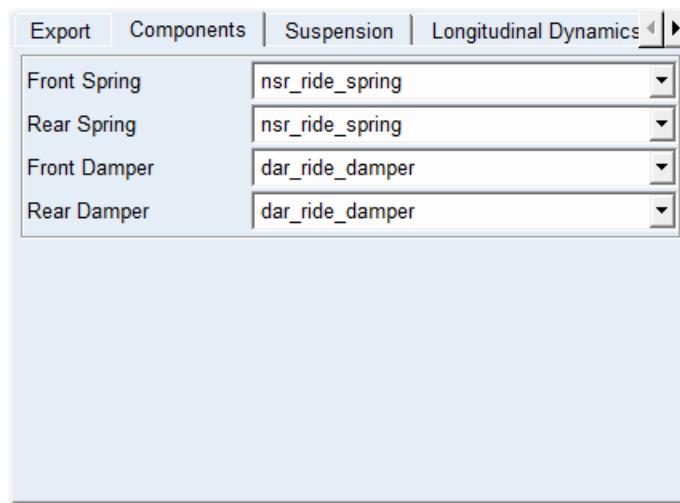
In the case of vehicle export in the context of validation only a common set of parameters is chosen for both front and rear suspensions.



The [Setup](#) button is used to define the export procedure parameters.

2.6.2 Components

The Components tab is used to select the spring and damper components to be used during the export procedure.

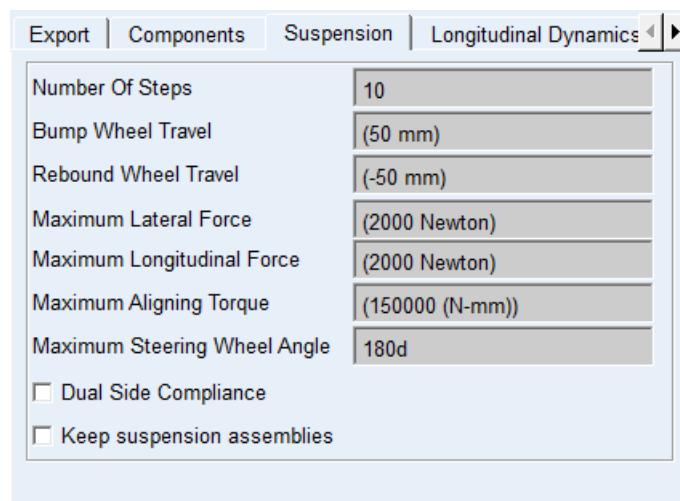


2.6.3 Suspension

Suspension validation task is including the following set of analyses:

- Parallel Wheel Travel;
- Opposite Wheel Travel;
- Longitudinal Force (nominal wheel travel);
- Lateral Force (nominal wheel travel);
- Aligning Torque (nominal wheel travel);
- Steering Sweep (nominal and max wheel travel).

The suspension tab is activated by the Suspension Validation (Front/Rear) check buttons. The tab includes the following parameters to drive the analyses:



- Number Of Steps

number of steps for each analysis.

- **Number Of Steps**

number of steps for each analysis.

- **Bump Travel**

wheel travel upper limit.

- **Rebound Travel**

wheel travel lower limit.

- **Maximum Lateral Force**

Maximum lateral force value (force ranges from -MaxForce to +MaxForce).

- **Maximum Longitudinal Force**

Maximum longitudinal force value (force ranges from -MaxForce to +MaxForce).

- **Maximum Aligning torque**

Maximum aligning torque value (torque ranges from -MaxTorque to +MaxTorque).

- **Maximum Steering Wheel Angle**

Maximum steering wheel angle value (angle ranges from -MaxSteerWheelAngle to +MaxSteerWheelAngle).

- **Dual Side Compliance**

Flag to activate the dual side force application in compliance loadcases. By default (flag inactive), the suspension characteristics are validated applying the force in a single side (i.e. left toe vs. left lateral force, right camber vs. right lateral force).

- **Keep suspension assemblies**

Flag to retain into Adams Car session suspension assemblies created during full vehicle export.

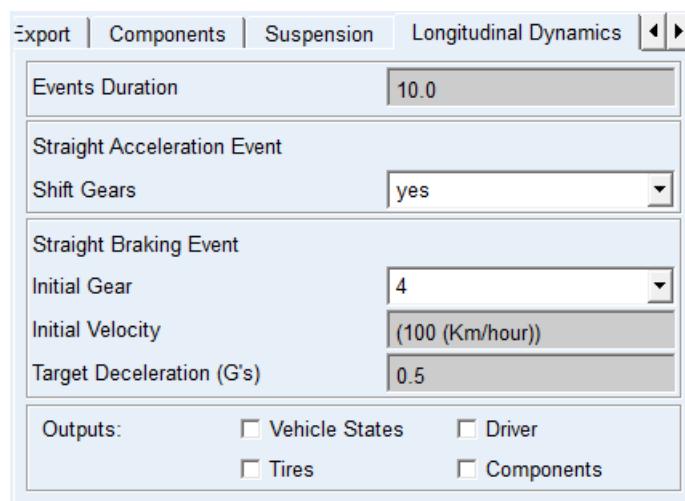
2.6.4 Longitudinal Dynamics

Longitudinal dynamics validation task is including the following set of analyses:

- Straight Line acceleration
- Straight Line braking

The Longitudinal dynamics tab is activated by the Longitudinal check button.

The tab includes the following parameters to drive the analyses:



- **Events Duration**

Time duration of analysis events.

- **Shift Gears**

It defines the option for longitudinal acceleration analysis to include gear shifting.

- **Initial Gear**

It sets the initial gear for braking event.

- **Initial Velocity**

It sets the initial velocity for braking event.

- **Target deceleration**

It sets the target (closed loop) longitudinal acceleration for braking event.

- **Outputs**

Check buttons for defining the channels set to be included in validation report; available flags are: Vehicle States, Driver, Tires, Components.

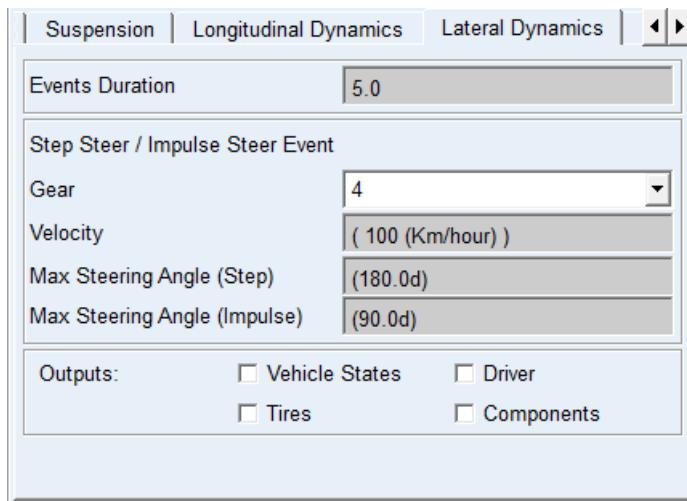
2.6.5 Lateral Dynamics

Lateral dynamics validation task is including the following set of analyses:

- Step Steer
- Impulse Steer

The Longitudinal dynamics tab is activated by the Lateral check button.

The tab includes the following parameters to drive the analyses:



- **Events Duration**

Time duration of analysis events.

- **Gear**

It sets the gear for braking event.

- **Velocity**

It sets the velocity for step and impulse steer events.

- **Max Steering Angle (Step)**

It sets the maximum steering angle for step steer event.

- **Max Steering Angle (Impulse)**

It sets the maximum steering angle for impulse steer event.

- **Outputs**

Check buttons for defining the channels set to be included in validation report; available flags are: Vehicle States, Driver, Tires, Components.

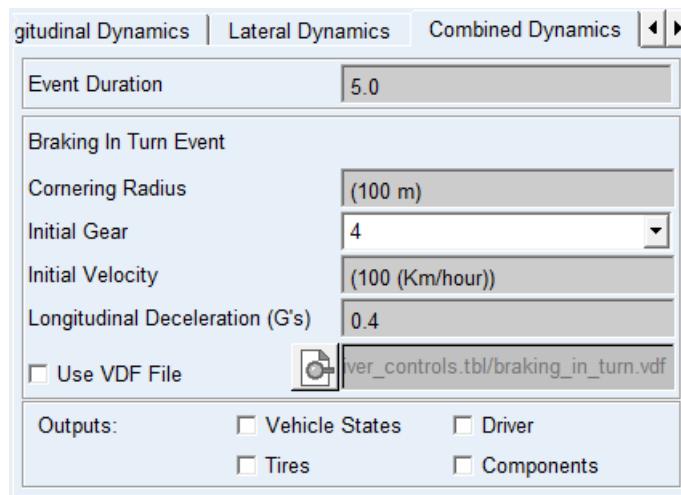
2.6.6 Combined Dynamics

Combined dynamics validation task is including the following analysis:

- Braking In Turn

The Combined dynamics tab is activated by the Combined check button.

The tab includes the following parameters to drive the analyses:



- **Event Duration**

Time duration of analysis event.

- **Cornering Radius**

Turn Radius for the closed loop steering.

- **Initial Gear**

It sets the initial gear for braking event.

- **Initial Velocity**

It sets the initial velocity for braking event.

- **Longitudinal deceleration (G's)**

It sets the target (closed loop) longitudinal acceleration for braking event.

- **Use VDF File**

when the toggle button is checked, it is possible to use a specific VDF File for the Combined event.

- **Outputs**

Check buttons for defining the channels set to be included in validation report; available flags are: Vehicle States, Driver, Tires, Components.

2.6.7 Vertical Dynamics

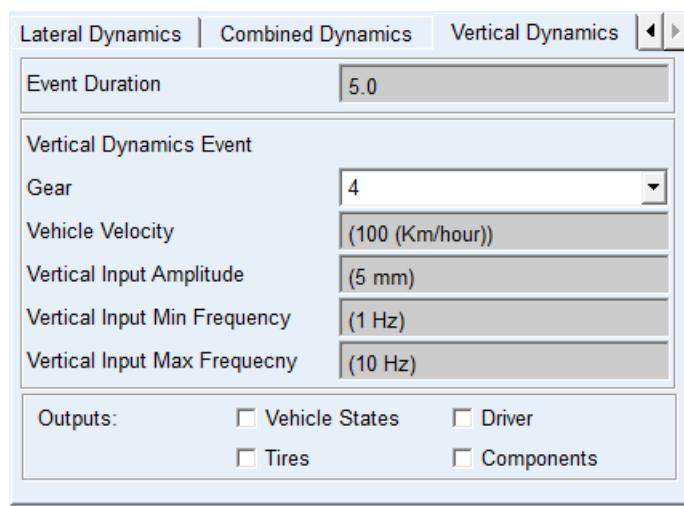
Vertical dynamics validation task is including the following analysis:

- Straight Line on Sinusoidal Disturbance Road

The event consists in a constant velocity maneuver performed on a road with vertical sinusoidal inputs with decreasing wave length.

The Vertical dynamics tab is activated by the Vertical check button.

The tab includes the following parameters to drive the analyses:



- **Event Duration**

Time duration of analysis event.

- **Gear**

It sets the initial gear for the event.

- **Vehicle Velocity**

It sets the target velocity for the event.

- **Vertical Input Amplitude**

It sets the amplitude for vertical road disturbances.

- **Vertical Input Min Frequency**

It sets the minimum frequency for vertical road disturbances.

- **Vertical Input Max Frequency**

It sets the maximum frequency for vertical road disturbances.

- **Outputs**

Check buttons for defining the channels set to be included in validation report; available flags are: Vehicle States, Driver, Tires, Components.

3 VI-CarRealTime HIL Overlays

3.1 VI-CarRealTime SCALEXIO Overlay

3.1.1 Introduction

The VI-CarRealTime Scalexio Overlay is an additional tool that allows the execution of the VI-CarRealTime model on the dSpace Scalexio board.

After the installation, it is possible to build a Scalexio application by using the VI-CarRealTime Matlab/Simulink interface and the dSpace Matlab RTI tools.

To support the VI-CarRealTime solver execution in the Scalexio board, a specific Virtual File System (VFS) and related tools have been developed.

The VI-CarRealTime Scalexio Overlay is provided under a specific board locked license.

[Installation](#)

[Licensing](#)

[Virtual File System](#)

[Limitations](#)

[Tutorial](#)

3.1.2 Installation

Welcome to the installation guide for VI-CarRealTime Scalexio Overlay. VI-CarRealTime Scalexio Overlay is currently for windows only.

In order to perform a complete installation of VI-CarRealTime Scalexio Overlay, you need to pickup the following files from the VI-grade [website support](#) area (registration is required):

- **VI_Crt_scalexio_20_0_x_Setup.exe**: main product installer.
- **VI_Crt_20_0_Installing.pdf**: this document.
- **VI_Crt_20_0_Release_Notes.pdf**: product information.

After reviewing the Release Notes document, the installation could start following the steps below:

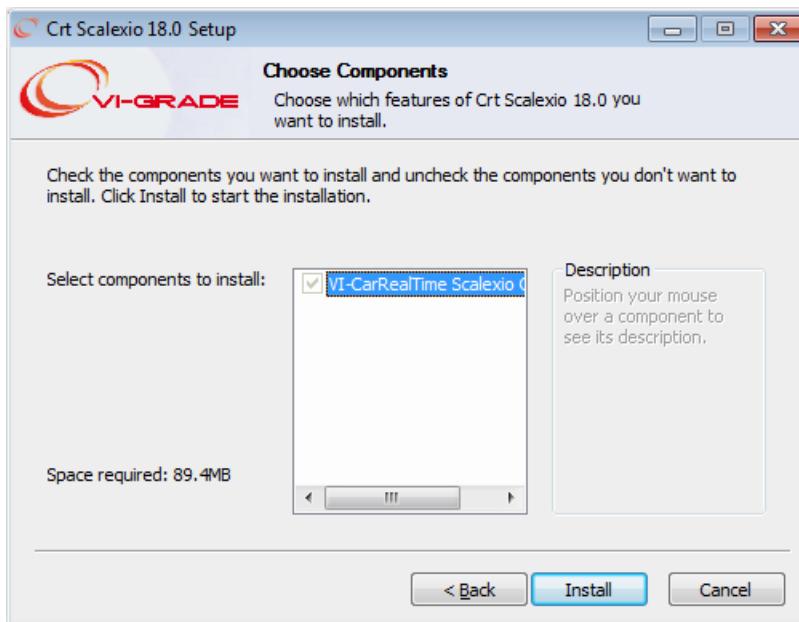
1. Double click over the VI_Crt_scalexio_20_0_x_Setup.exe file, and follow the instructions that will appear on the screen:



The following component will be installed:

- **VI-CarRealTime Scalexio Overlay**

a specific VI-CarRealTime Scalexio compliant library and other support files. Older VI-CarRealTime 20 installation, already available on the target machine will be updated.



At the end of the installation the following window will appear:



Click "Finish" to complete the installation.

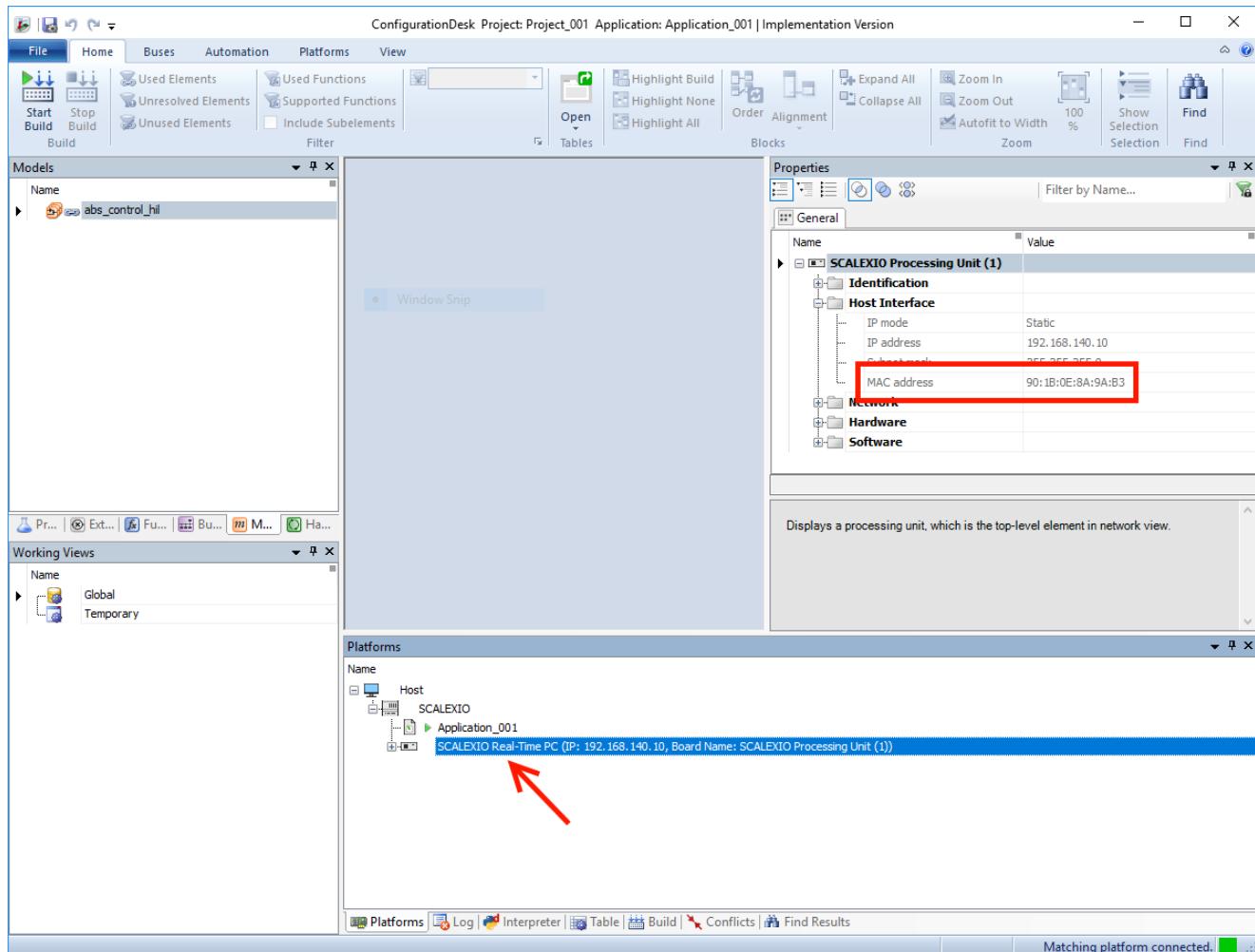
This procedure installs the scalexio directory inside the VI-CarRealTime installation directory (or inside the one selected during the installation phase).

3.1.3 Licensing

In order to execute VI-CarRealTime in the dSpace SCALEXIO board, a board specific license file is needed.

The license file can be retrieved sending to our [licensing support](#) the SCALEXIO board MAC address displayed by the dSpace ConfigurationDesk in the **Properties** right pane under **Host Interface** section.

Note: In order to easily manage the license among more platforms, we can provide a unique license file for all the SCALEXIO platforms on which VI-CarRealTime needs to be run.



3.1.4 Virtual File System

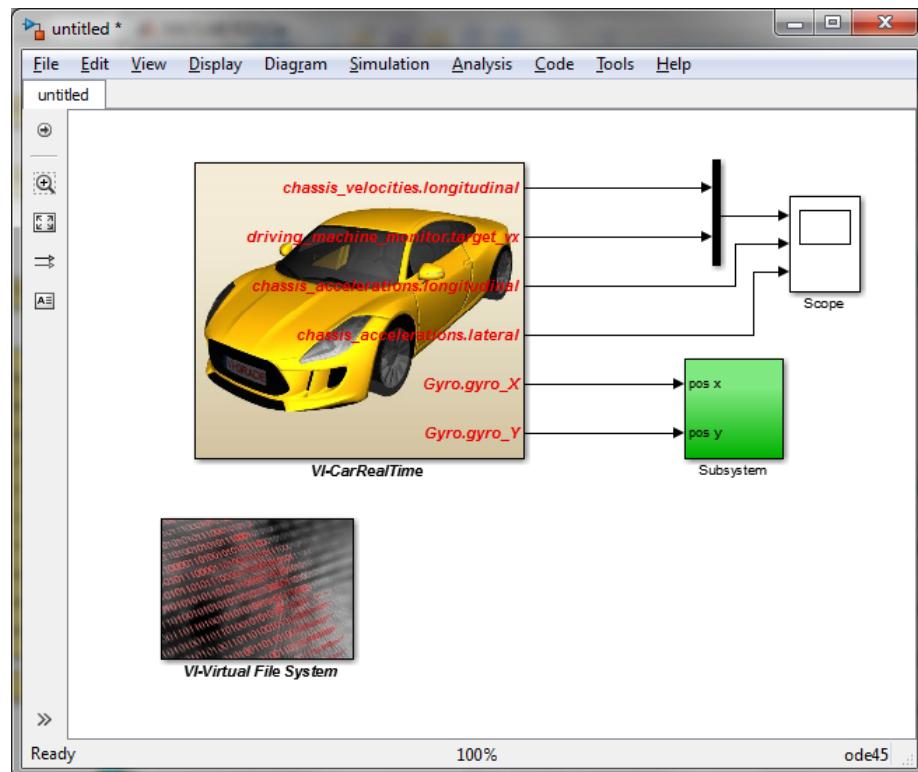
The Virtual File System is an "in memory" binary representation of the files needed by the VI-CarRealTime solver. In order to keep as small as possible the memory consumption due to the Virtual File System package, the building tool creates a package containing only the files opened by the VI-CarRealTime solver to run the desired maneuver.

In order to automate the building procedure and keep the Virtual File System package consistent with the VI-CarRealTime input file, the **"VI-Virtual File System"** block must be placed in the Simulink model.

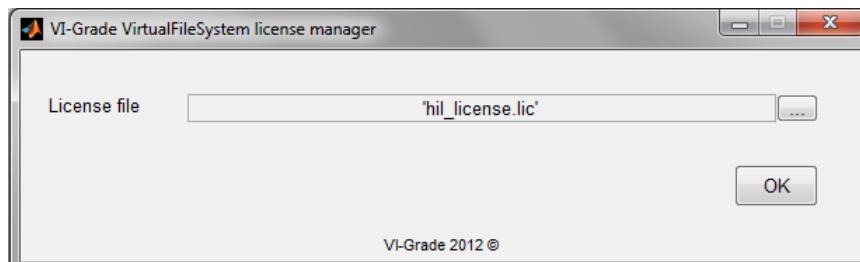
In this way, when the Real Time Workshop building procedure starts, a **"fs_package_archive.o86"** file (representing the Virtual File System in the SCALEXIO **obj** format) is created, and will be linked by the dSpace linker at the end of the process.

Note: The VI-Virtual File System block must be placed in the same subsystem of the VI-CarRealTime Matlab/Simulink solver and can be retrieved from the VI-CarRealTime Simulink Library.

VI-CarRealTime HIL Overlays

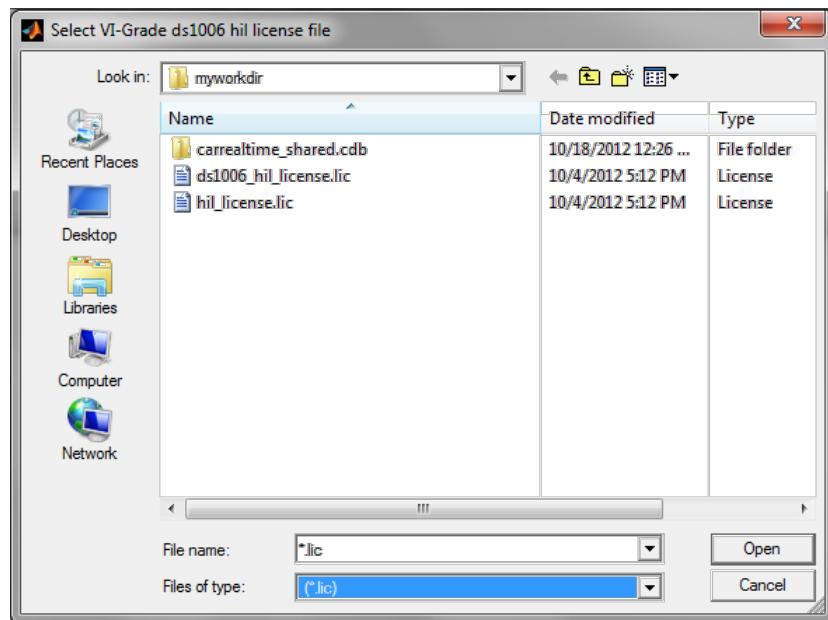


To make it easy to manage the license file, a selection mask is provided by the "**VI-Virtual File System**" block.



The license file can be set in three different ways:

- using a workspace variable, that must be written in the "License file" text field;
- writing the full path of the license file (or the file name if it is in the working dir) in the "input file" text field and pushing the enter button;
- using the file explorer through the button placed near the text field ();



3.1.5 Limitations

The current version of the VI-CarRealTime porting for SCALEXIO does not support the full set of functionalities featured by the Windows version of the software. Please refer to the following table for the unsupported capabilities.

Modelling	Trailer External FMUs Custom dll Custom aero Custom tires FTire MF-Tyre / MF-Swift
Events	Max Performance Press Maneuvers SpeedGen Sevenpostrig
PostProcessing	Live Animation with VI-Animator
VI-Driver	Human controller

3.1.6 Tutorial

ABS Control

This tutorial shows how to run a standard VI-CarRealTime Simulink model on dSpace SCALEXIO board. The Simulink model is a co-simulation between VI-CarRealTime and an external ABS control system. The tutorial is divided into these steps:

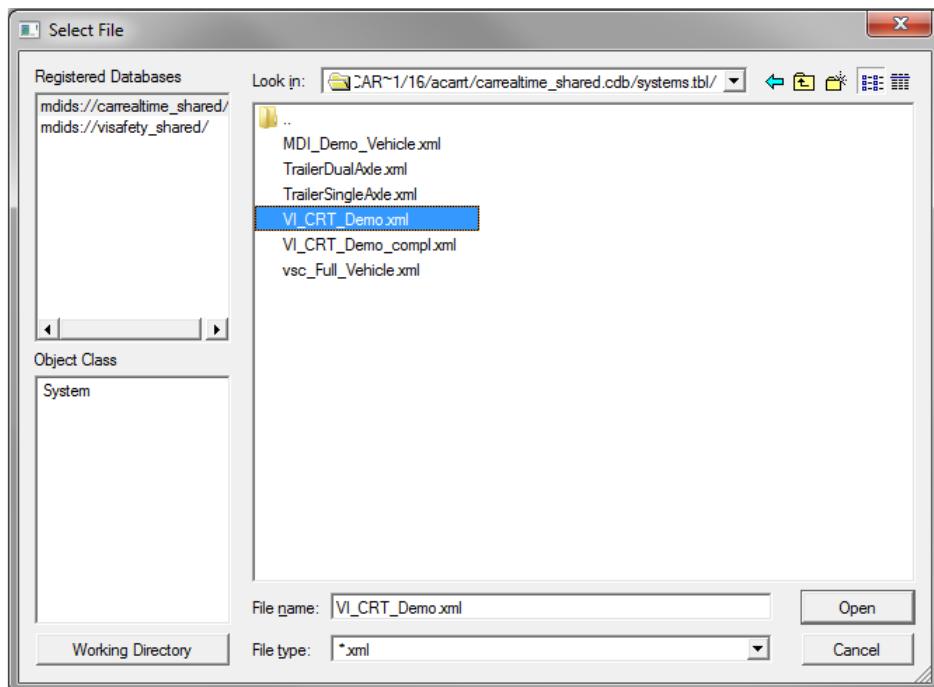
1. Generating the Maneuver file
2. Setting up the MATLAB model
3. Building and running the application on SCALEXIO
4. Monitoring the application with ControlDesk

Generating the Maneuver file

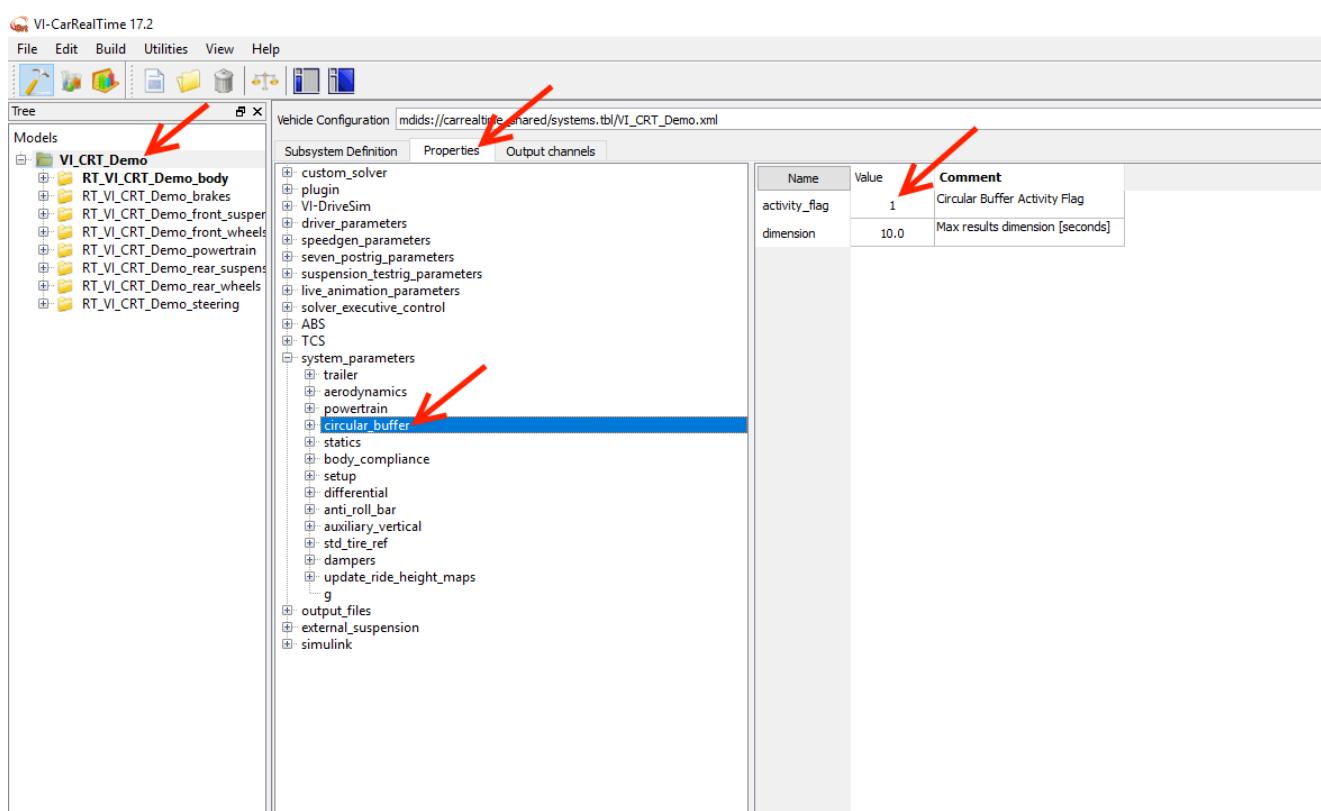
Create a new empty directory that will be used as *working directory* .

Start a new VI-CarRealTime session using the **vicrt20** command from a dos shell or by using the shortcut from Windows Start Menu and switch to your *working directory* . You can do that in VI-CarRealTime by choosing the menu Edit->Preferences and setting the **Current Working Directory** field with the path to your *working directory* .

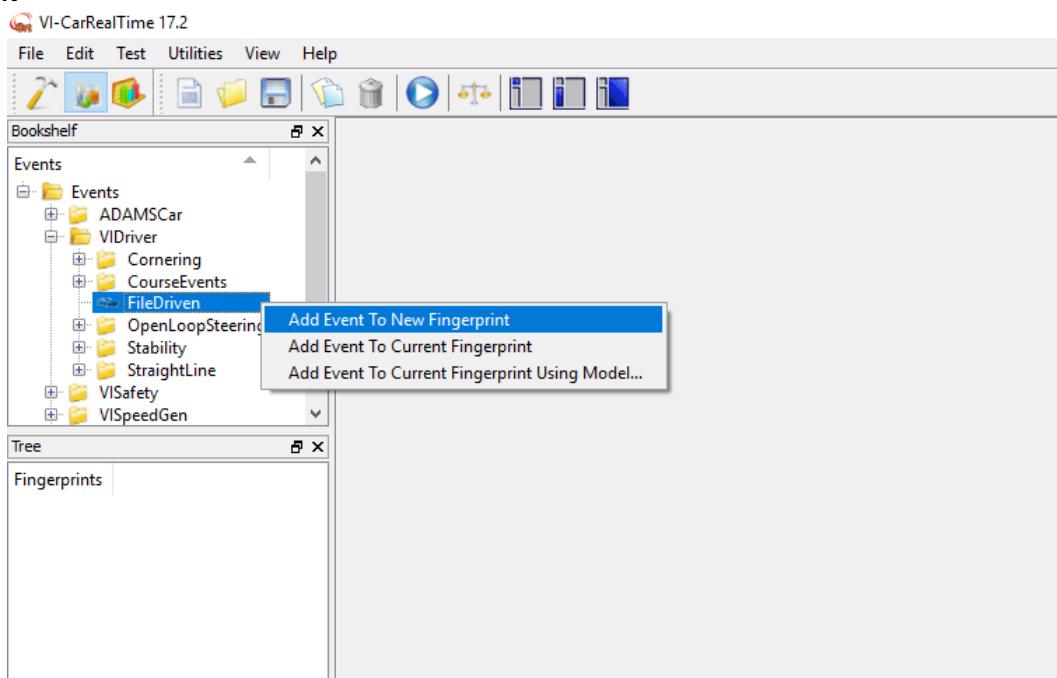
In Build mode, open the model VI_CRT_Demo.xml from the shared database.



Then, **activate the circular buffer** by clicking on the root node of the VI_CRT_Demo project, switch to the **Properties** tab, activate the `system_parameters->circular_buffer` node and, then, set the activity flag to the value 1.

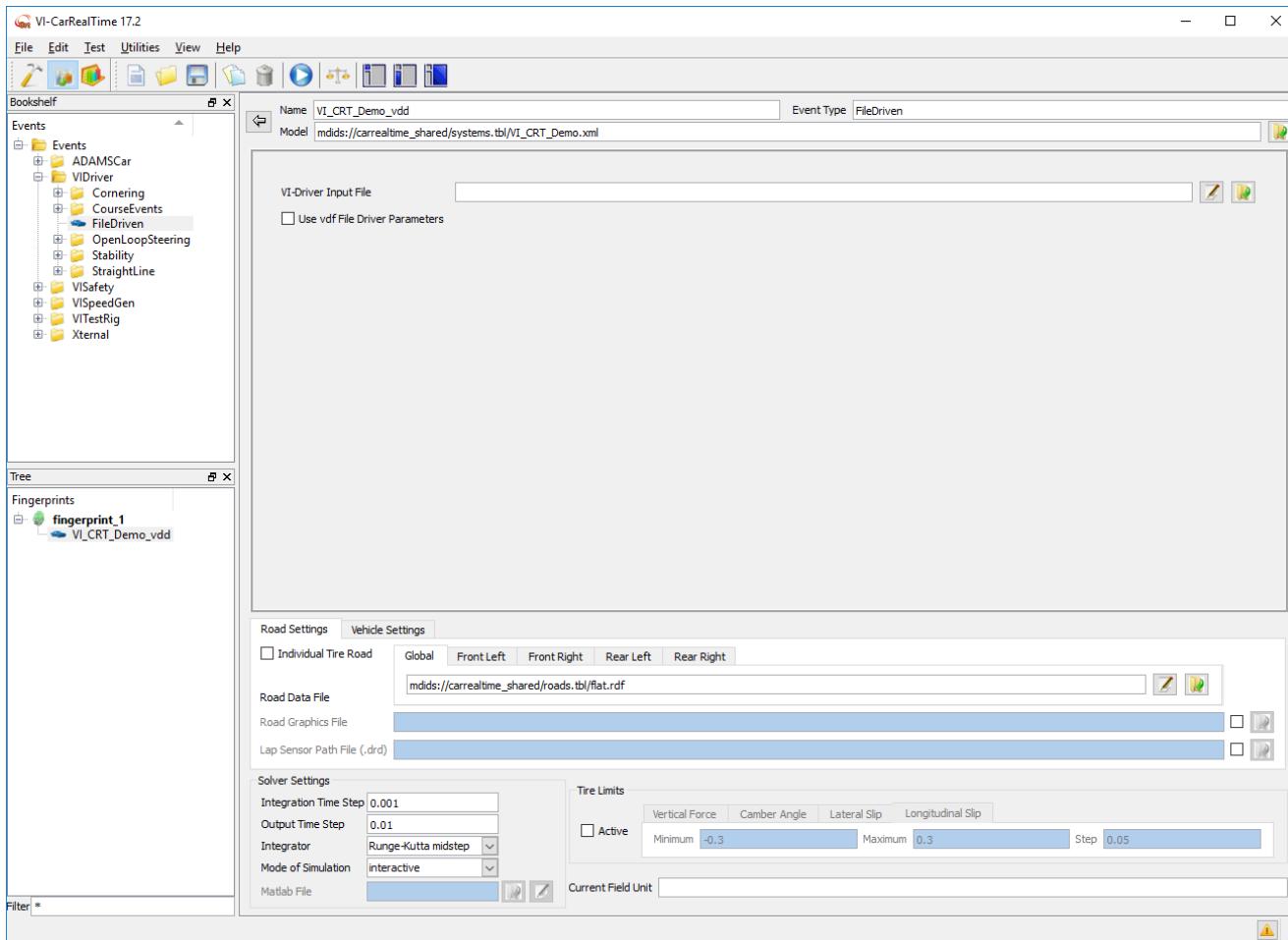


Now, you can switch to the **Test Mode** () where the event specification will be defined. Create a new fingerprint by expanding the **Events** hierarchy, right click on the **FileDriven** item and select **Add Event to New Fingerprint**.



You will end up with a new event specification as depicted in the following.

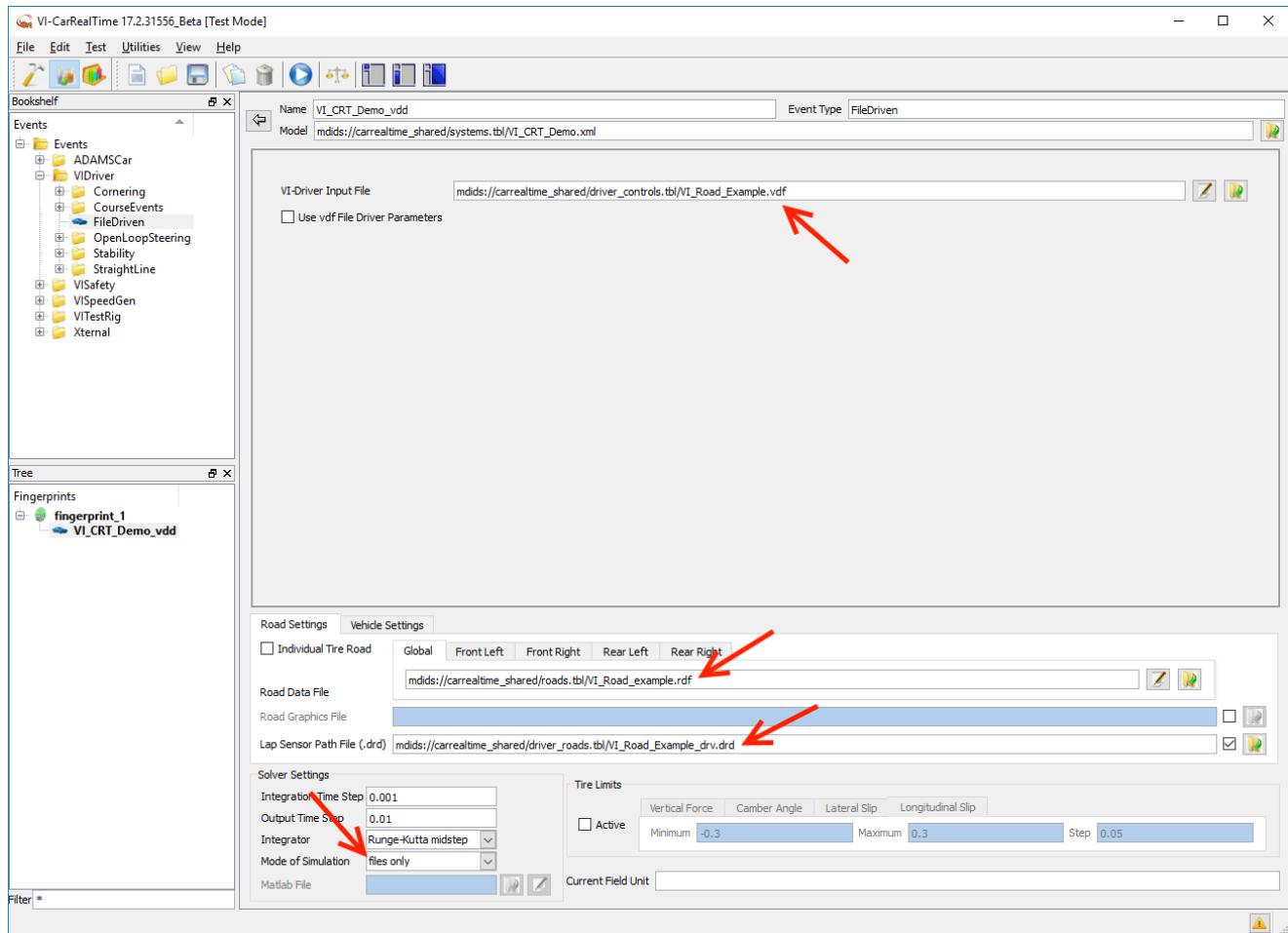
VI-CarRealTime HIL Overlays



Now, the event specification must be completed as follows:

1. Select from the shared database the **VI_Road_Example.vdf** as the **VI-Driver input file**
2. Select from the shared database the **VI_Road_Example.rdf** as the **Road Data File**
3. Select from the shared database the **VI_Road_Example_drv.drd** file as the **Lap Sensor Path File**
4. Set the simulation mode to **files only**

You will end up with the following event specification:



You can now run the maneuver () in order to generate the set of files needed by the VI-CarRealTime Matlab/Simulink interface. A console will appear with the following message:

```
File VI_CRT_Demo_vdd_send_svm.xml was successfully created..
```

Your working directory will now contain the following files:

```
VI_CRT_Demo_vdd.cmd
VI_CRT_Demo_vdd.obj
VI_CRT_Demo_vdd_graphics.xgr
VI_CRT_Demo_vdd_road_graphics.xgr
VI_CRT_Demo_vdd_send_svm.xml
VI_CRT_Demo_vdd_viani_live_startup.py
vicrt_cdb.cfg
viroad.mtl
```

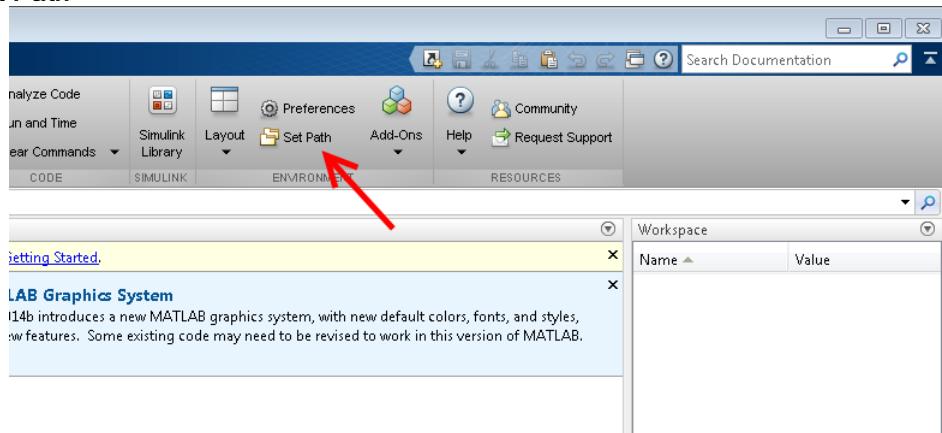
Setting up the MATLAB model

Now, copy to your *working directory* the Simulink project that is located at the following (where %CRT_DIR% is the VI-CarRealTime installation folder):

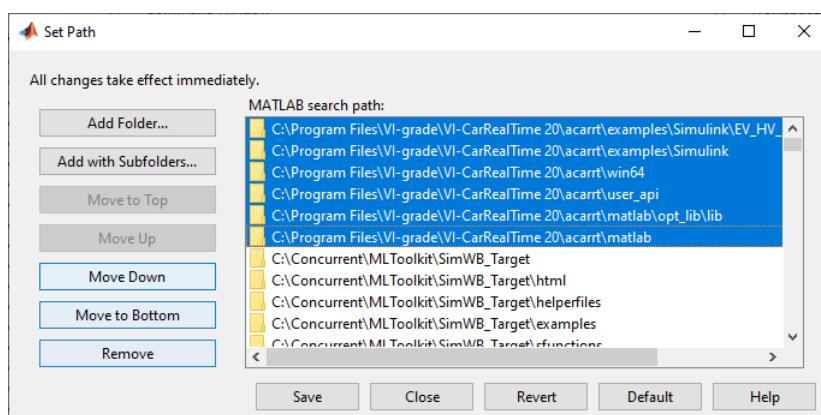
```
%CRT_DIR%\acarrt\examples\Simulink\abs_control_hil.mdl
```

Start a Matlab session and switch to that *working directory*.

If you are opening a Matlab session for the first time, add to Matlab search paths the VI-CarRealTime paths by using the "addpath_vicrt_20" script. You can check that the search paths has been configured correctly by clicking the **Set Path** button in the MATLAB toolbar:



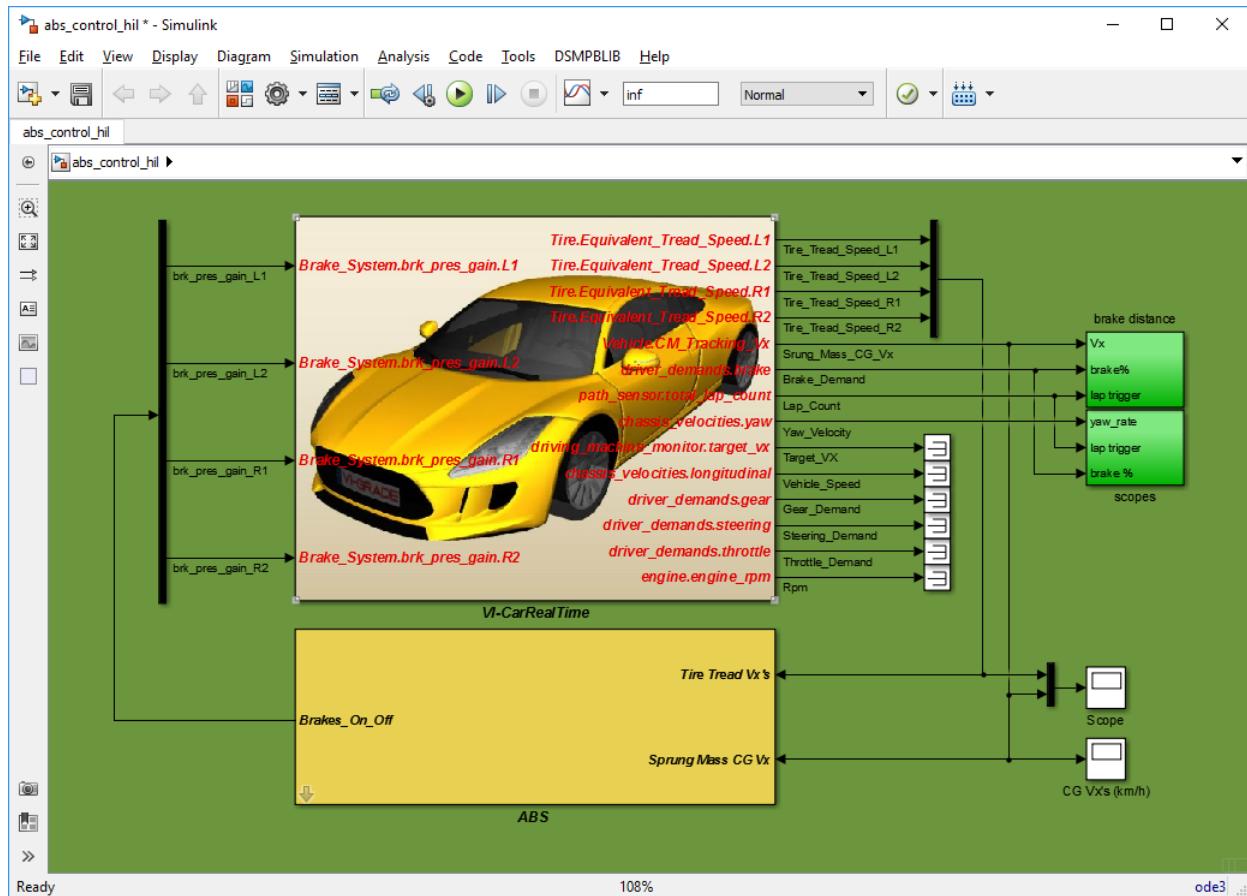
You should end up with a list of search paths similar to the one in the following pictures:



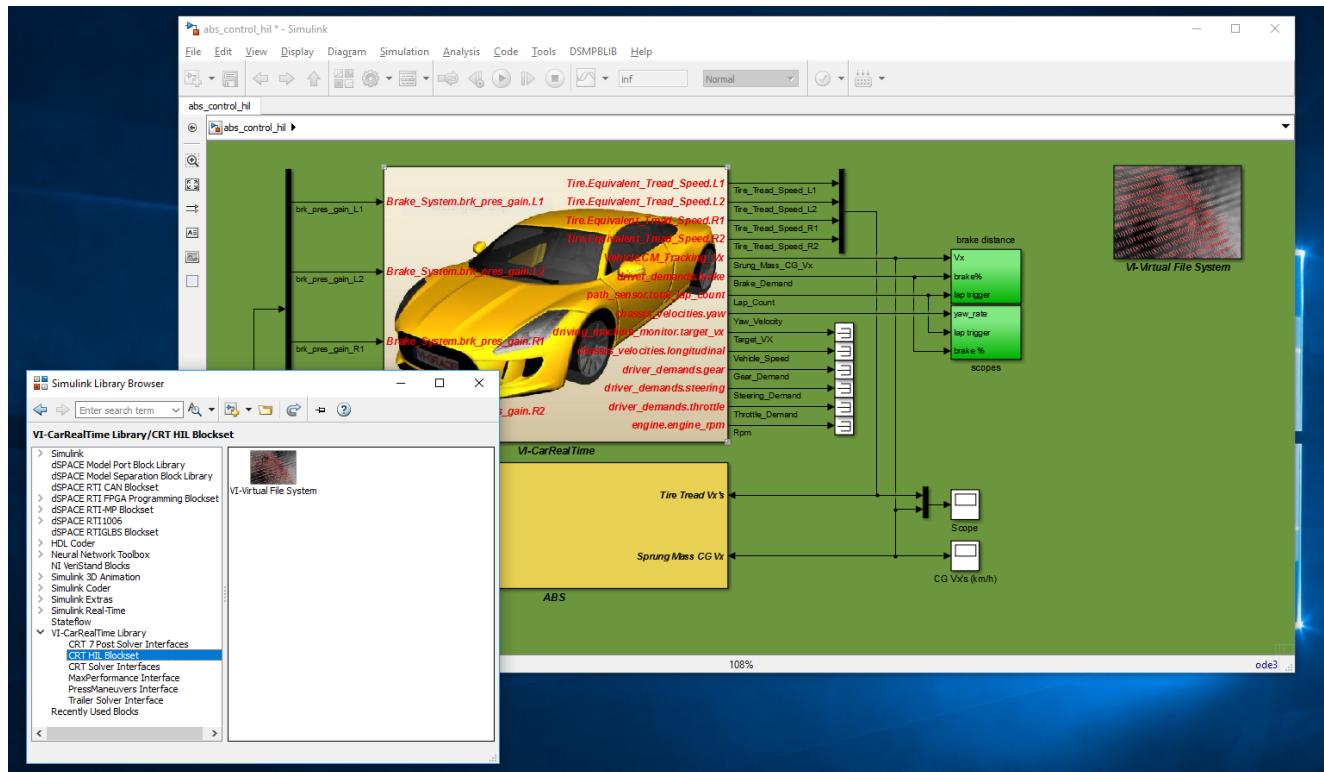
From your Matlab session, open the file `abs_control_hil.mdl` that you copied before.

This model contains several Simulink subsystems:

- VI-CarRealTime: the solver Simulink interface;
- ABS Controller: the ABS controller implementation.

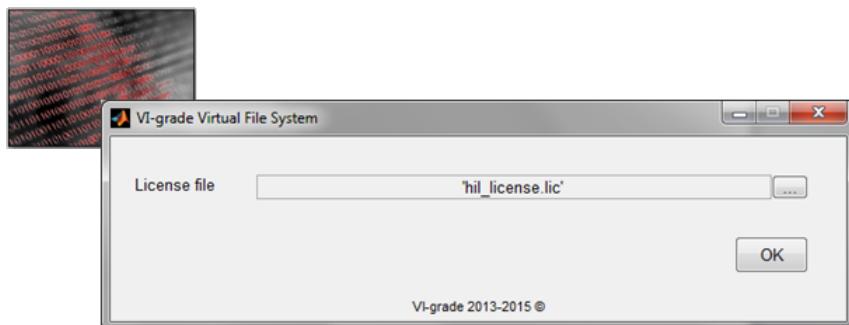


Now instrument the simulink model by using the **VI-Virtual File System** block placed in the VI-CarRealTime library.

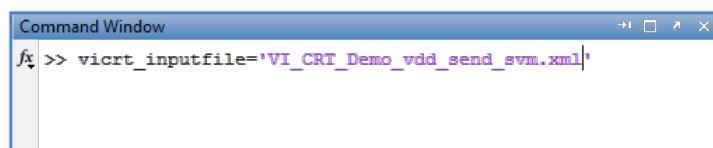


VI-CarRealTime HIL Overlays

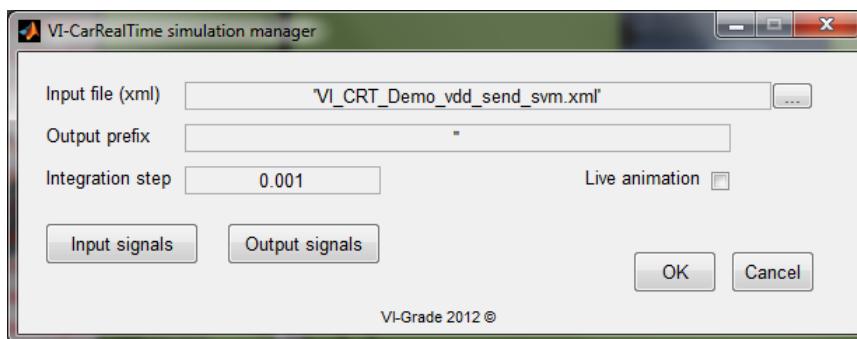
Now, set the license file using the VI-Virtual File System block mask (as described in [set the license file](#));



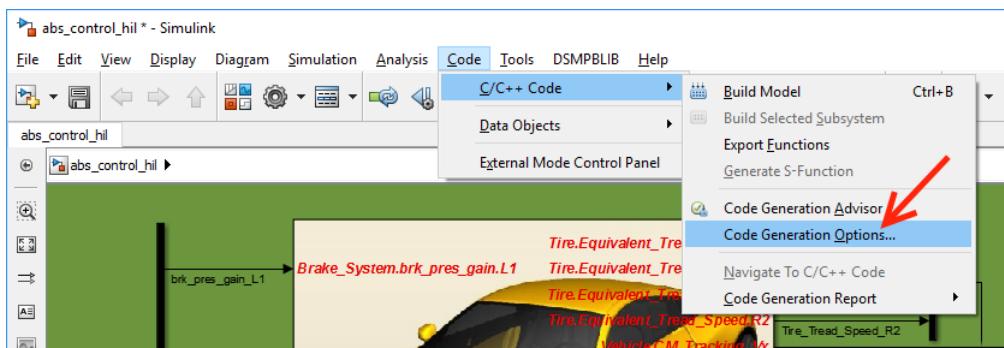
Then, set the VI-CarRealTime input file using the `vicrt_inputfile` variable.



or using the VI-CarRealTime Solver mask:



Now, please open the **Code Generation** options:



Under the category **Code Generation** and subcategory **Custom Code**, fill in the fields **Include Directories**, **Source Files** and **Libraries** as described in the following (where `%CRT_DIR%` is the VI-CarRealTime installation folder):

- **Include Directories:**

```
"%CRT_DIR%\acarrt\user_api"
```

- **Source Files:**

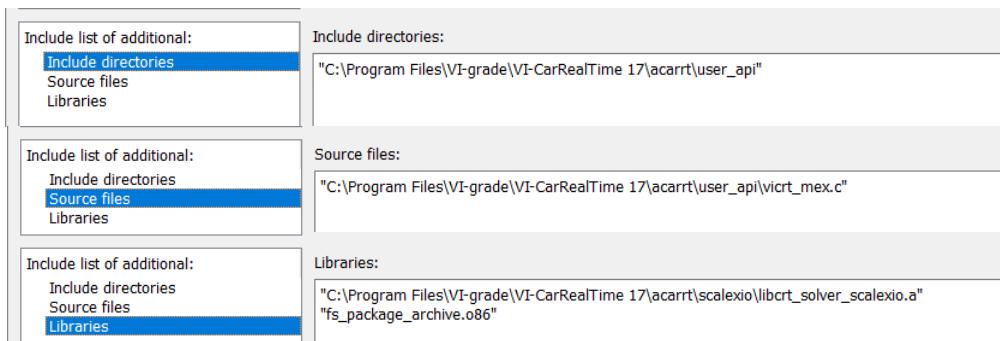
```
"%CRT_DIR%\acarrt\user_api\vicrt_mex.c"
```

- **Libraries:**

```
"%CRT_DIR%\acarrt\scalexio\libcrt_solver_scalexio.a"
```

```
"fs_package_archive.o86"
```

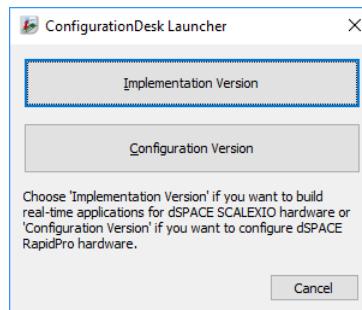
You should end up with the following settings:



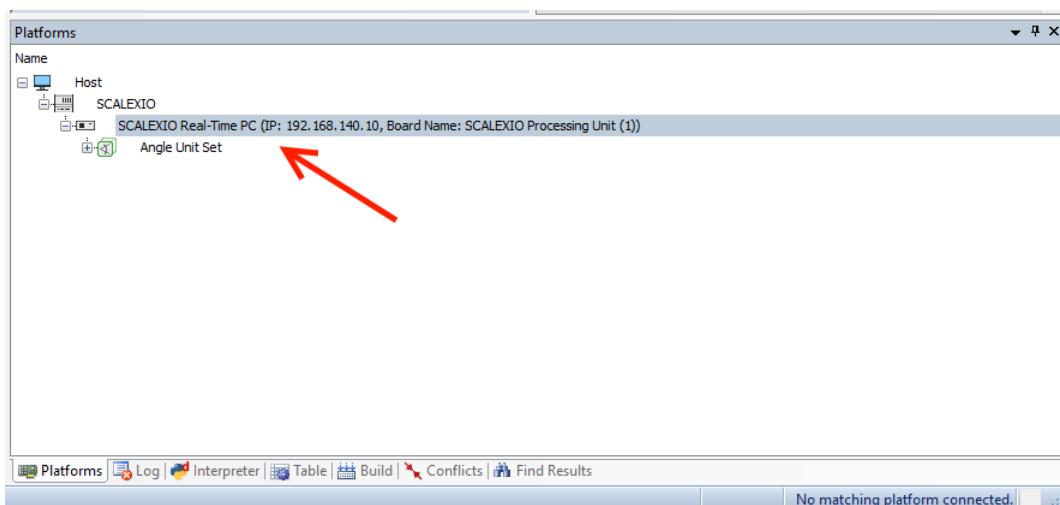
The MATLAB and Simulink configuration is now completed. You can save and close the model.

Building and running the application on SCALEXIO

In order to build the model and run it on a SCALEXIO board, you need to launch the dSpace ConfigurationDesk and choose the option **Implementation Version**.

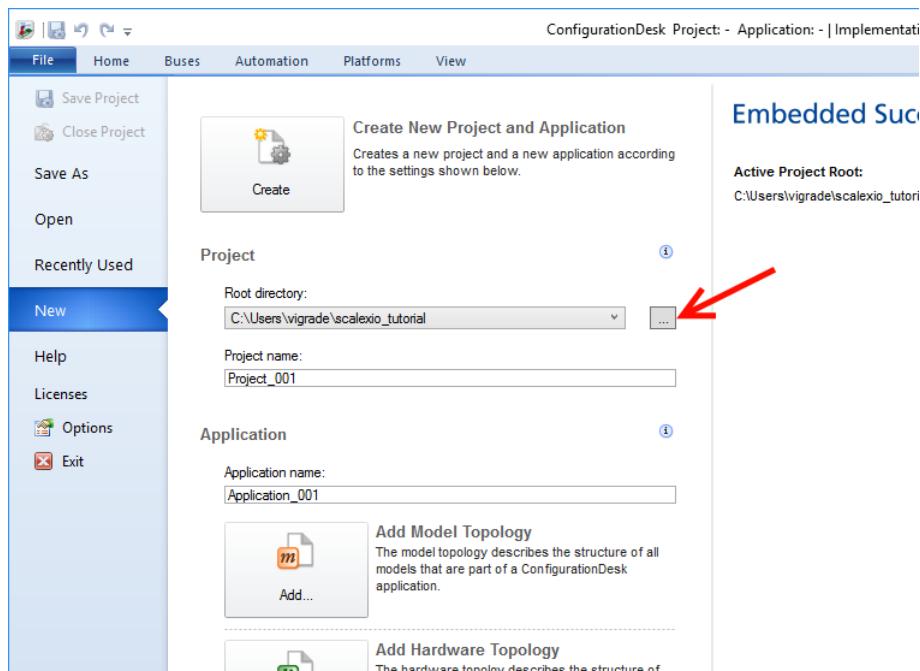


The ConfigurationDesk application will start and you should find your SCALEXIO device available in the **Platforms** lower pane:

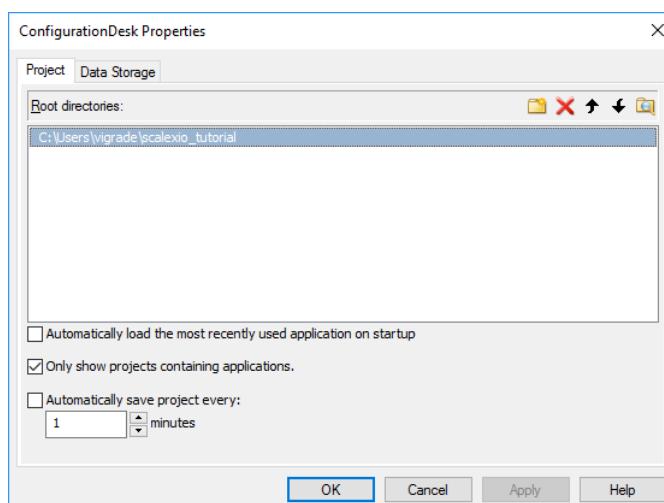


If this is not the case, please read the dSpace reference documentation for details on how to register a SCALEXIO board with the dSpace software tools.

In order to create a **New Project**, you need to select a new root directory; the root directory will be used as a workspace where subdirectories will be created for each HIL application that will be built. For the sake of simplicity, let's use the same *working directory* that was used in the previous steps of this tutorial. This can be done by clicking on the menu **File->New** and clicking on the button highlighted in the following:



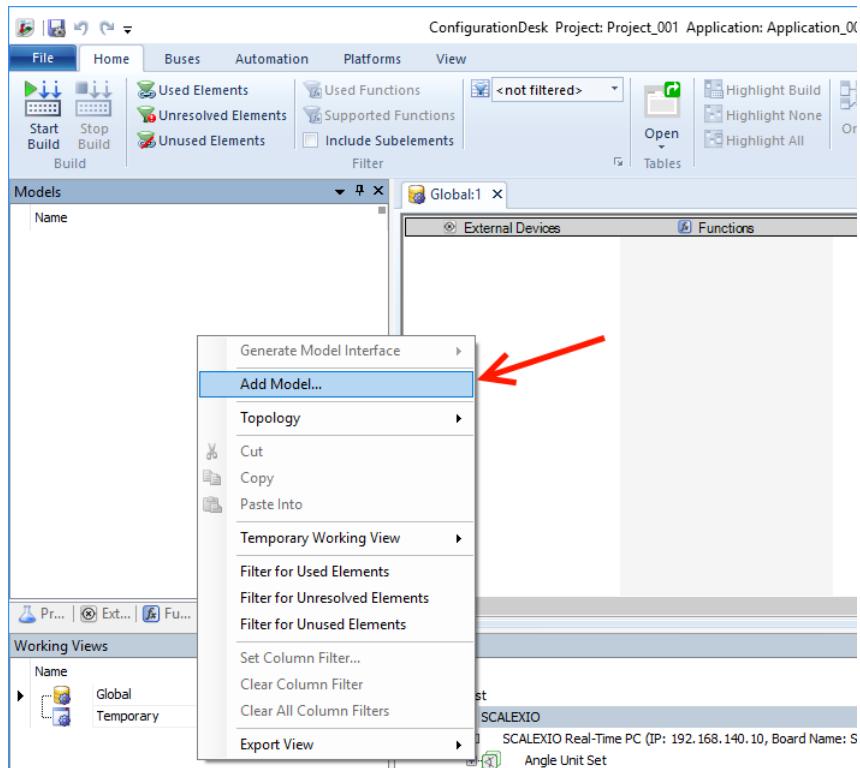
Please, insert your *working directory* and click OK.



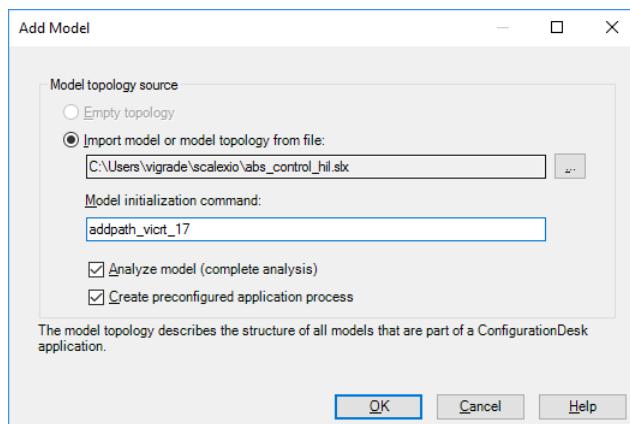
Now that the root has been selected, please push the button **Create** to create a new project and accept the defaults. An empty project will be created and a corresponding new directory will be created in the *working directory*.

You can now select the **MATLAB model** that will be exercised on the SCALEXIO board by clicking on the **Models** tab in the left pane, right-click in the middle of the pane and select **Add model...**

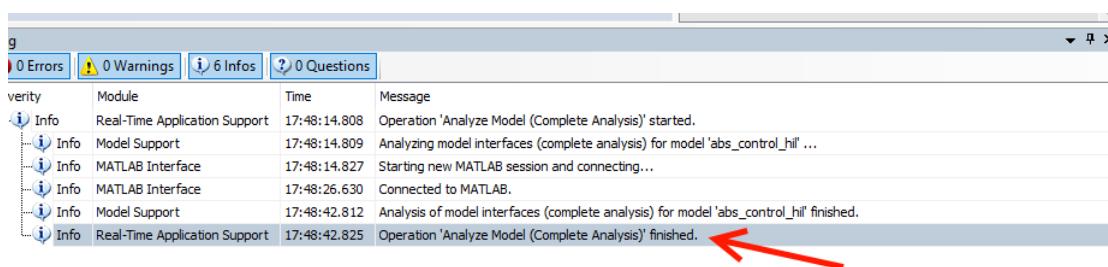
VI-CarRealTime HIL Overlays



A dialog will appear in which you are asked to insert the path of the Simulink model and an initialization script. Please, insert the complete path to the model `abs_control_hil.mdl`, set "addpath_vicrt_20" as the initialization script and then click OK.

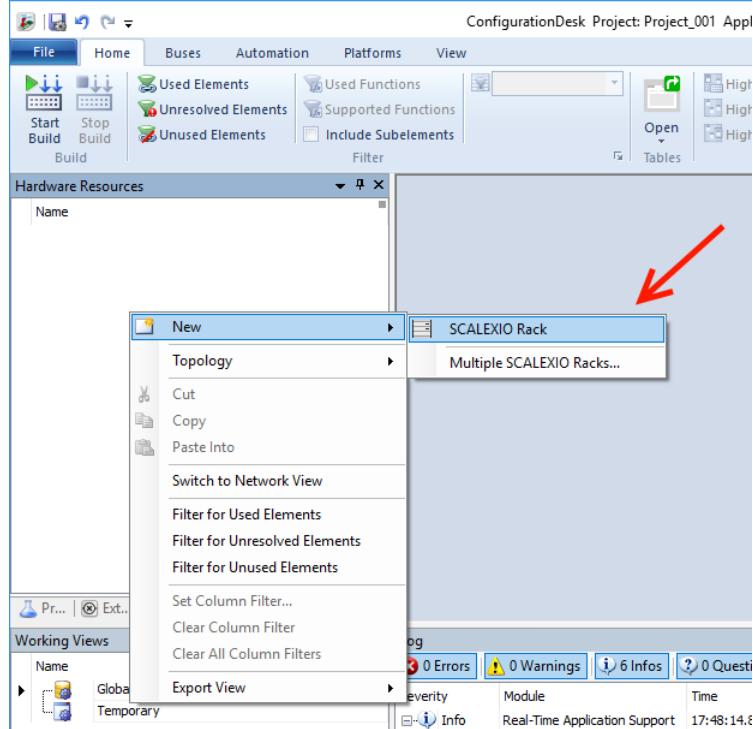


MATLAB will be invoked by ConfigurationDesk in order to analyze the model; once analysis has finished, please check that the lower **Log** pane contains no errors as in the following picture.

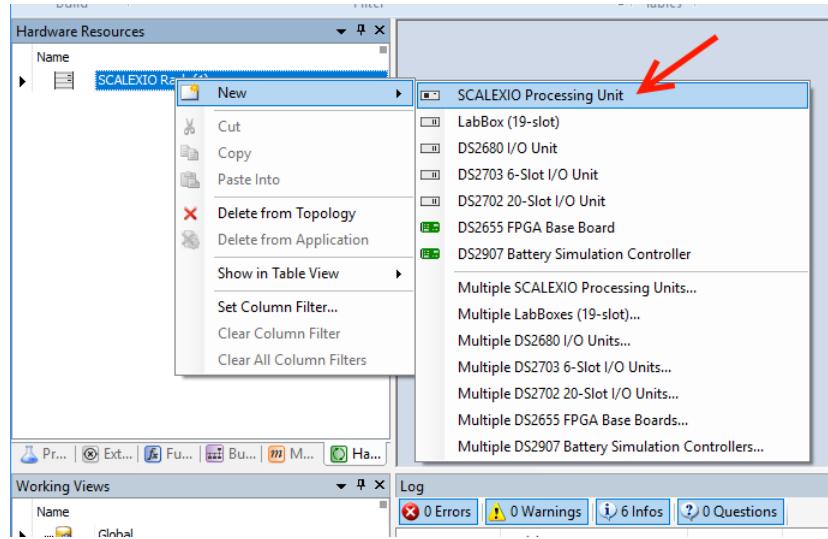


Note: when the analysis has finished, no matter if successfully or not, the MATLAB window will remain in the foreground. We suggest you to give focus to ConfigurationDesk while waiting for the analysis to complete and, then, only in case of issues, to switch back to MATLAB for inspecting the details of the issue.

You can now select the **Hardware** tab, right click in the middle of the pane and select **New->SCALEXIO Rack**.

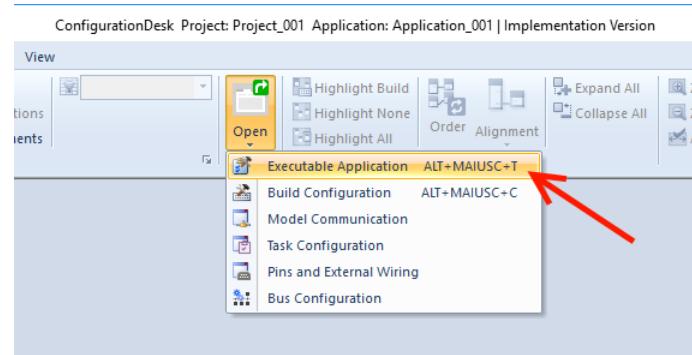


A new item will be created; right-click on it and choose **New->SCALEXIO Processing Unit**:

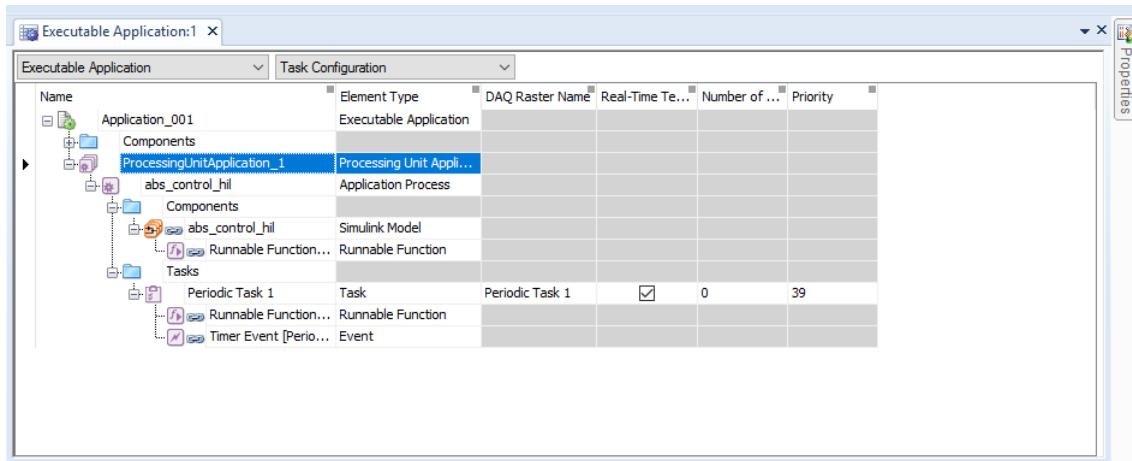


Before starting the build, please check that your configuration is correct by clicking on the **Open Tables** button in the toolbar and choose **Executable Application**.

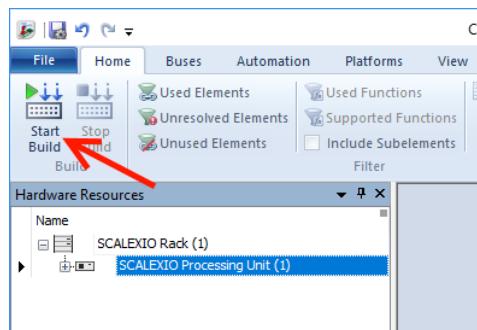
VI-CarRealTime HIL Overlays



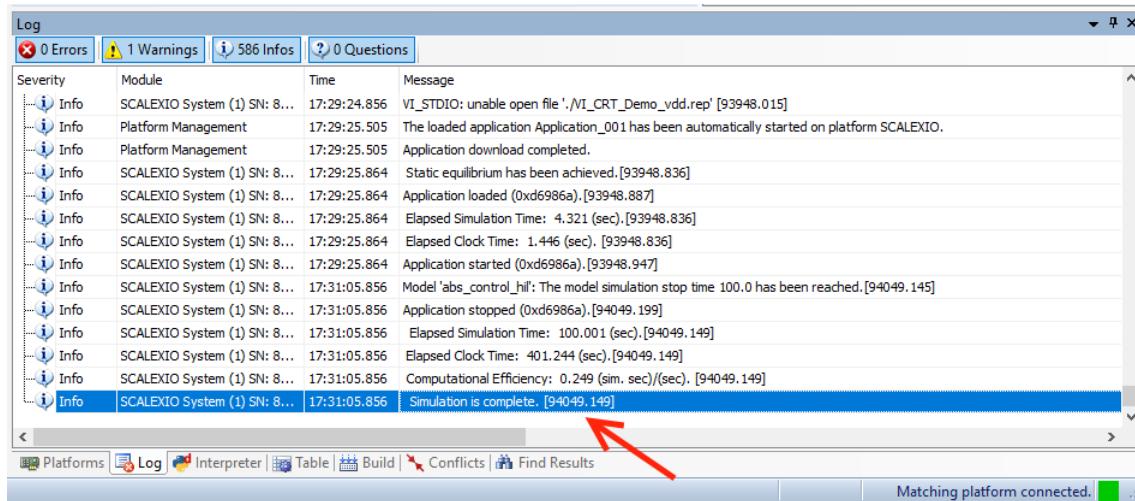
Check that your configuration is like the following:



If your configuration is correct, you can now click on the **Start Build** button on the toolbar.



By using the default settings, the source code will be generated and cross-compiled, the binary will be downloaded to the SCALEXIO board and the application will be run on the board. You will see that by checking the **Log** tab in the lower pane.



Please, examine the items in the log. If the simulation is running correctly, you should find the following sequence of messages coming from the core of VI-CarRealTime (interspersed with messages coming from the dSpace software):

```
=====
=           VI-CarRealTime          =
=====

VERSION : 17.1
REVISION: 0/None
BUILT ON: taurus/10-Oct-16
=====

Using Simulation File: C:
\Users\vigrade\scalexio_tutorial\VI_CRT_Demo_vdd_send_svm.xml
Event data is being read from input XML file...
VI_STDIO: unable open file '.'

-- WARNING -- ./ is not a valid directory
VI_STDIO: unable open file './vicrt_cdb.cfg'
Performing the simulation: VI_CRT_Demo_vdd.
Initializing powertrain / driveline...
Done.
Initializing aero force...
Done.
Initializing tires...
Loading road data file... (file=C:/PROGRA~1/VI-grade/VI-
CAR~1/acarrt/carrealtime_shared.cdb/roads.tbl/VI_Road_example.rdf)
road file total reading time = 0.008s[173377.761] (0xCA,1)
ANALYTIC road model has been initialized...
initializing VI-TIRE Pacejka 96/02 model... OK
Done.
Initializing driver...
Done.
-- INFO -- Enabling Circular Buffer: Only the last 5.00 seconds of analysis
will be written.
Solving for static equilibrium...
Static equilibrium has been achieved.
```

VI-CarRealTime HIL Overlays

```
Model 'abs_control_hil': The model simulation stop time 100.0 has been
reached.
Application stopped (0xd6986d).
Elapsed Simulation Time: 100.001 (sec).
Elapsed Clock Time: 401.172 (sec).
Computational Efficiency: 0.249 (sim. sec) / (sec).
Simulation is complete.
```

If you can find the log item Static equilibrium has been achieved, it means that the simulation is running correctly.

Please, check that, under your *working directory* , the subfolder Project_001\Application_001\Build Results contains the file Application_001.sdf. This will be necessary for completing the next step of the tutorial.

Monitoring the application with ControlDeskNG

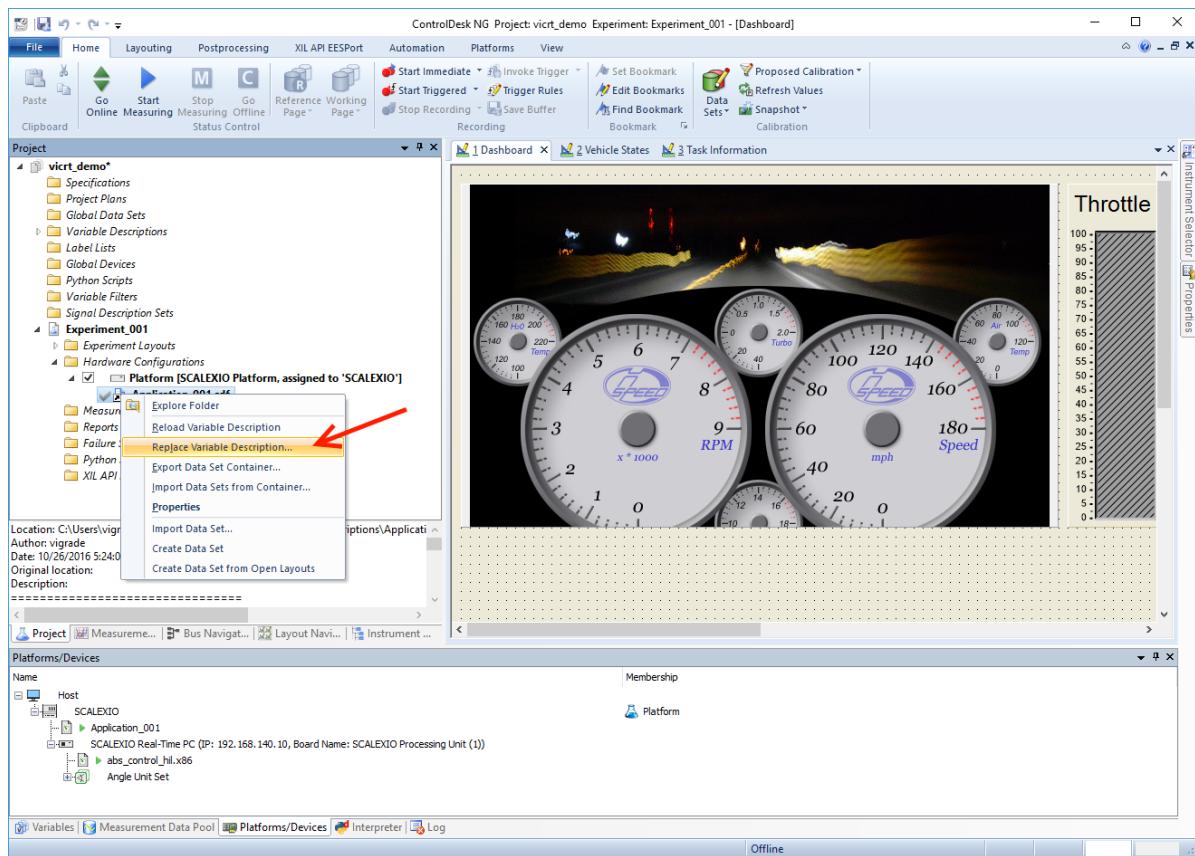
In order to monitor the application and check that it is behaving as expected, we will use a demo ControlDeskNG project that is distributed with the SCALEXIO overlay.

Please, copy the following folder to your *working directory* (%CRT_DIR% is the VI-CarRealTime installation folder):

```
%CRT_DIR%\acarrt\examples\scalexio\dsProjects
```

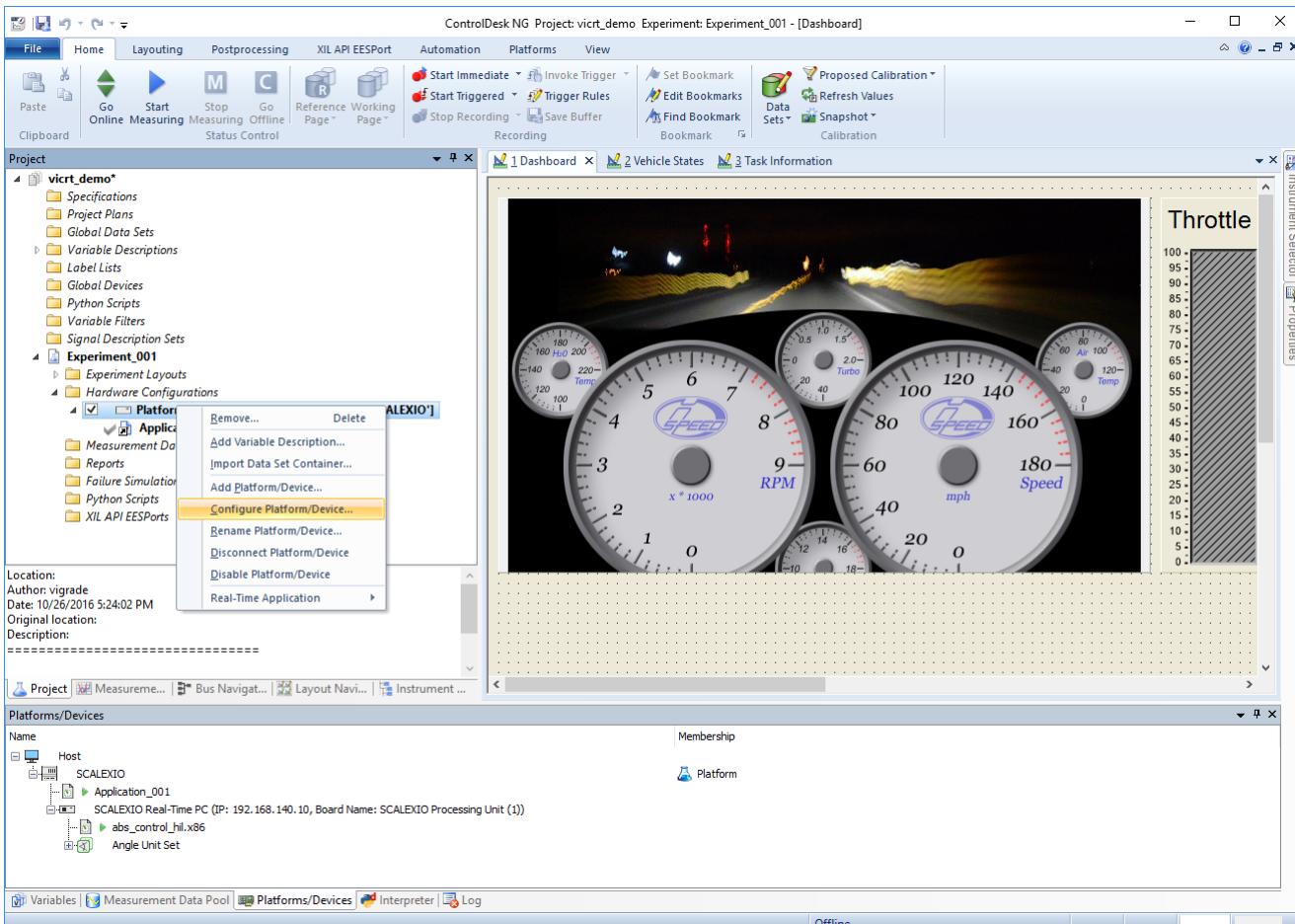
Your working directory will now contain a folder called `dsProjects`. Launch ControlDeskNG, select as root directory the folder `dsProjects` under your *working directory* and load the **vicrt_demo** project.

Under the **project pane**, right click on the item `abs_control_hil.sdf` under Hardware Configurations, select **Replace variable description** and select the file `Application_001.sdf` that you checked at the end of the previous step of this tutorial.



If the platform isn't detected, check the platform configuration using the **Configure Platform\Device** from the Platform contextual menu (right click on the Platform), and select the correct connection for the platform (BUS or NET).

VI-CarRealTime HIL Overlays



Finally, start the run time analysis by pushing the **Start Measuring** button.

3.2 VI-CarRealTime NI-PXI Overlay

3.2.1 Introduction

The VI-CarRealTime NI-PXI Overlay is an additional tool that allows the execution of the VI-CarRealTime model on the National Instruments NI-PXI controller.

After the installation, it is possible to build an NI-PXI application as follows:

- either using the VI-CarRealTime Matlab/Simulink interface and the NI-VeriStand Real Time Workshop extension
- or, using the VI-CarRealTime NI-VeriStand interface

The VI-CarRealTime NI-PXI Overlay is provided under a specific board locked license.

Notes:

- The **Microsoft Visual C++ 2010** compiler toolchain is required to build VI-grade software for the NI-PXI controller.
- In order for VI-CarRealTime NI-PXI Overlay to work properly, a **multi-core NI-PXI target machine is required**. A single core machine will crash; a dual-core machine is the minimum but VI-grade highly recommends a 4 core machine.

[Installation](#)

[Licensing](#)

[Limitations](#)

[Tutorial](#)

3.2.2 Installation

Welcome to the installation guide for VI-CarRealTime NI-PXI Overlay. VI-CarRealTime NI-PXI Overlay is currently for windows only.

In order to perform a complete installation of VI-CarRealTime NI-PXI Overlay, you need to pickup the following files from the VI-grade [website Software Download section](#) area (registration is required):

- **VI_Crt_ni_pxi_20_0_x_Setup.exe**: main product installer;
- **VI_Crt_ni_pxi_20_0_Installing.pdf**: this document;
- **VI_Crt_ni_pxi_20_0_Release_Notes.pdf**: product information.
- **VI_Crt_overlay_20_0_x86_Setup.exe**: 32-bit overlay of VI-CarRealTime

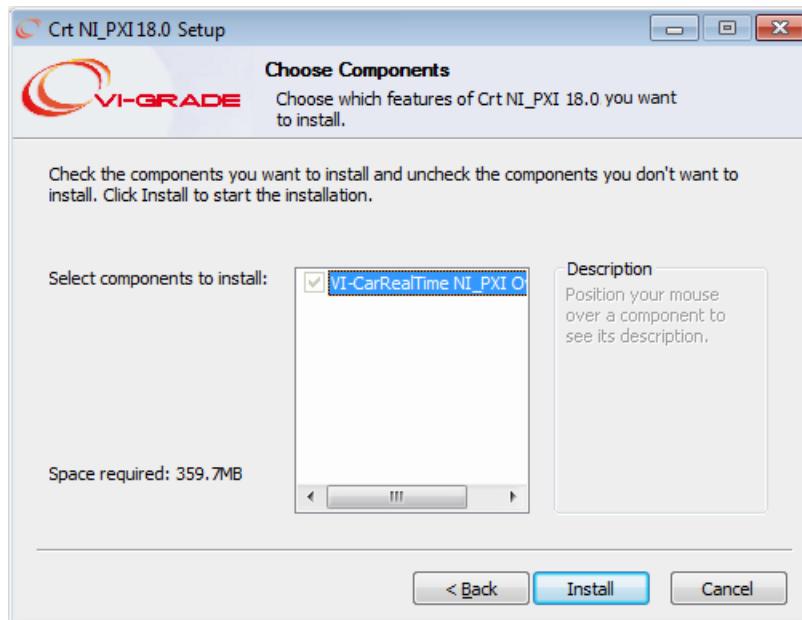
After reviewing the Release Notes document, the installation can start following the steps below (please, follow the same approach also to install the 32-bit overlay of VI-CarRealTime):

1. Double click on the **VI_Crt_ni_pxi_20_0_x_Setup.exe** file, and follow the instructions that will appear on the screen:



The following component will be installed:

- **VI-CarRealTime NI_PXI Overlay**
a specific VI-CarRealTime NI-PXI compliant library and other support files. Older VI-CarRealTime NI-PXI Overlay 20 installation, already available on the target machine will be updated.



At the end of the installation the following window will appear:



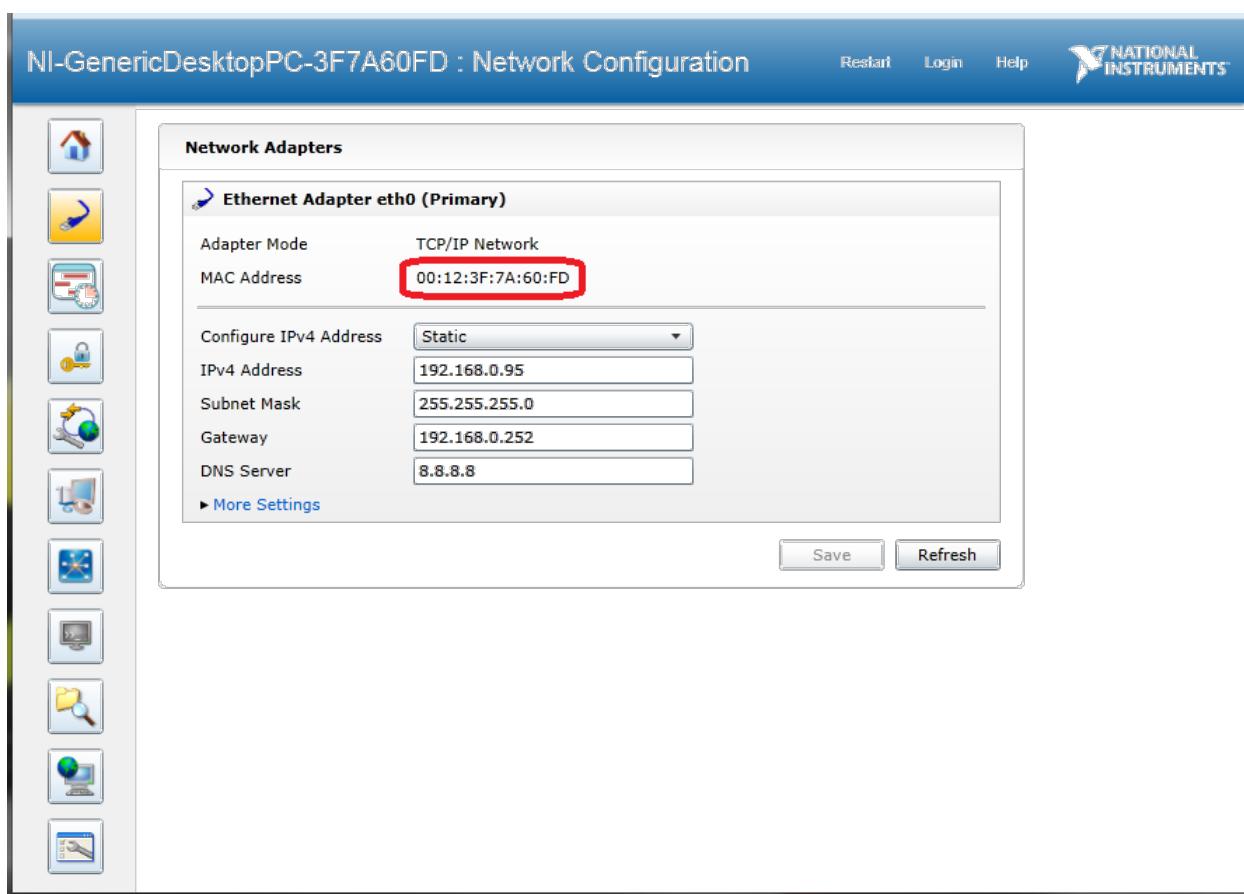
Click "Finish" to complete the installation.

This procedure install the NI_PXI directory inside the VI-CarRealTime installation directory (or inside the one selected during the installation phase).

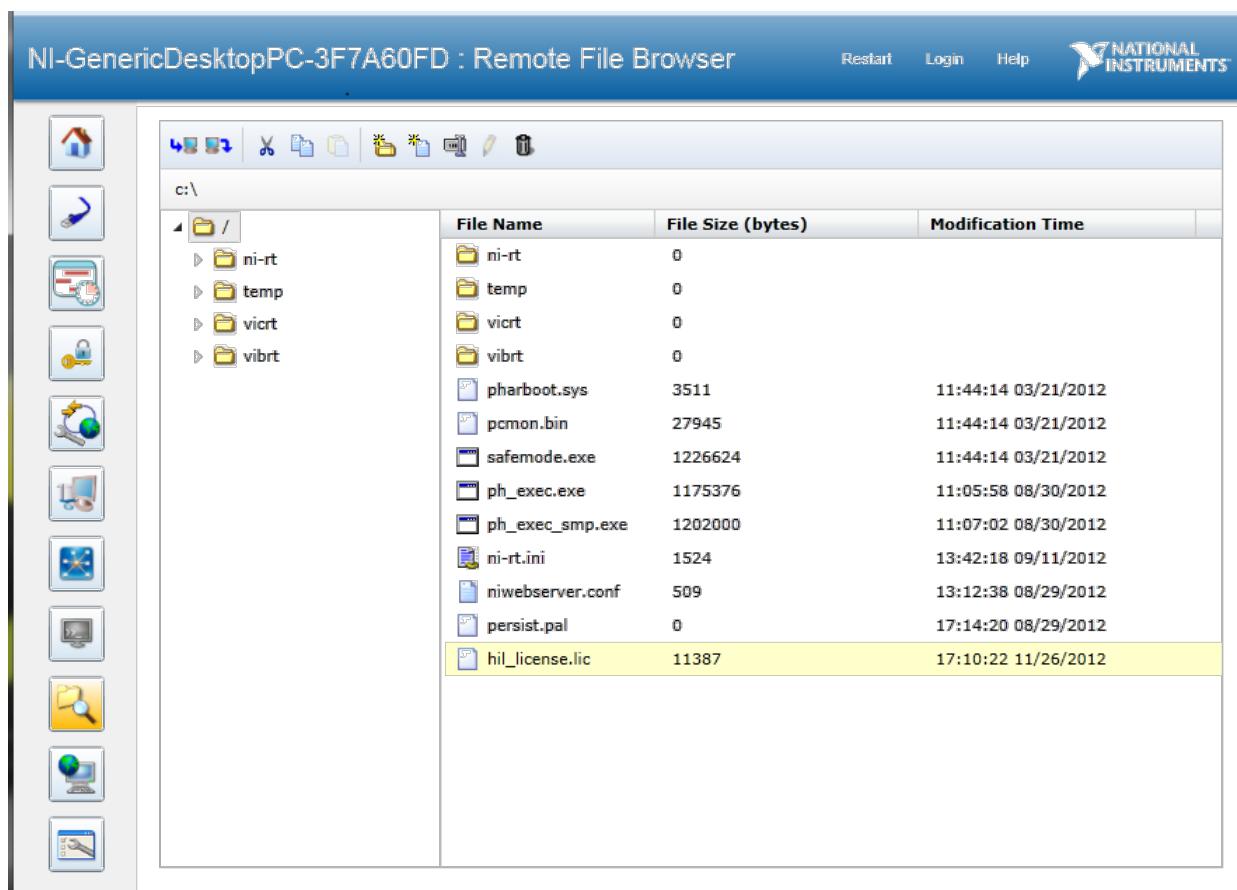
3.2.3 Licensing

In order to execute VI-CarRealTime on the National Instruments PXI board, a board specific license file is needed.

The license file can be retrieved sending to our [licensing support](#) the NI-PXI controller MAC address, displayed by the Network Configuration pane.



After receiving the license file, it must be placed in the **root** directory of the NI-PXI platform with the name **hil_license.lic**.



3.2.4 Limitations

The current version of the VI-CarRealTime porting for NI_PXI does not support the full set of functionalities featured by the Windows version of the software. Please refer to the following table for the unsupported capabilities.

Modelling	External FMUs FTire
Events	MF-Tyre / MF-Swift Max Performance Press Maneuvers SpeedGen Sevenpostrig

Model extensions such as custom tires and aero require specific builds of the user code for the NI_PXI platform.

3.2.5 Tutorials

This section contains tutorials for experimenting the 2 approaches of running VI-CarRealTime on NI-PXI targets:

- with the traditional approach of **using Simulink** to create a model in which VI-CarRealTime interacts with other Simulink blocks ([ABS Control](#)), then cross-compile the model for the NI-PXI target and finally use NI-Veristand to run the experiment
- with the more direct approach of **using NI-Veristand** to compose models built for the NI-PXI target together with the VI-CarRealTime NI-Veristand interface ([FMI with the NI-Veristand Interface](#))

In these tutorials, we will refer to the directory where VI-CarRealTime is installed with the symbol %CRT_DIR%.

ABS Control

This tutorial shows how to run a standard VI-CarRealTime Simulink model on the NI-PXI controller. The Simulink model is a co-simulation between VI-CarRealTime and an external ABS control system. The tutorial is divided into these steps:

[Setting up the working directory](#)

[Generating the maneuver file](#)

[Uploading the maneuver to the target](#)

[Setting up the Simulink Model](#)

[Building the application](#)

[Monitoring the application with VeriStand](#)

Setting up the working directory

In order to create the working directory, please double click on the following batch file:

```
%CRT_DIR%\acarrt\examples\NI_PXI\nipxi_prepare_tutorial.bat
```

The script will execute the following sequence of steps (that can be interrupted by pressing the key combination CTRL+C):

1. Collect information: local working directory for the tutorial, IP address of the target NI-PXI machine, FTP credentials, VI-grade license file
2. Check that the target NI-PXI machine can be reached
3. Make an FTP connection
4. Prepare the local working directory
5. Copy VI-CarRealtime databases to the local working directory
6. Copy NI-Veristand demo project to the local working directory
7. Copy MATLAB demo project to the local working directory
8. Copy support python scripts...
9. Create script to launch NI-Veristand with the modified PATH env. variable so that the Simulink model DLL can be correctly added to the System Definition File
10. Create script for transferring VI-CarRealtime DLLs, VI-CarRealtime shared databases and license files to the target NI-PXI via FTP
11. Summary

There are **3 steps that deserve attention**: Step 1., Step 2. and Step 3..

During **Step 1.**, the user will be asked to type in the following information:

- **local working directory**: this directory will be used for copying all the files required for completing the tutorial - the working directory must be non-existent (that's because it will be created automatically) and must be in the root of the drive (e.g., I:\vicrt)
- **IP address of the target**: this is the IP address of the NI-PXI machine where the model will be running
- **FTP credentials of the target**: these are the FTP username and password of the NI-PXI machine
- **license file for the target**: this is the absolute path to the license file that VI-grade has generated for the specific target NI-PXI machine

Please, collect all these pieces of information *before* running nipxi_prepare_tutorial.bat. Here is an example of running the script and typing in the required information:

...

VI-CarRealTime HIL Overlays

1. Collect information: local working directory for the tutorial, IP address of the target NI-PXI machine, FTP credentials, VI-grade license file

Please type in a valid directory path (the directory must be NOT existent and must be located in the root of an hard drive): I:\vicrt

Please type in a valid IP address for the target NI-PXI machine: 192.168.0.190

Please type in the FTP username for accessing the target NI-PXI machine: admin

Please type in the FTP password:

Please type in the absolute path of a license file for the target NI-PXI machine: I:\\VIGRADE_9340.lic

...

During **Step 2.**, the target machine will be *ping-ed* in order to check that the specified IP address is reachable. During **Step 3.**, an FTP connection will be attempted with the specified user name and password. During these steps, if anything fails, you will be asked to check that the machine is on and that it is connected to the network; then, you will be able to make additional attempts.

Let's suppose that you have chosen the path I:\\vicrt as your working directory. You will end up with the following directories/files located in I:\\vicrt:

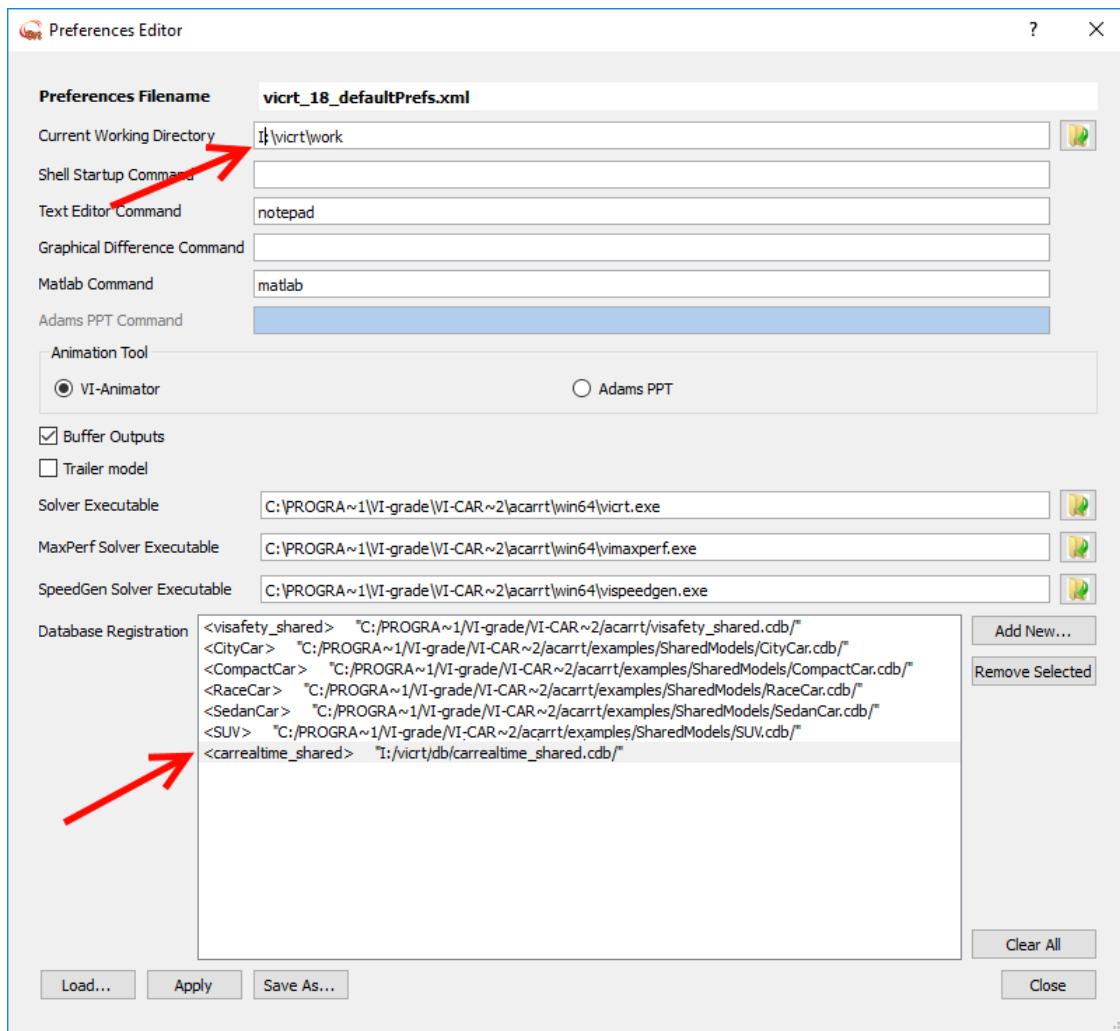
- crt: this is the directory where the maneuver file will be generated (see [Generating the maneuver file](#))
- assets: this directory contains the VI-CarRealTime database and a copy of the specified VI-grade license file for the NI-PXI target
- demo_matlab: this directory contains the demo MATLAB/Simulink model (see [Setting up the Simulink Model](#))
- demo_veristand: this directory contains the demo Veristand project (see [Monitoring the application with VeriStand](#))
- scripts: this directory contains support scripts
- nivs: this directory contains facilities for launching Veristand together with the VI-CarRealTime Veristand interface (see [FMI with the NI-Veristand Interface](#))
- launch_veristand.bat: this is the script to launch Veristand (see [Monitoring the application with VeriStand](#))
- transfer_to_target.bat: this is the script to transfer to the NI-PXI target machine all the files needed by VI-CarRealTime in order to execute the simulation (see [Uploading the maneuver to the target](#))
- transfer_binaries_only.bat: this is the script to transfer to the NI-PXI target machine only the binary files (this proves to be useful when the target NI-PXI machine has been previously setup and used and now needs to be updated with a new release of the VI-CarRealTime overlay for NI-PXI)

Generating the maneuver file

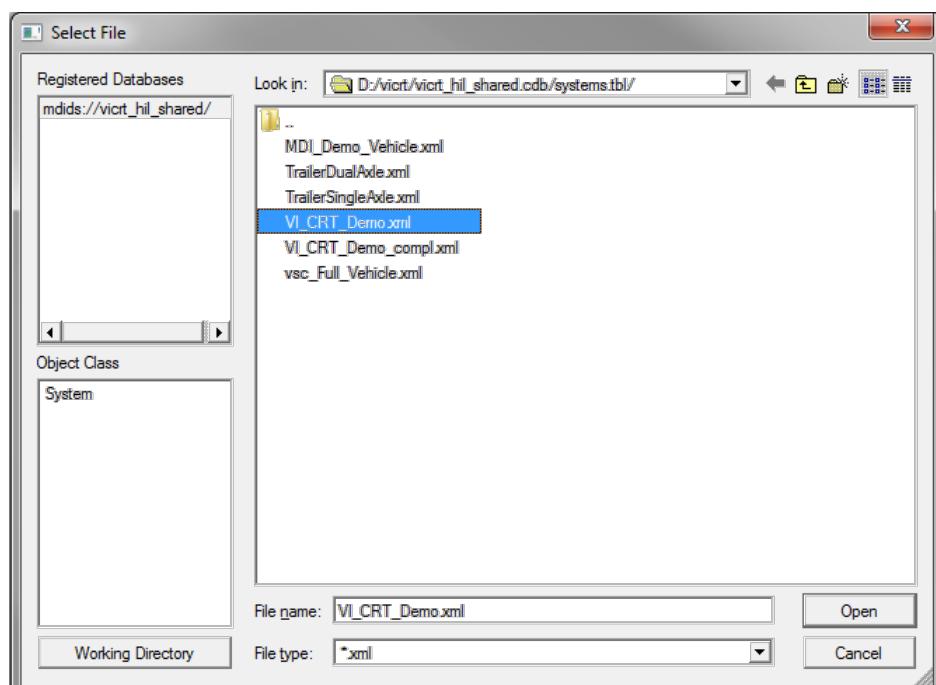
Now, start a new VI-CarRealTime session using the **vicrt20** command from a dos shell or by using the shortcut from Windows Start Menu.

- Set up the VI-CarRealTime working directory to I:\\vicrt\\crt.
- Set up the VI-CarRealTime shared database in order to use the one placed under I:\\vicrt\\db\\carrealtime_shared.cdb.

Both of these operations can be done in VI-CarRealTime by choosing the menu Edit->Preferences and set the fields as highlighted in the following picture:

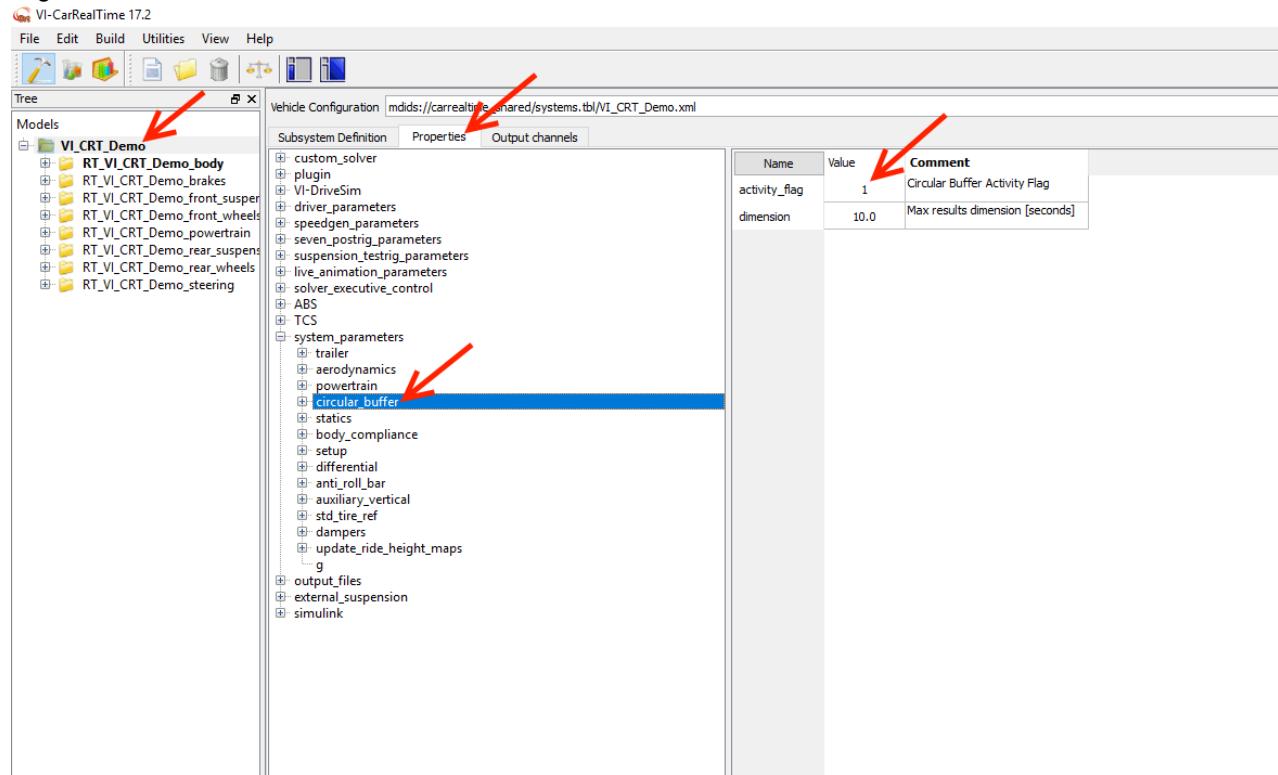


In Build mode, open the model `VI_CRT_Demo.xml` from the `carrealtime_share` database.

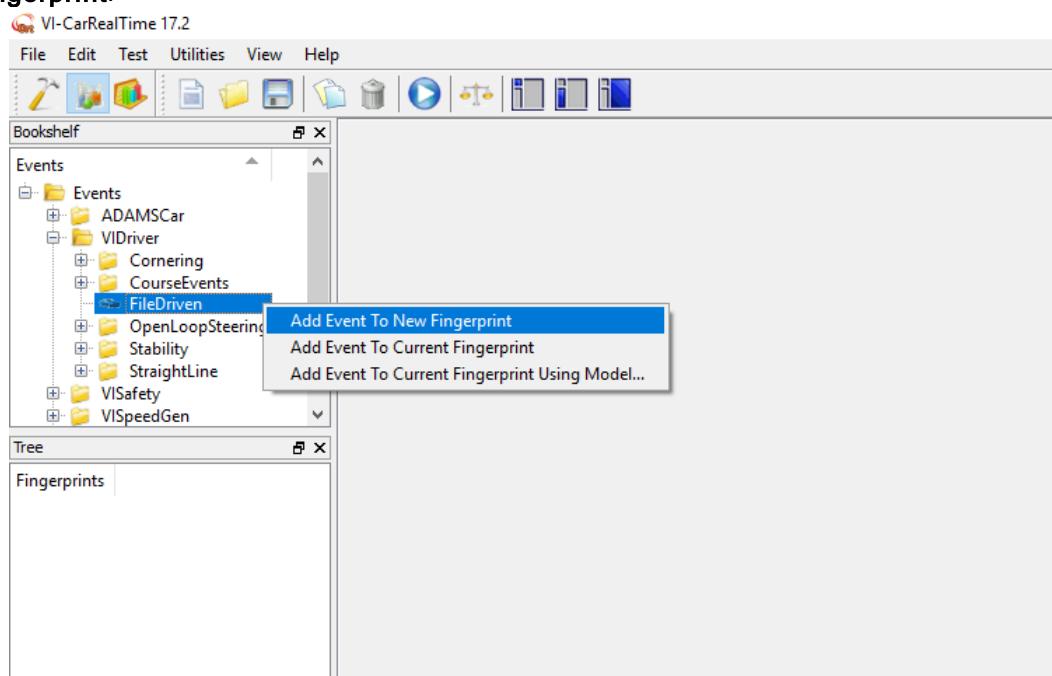


VI-CarRealTime HIL Overlays

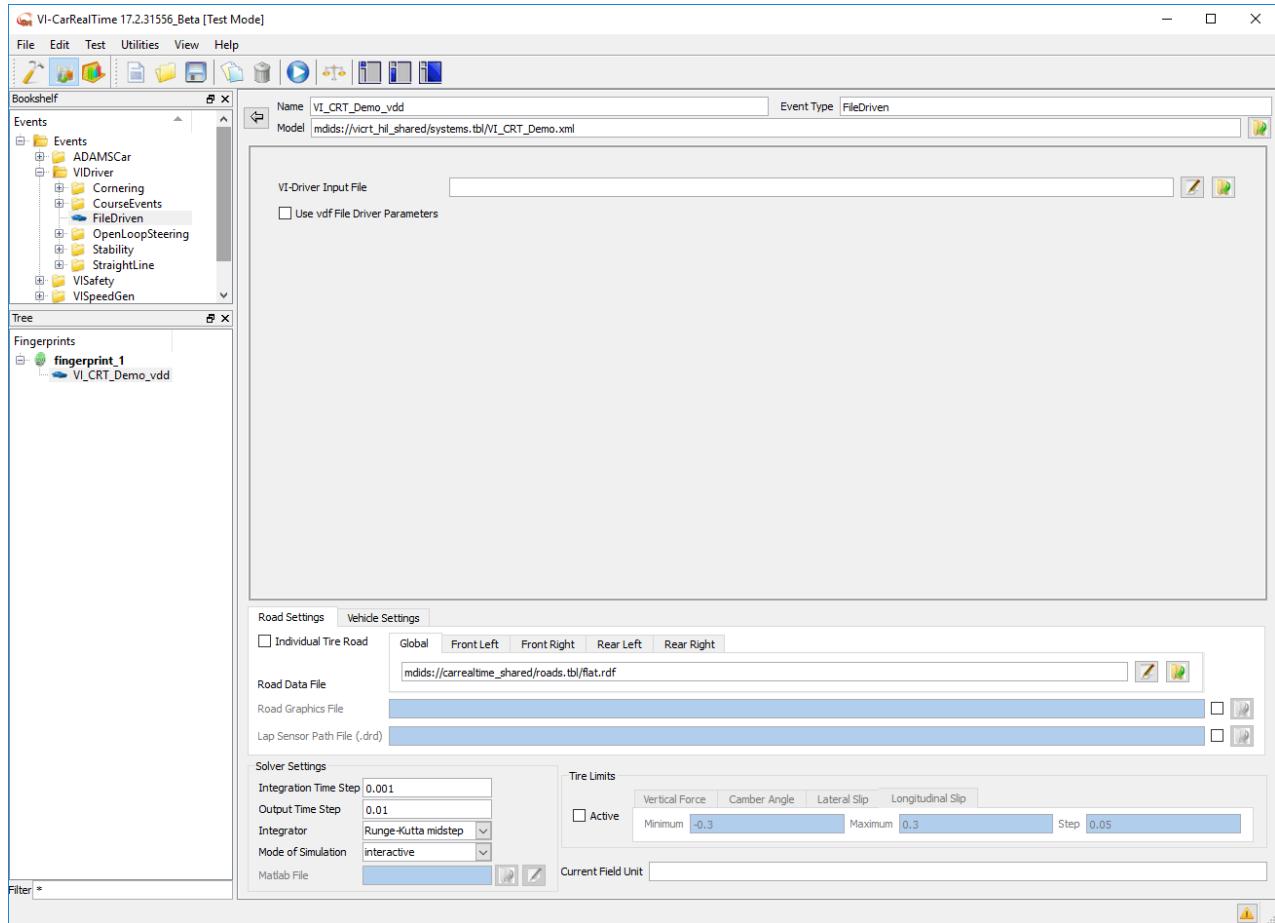
Then, **activate the circular buffer** by clicking on the root node of the VI_CRT_Demo project, switch to the **Properties** tab, activate the `system_parameters->circular_buffer` node and, then, set the activity flag to the value 1.



Now, you can switch to the **Test Mode** () where the event specification will be defined. Create a new fingerprint by expanding the **Events** hierarchy, right click on the **FileDriven** item and select **Add Event to New Fingerprint**.



You will end up with a new event specification as depicted in the following.

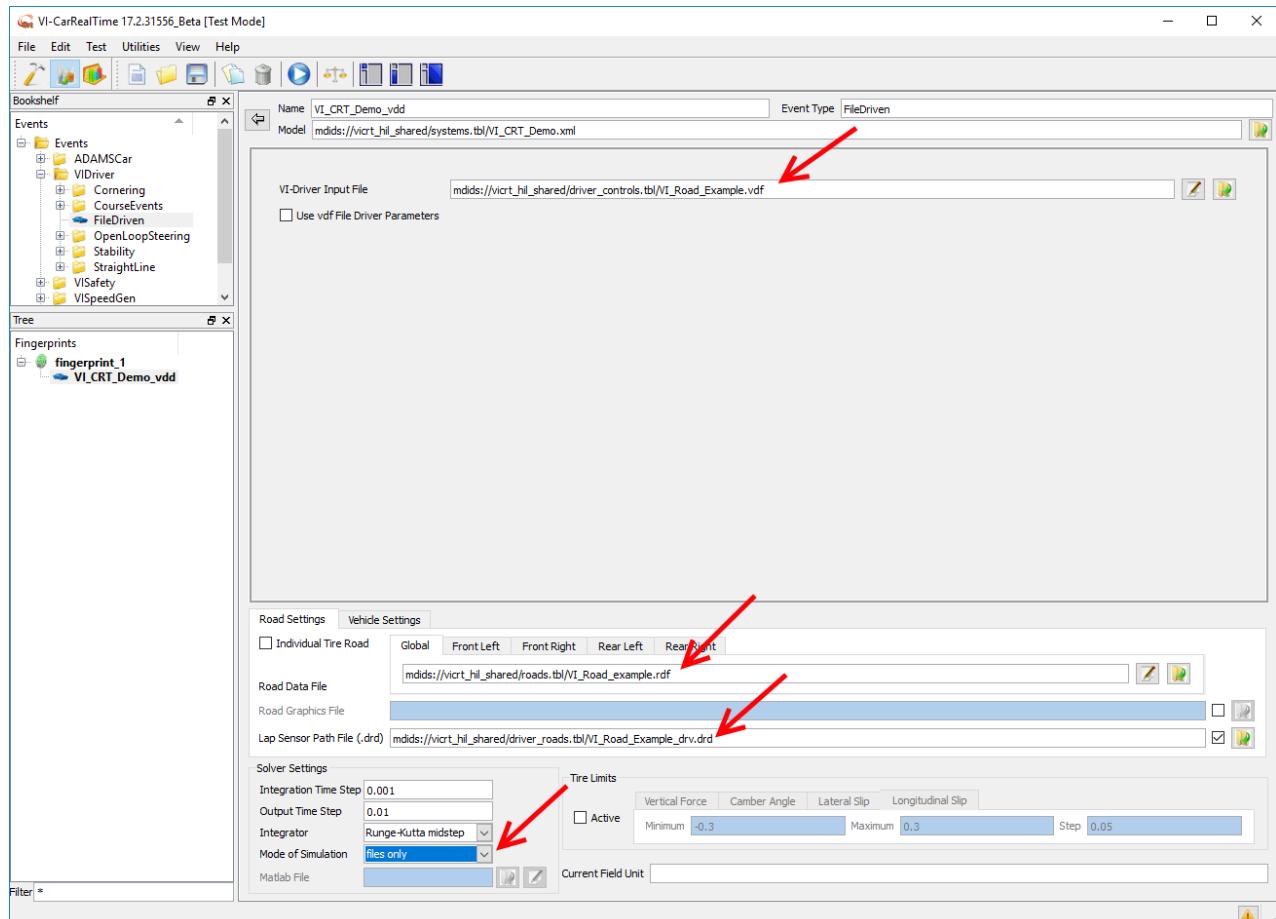


Now, the event specification must be completed as follows:

1. Select from the `vicrt_hil_shared` database the `VI_Road_Example.vdf` as the **VI-Driver input file**
2. Select from the `vicrt_hil_shared` database the `VI_Road_Example.rdf` as the **Road Data File**
3. Select from the `vicrt_hil_shared` database the `VI_Road_Example_drv.drd` file as the **Lap Sensor Path File**
4. Set the simulation mode to **files only**

You will end up with the following event specification:

VI-CarRealTime HIL Overlays



You can now run the maneuver () in order to generate the set of files needed by the VI-CarRealTime Matlab/Simulink interface. A console will appear with the following message:

```
File VI_CRT_Demo_vdd_send_svm.xml was successfully created..
```

The following files will be generated in the working directory:

```
VI_CRT_Demo_vdd.cmd
VI_CRT_Demo_vdd.obj
VI_CRT_Demo_vdd_graphics.xgr
VI_CRT_Demo_vdd_road_graphics.xgr
VI_CRT_Demo_vdd_send_svm.xml
VI_CRT_Demo_vdd_viani_live_startup.py
vicrt_cdb.cfg
viroad.mtl
```

Uploading to the target

If you followed through the previous steps of this tutorial, you ended up with a directory I:\vicrt. In order to upload everything needed to the target there are 2 steps to do:

1. double click on the script transfer_to_target.bat; the script will take care of the following:

- transferring to the target the VI-CarRealTime DLLs
- transferring to the target the license file

- creating on the target the directory `/vicrt` that mirrors the one located at `I:\vicrt` (namely, the `db` directory and the `crt` directory)
- creating the file `/vicrt/crt/vicrt_cdb.cfg` customized for the target machine

2. then, an **additional operation** that must be done manually:

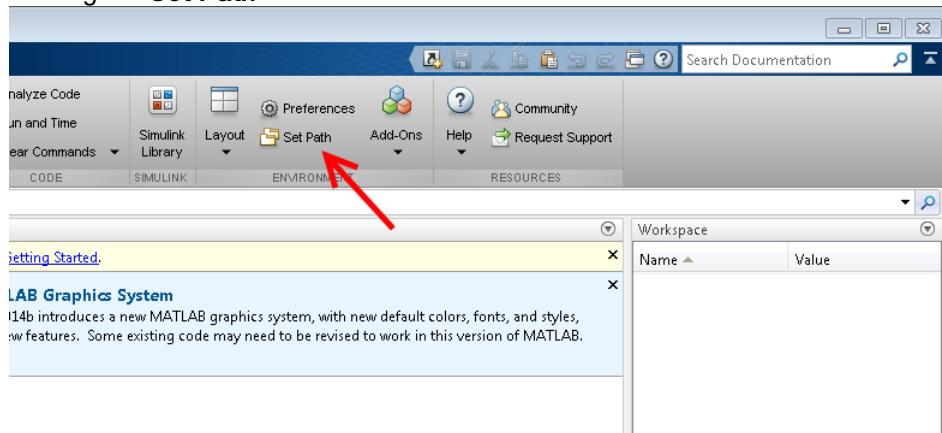
- open an FTP client and connect to the target machine
- transfer the local file `I:\vicrt\crt\VI_CRT_Demo_vdd_send_svm.xml` to the target machine at location `/vicrt/crt/VI_CRT_Demo_vdd_send_svm.xml`
- edit the `VI_CRT_Demo_vdd_send_svm.xml` on the target and change the `working_directory` field to the value `/vicrt/crt`, as in the following:

```
...
<StringParameter      name="working_directory"      active="true"      userDefined="false"
value="/vicrt/crt" />
...
...
```

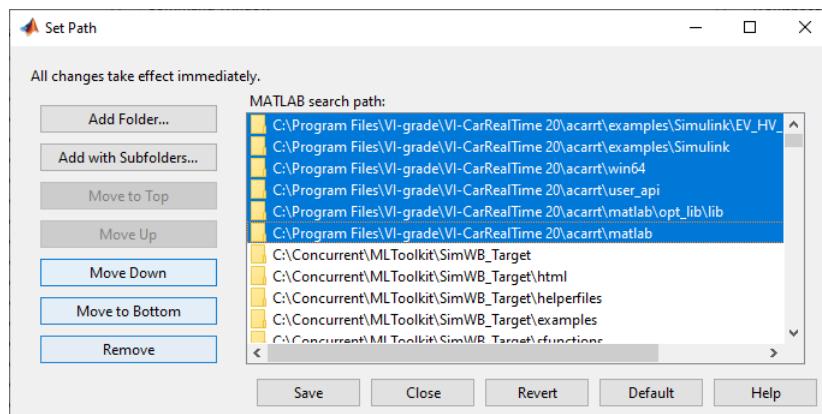
Setting up the Simulink Model

Start a Matlab session and switch to the `I:\vicrt\demo_matlab` directory.

If you are opening a Matlab session for the first time, add to Matlab search paths the VI-CarRealTime paths by using the "addpath_vicrt_20" script. You can check that the search paths have been configured correctly by clicking the **Set Path** button in the MATLAB toolbar:



You should end up with a list of search paths similar to the one in the following picture:

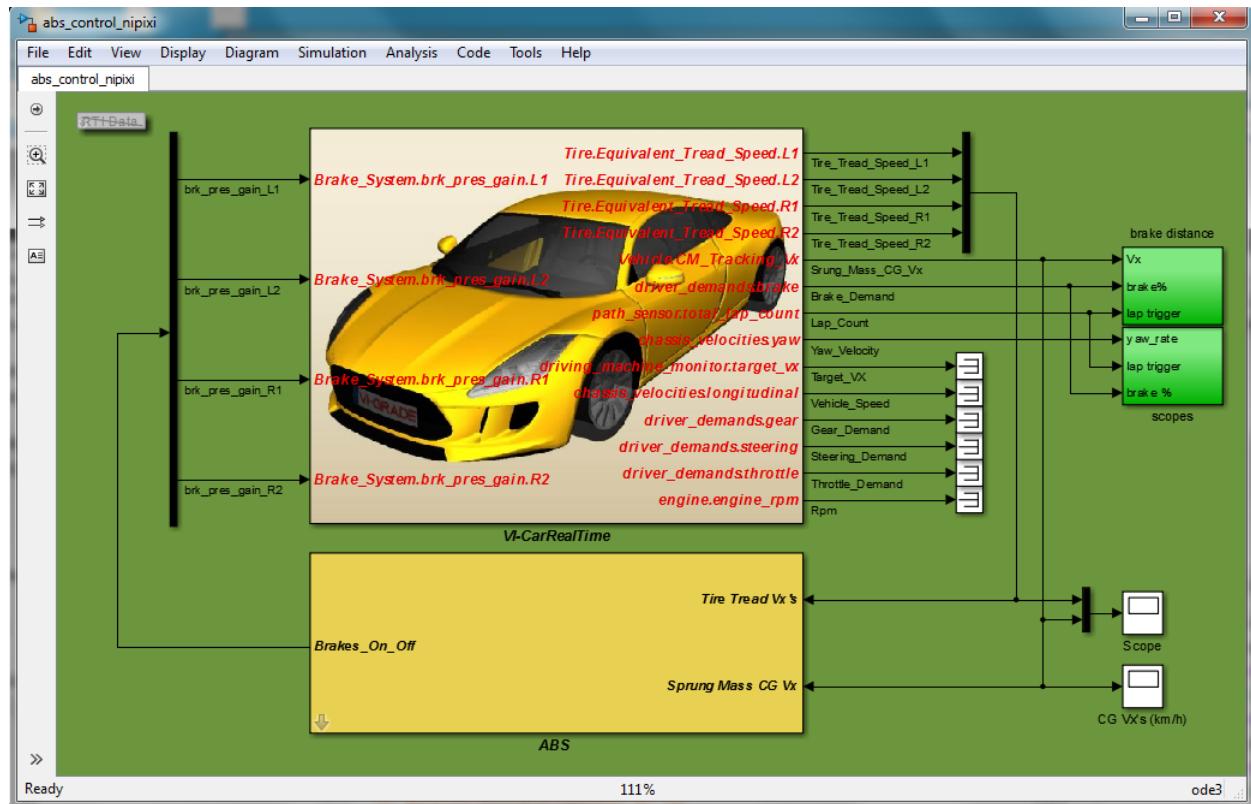


From your Matlab session, open the file `abs_control_nipxi.mdl`.

VI-CarRealTime HIL Overlays

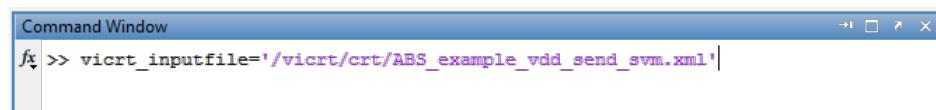
This model contains several Simulink subsystems:

- VI-CarRealTime: the solver Simulink interface;
- ABS Controller: the ABS controller implementation.

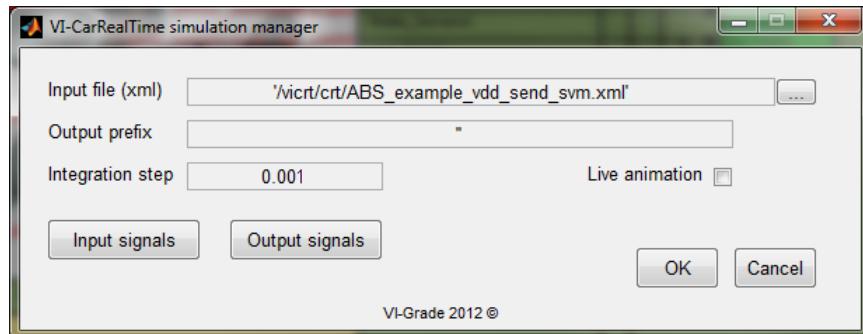


Now set:

- the VI-CarRealTime input file using the `vicrt_inputfile` variable.

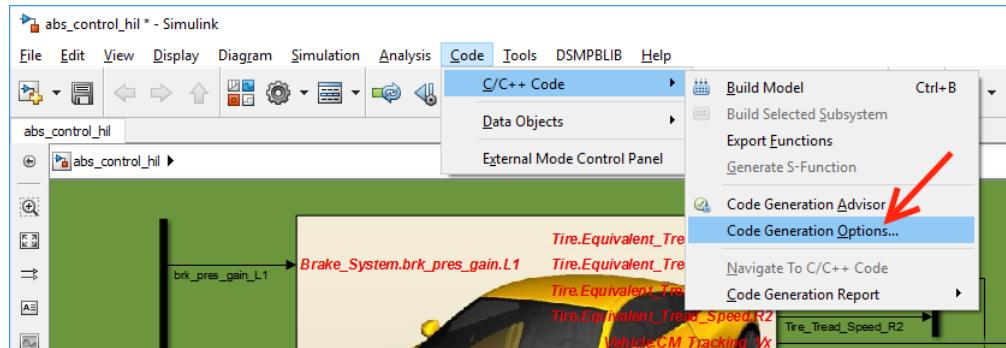


or using the VI-CarRealTime Solver mask

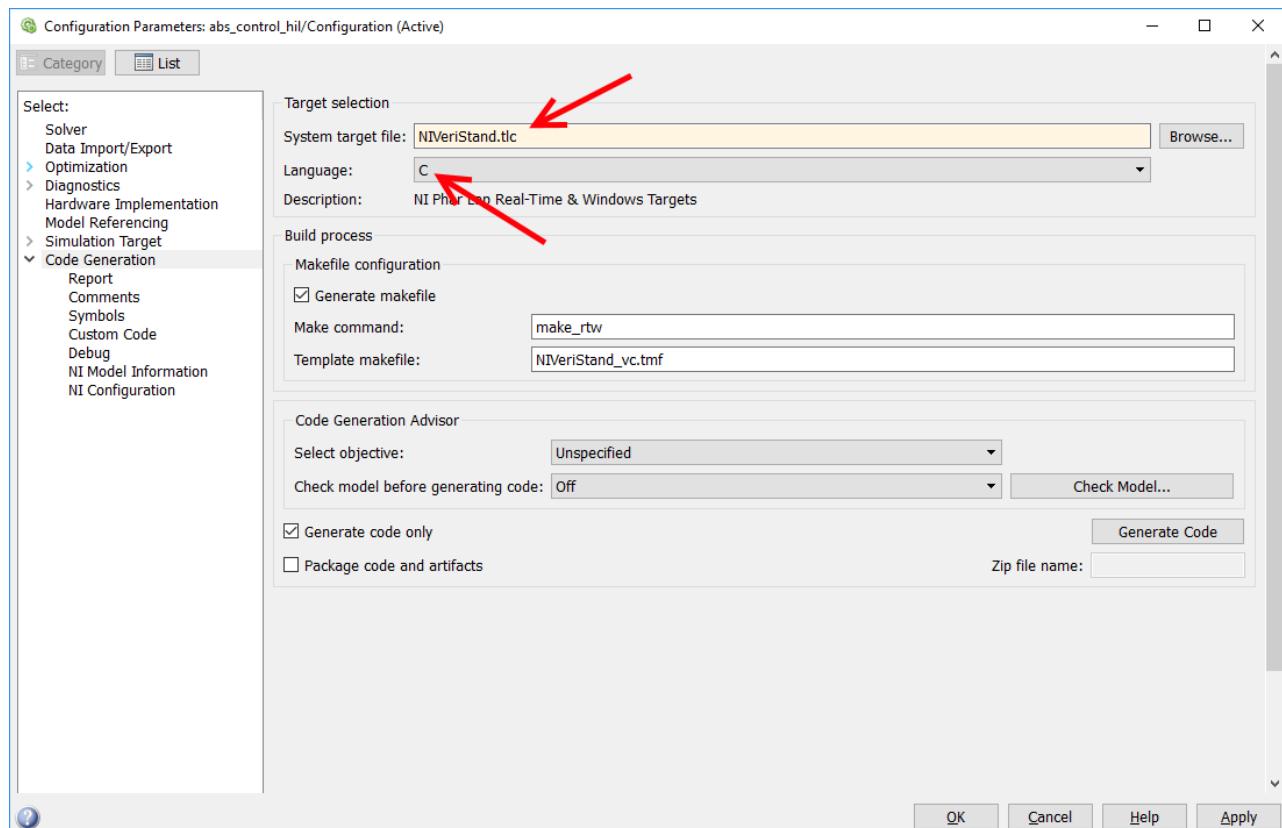


Note: The input file path must be the same as that used in the NI-PXI controller (please, notice that it does not contain the drive letter).

Now, please open the **Code Generation** options of the Simulink model:

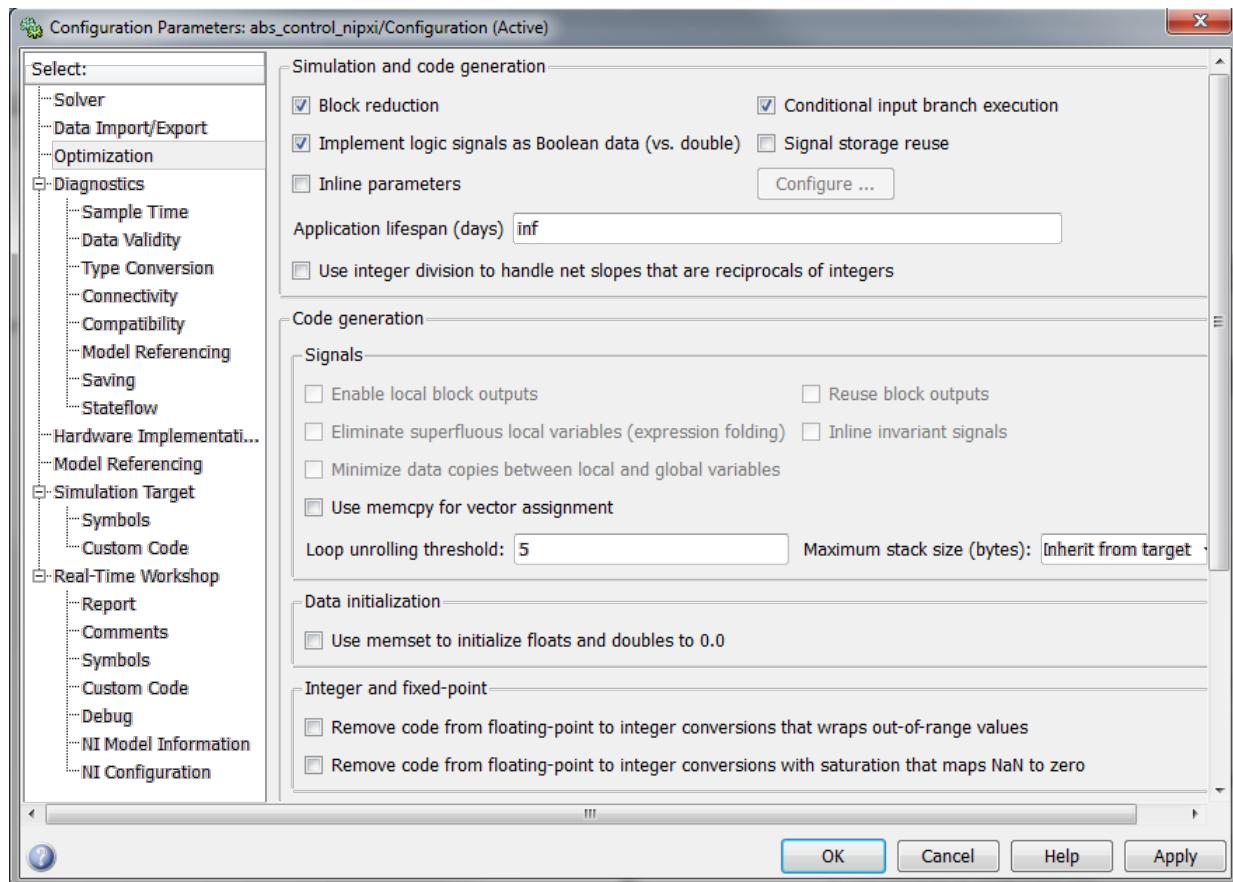


Under the category **Code Generation**, set the **System Target File** to NIVeristand.tlc and the **Language** field to C (so that Simulink will use the NI-Veristand tool chain to target the NI-PXI machine).



After that, under the category **Optimization pane** check the optimization option as in the figure below (for more details refer to the NI-Veristand documentation).

VI-CarRealTime HIL Overlays



Under the category **Code Generation** and subcategory **Custom Code**, fill in the fields **Include Directories**, **Source Files** and **Libraries** as described in the following:

- **Include Directories:**

```
"%CRT_DIR%\acarrt\user_api"
```

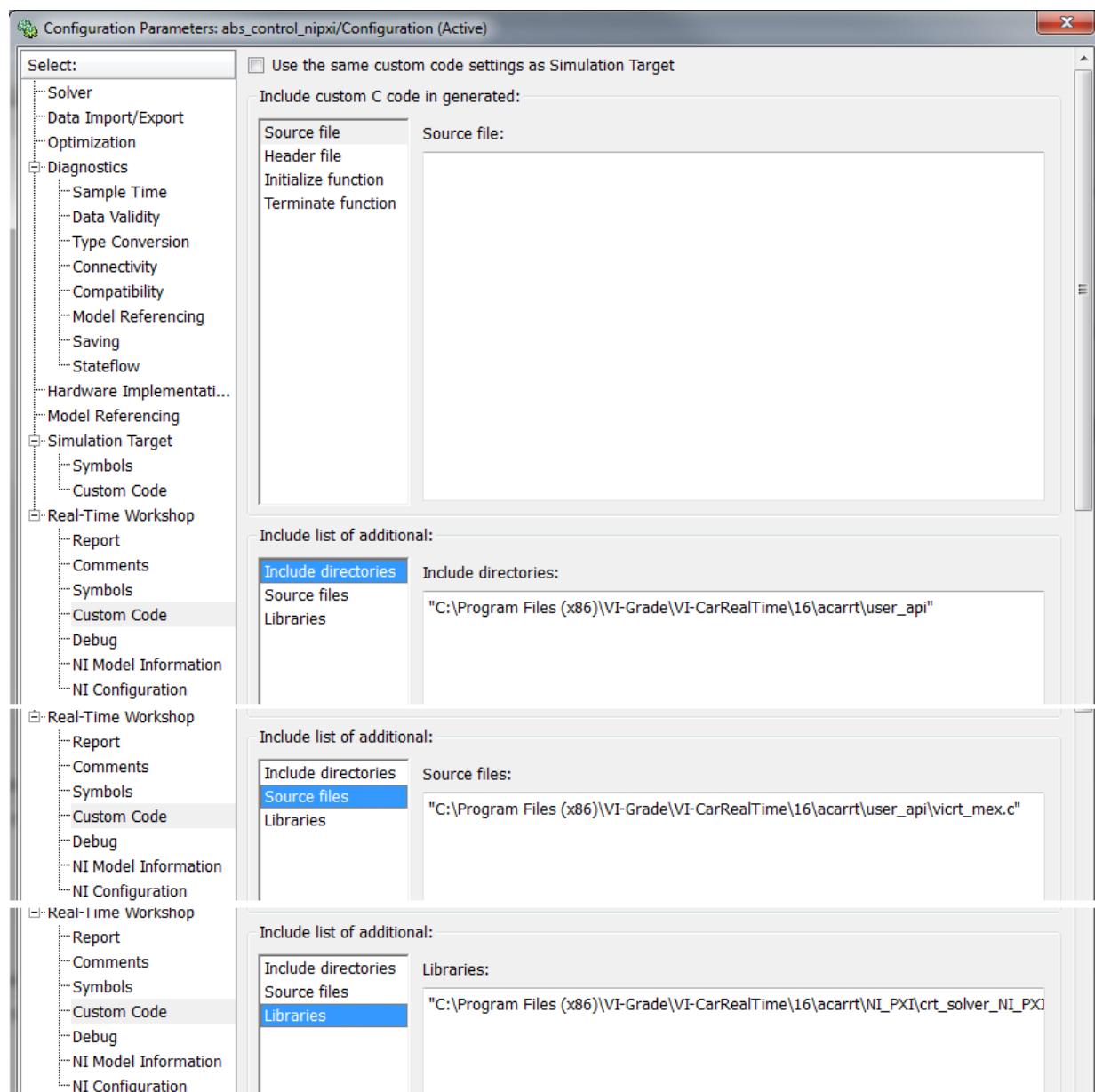
- **Source Files:**

```
"%CRT_DIR%\acarrt\user_api\vicrt_mex.c"
```

- **Libraries:**

```
"%CRT_DIR%\acarrt\NI_PXI\crt_solver_imp.lib"
```

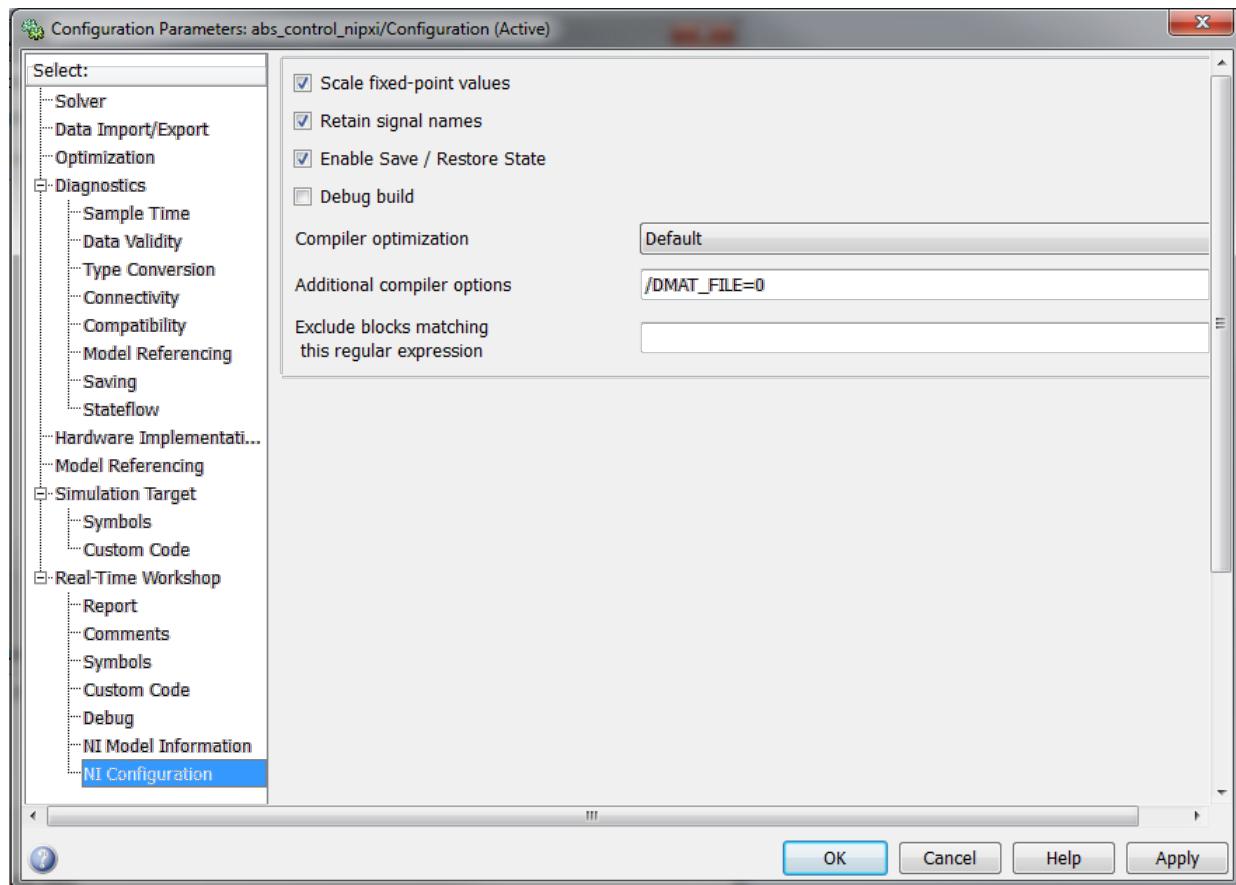
You should end up with the following settings:



Note: In order to avoid system path resolution error, remember to enclose path string between ".

Finally, in order to avoid application crash due to the logging system in case of initialization error, under the category **Code Generation** and subcategory **NI Configuration** pane, set the compiler option **/DMAT_FILE=0**

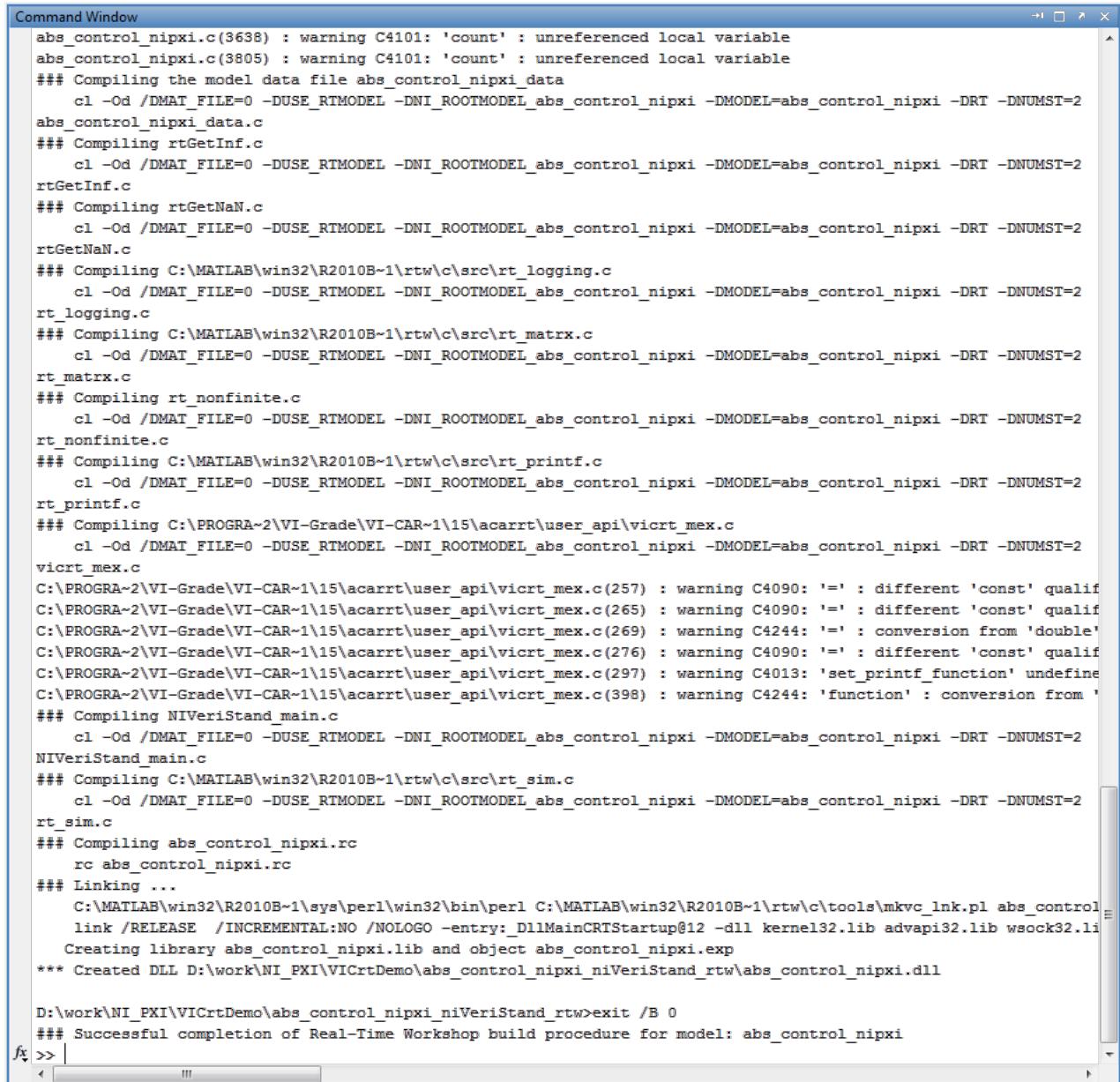
VI-CarRealTime HIL Overlays



Building the application

After configuring the building environment, the Simulink model can be built using the **Build** button placed in the **Simulink toolbar** (or using the **Ctrl+B** shortcut).

The compile procedure reports on the matlab command window the compile commands executed.



```

Command Window
abs_control_nipxi.c(3638) : warning C4101: 'count' : unreferenced local variable
abs_control_nipxi.c(3805) : warning C4101: 'count' : unreferenced local variable
### Compiling the model data file abs_control_nipxi_data
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
abs_control_nipxi_data.c
### Compiling rtGetInf.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rtGetInf.c
### Compiling rtGetNaN.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rtGetNaN.c
### Compiling C:\MATLAB\win32\R2010B~1\rtw\c\src\rt_logging.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rt_logging.c
### Compiling C:\MATLAB\win32\R2010B~1\rtw\c\src\rt_matrx.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rt_matrx.c
### Compiling rt_nonfinite.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rt_nonfinite.c
### Compiling C:\MATLAB\win32\R2010B~1\rtw\c\src\rt_printf.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rt_printf.c
### Compiling C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
vicrt_mex.c
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(257) : warning C4090: '=' : different 'const' qualif
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(265) : warning C4090: '=' : different 'const' qualif
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(269) : warning C4244: '=' : conversion from 'double'
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(276) : warning C4090: '=' : different 'const' qualif
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(297) : warning C4013: 'set_printf_function' undefined
C:\PROGRA~2\VI-Grade\VI-CAR~1\15\acarrt\user_api\vicrt_mex.c(398) : warning C4244: 'function' : conversion from 'void' to 'int'
### Compiling NIVeriStand_main.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
NIVeriStand_main.c
### Compiling C:\MATLAB\win32\R2010B~1\rtw\c\src\rt_sim.c
  cl -Od /DMAT_FILE=0 -DUSE_RTMODEL -DNI_ROOTMODEL_abs_control_nipxi -DMODEL=abs_control_nipxi -DRT -DNUMST=2
rt_sim.c
### Compiling abs_control_nipxi.rc
  rc abs_control_nipxi.rc
### Linking ...
  C:\MATLAB\win32\R2010B~1\sys\perl\win32\bin\perl C:\MATLAB\win32\R2010B~1\rtw\c\tools\mkvc_lnk.pl abs_control_nipxi.rc
  link /RELEASE /INCREMENTAL:NO /NOLOGO -entry:_DllMainCRTStartup@12 -dll kernel32.lib advapi32.lib ws2_32.lib
  Creating library abs_control_nipxi.lib and object abs_control_nipxi.exp
*** Created DLL D:\work\NI_PXI\VICrtDemo\abs_control_nipxi_niVeriStand_rtw\abs_control_nipxi.dll

D:\work\NI_PXI\VICrtDemo\abs_control_nipxi_niVeriStand_rtw>exit /B 0
### Successful completion of Real-Time Workshop build procedure for model: abs_control_nipxi

```

If the build procedure completes successfully, a sub-folder `abs_control_nipxi_niVeriStand_rtw` will be generated under the `I:\vicrt\demo_matlab` folder. That sub-folder will contain the `abs_control_nipxi.dll` file that will eventually be uploaded to the target NI-PXI machine.

Monitoring the application with VeriStand

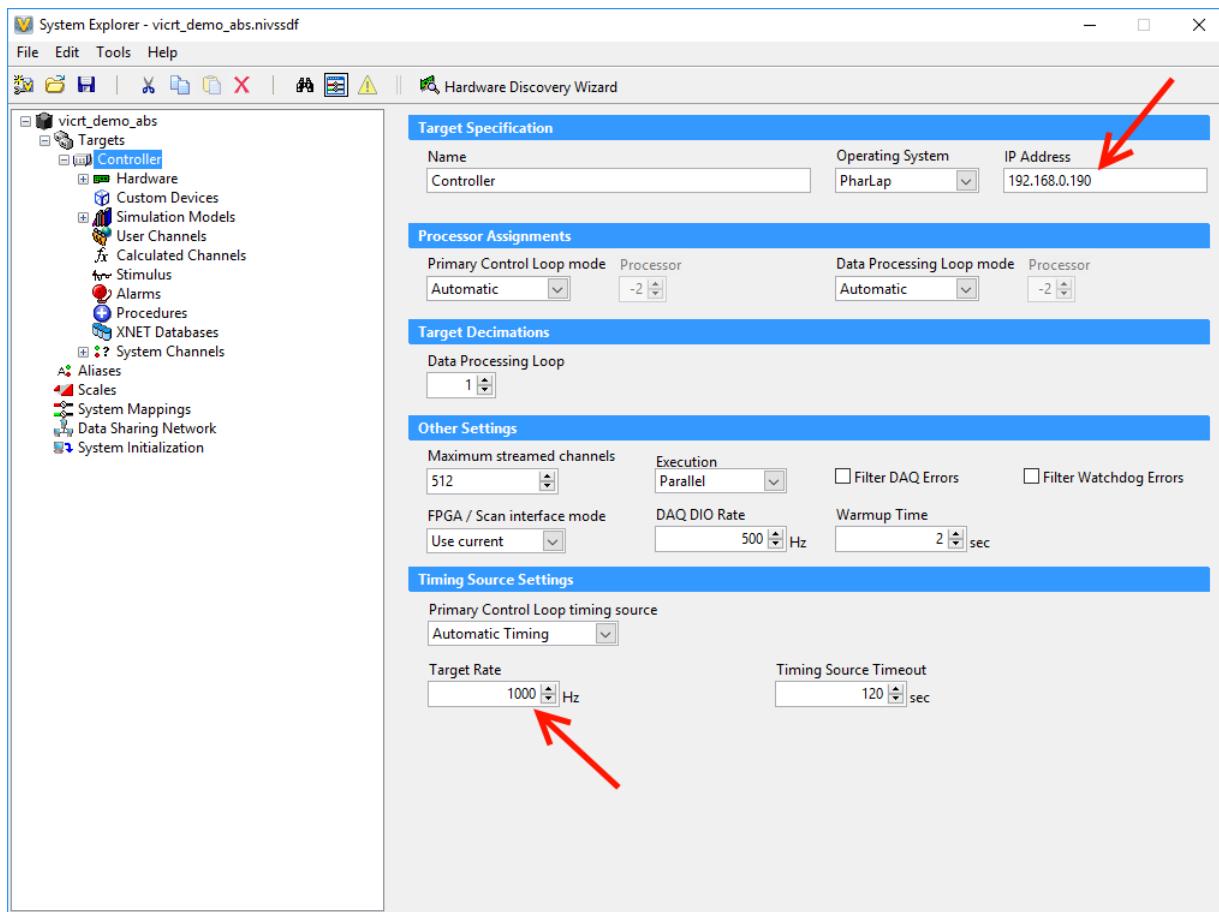
Now, launch Veristand by double-clicking on the script located at the following:

```
I:\vicrt\launch_veristand.bat
```

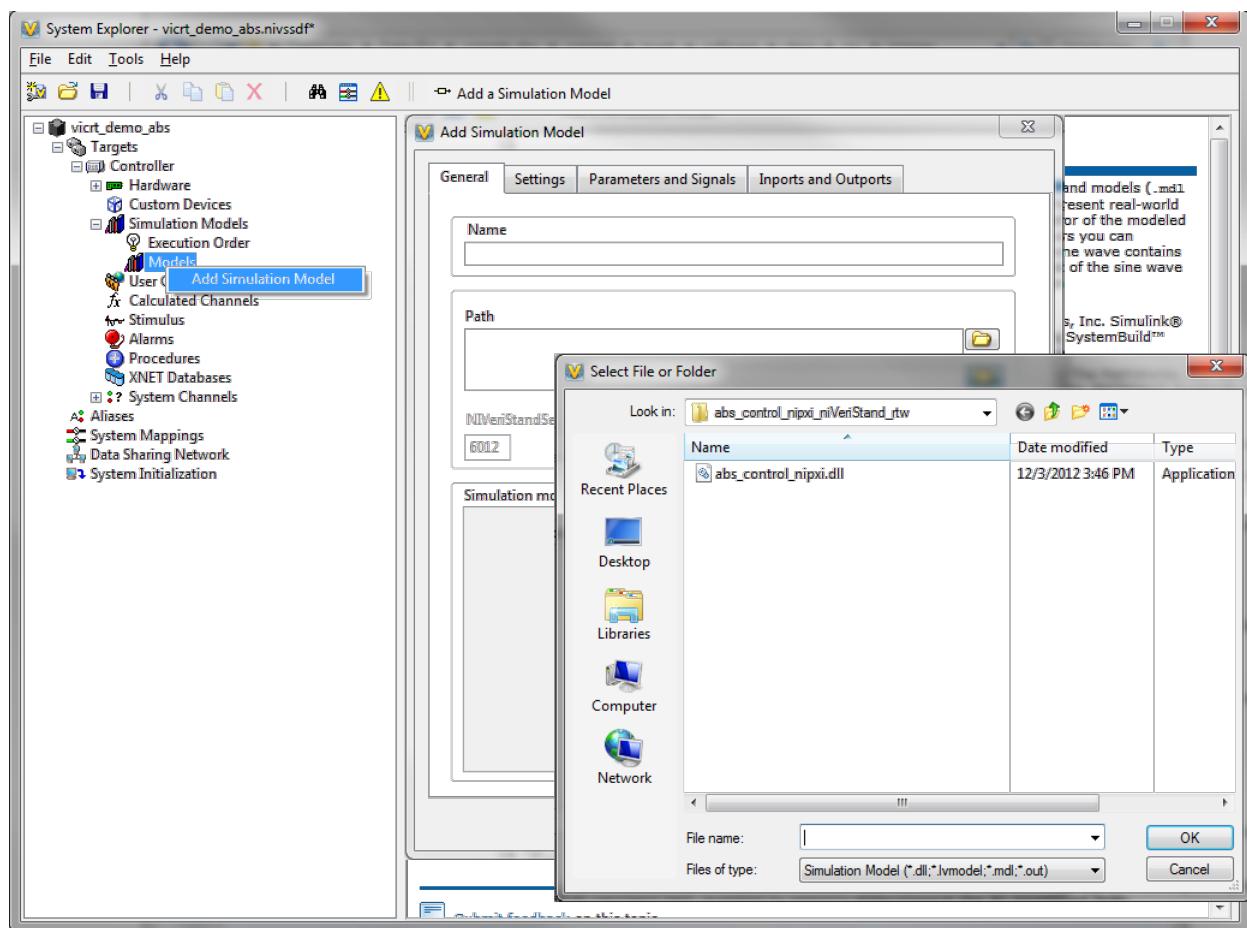
The script will open the Veristand project located at `I:\vicrt\demo_veristand\vicrt_demo_abs.nivsproj`. Please, notice that the script launches Veristand with a modified environment so that the simulation model created in Simulink can be correctly added to the system definition; launching Veristand without modifying the environment would produce a failure while adding the simulation model.

VI-CarRealTime HIL Overlays

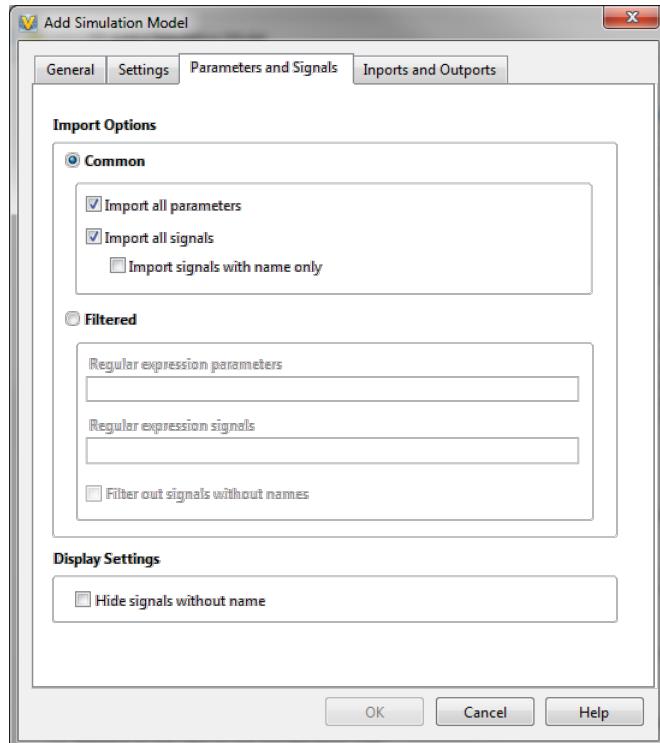
Then, edit the **System Definition File** (by double-clicking the **vicrt_demo_abs** item under **System Definition File**) and set the controller IP address in order to use the correct controller and set the **Target Rate** to 1000Hz.



Now, using the **Add Simulation Model** command, add the `abs_control_nipxi.dll` from the `abs_control_nipxi_niVeriStand_rtw` sub-folder generated by the Simulink build procedure.



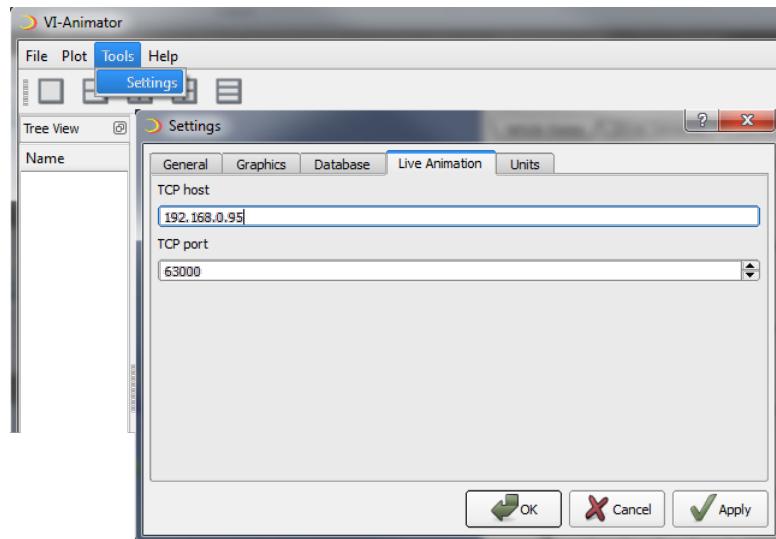
Before completing the simulation model import, switch to the **Parameter and Signals** tab and check if the import flag are enabled.



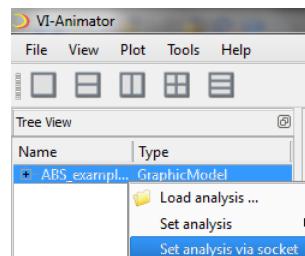
VI-CarRealTime HIL Overlays

Now, save and close the system definition. You can now deploy to the target machine.

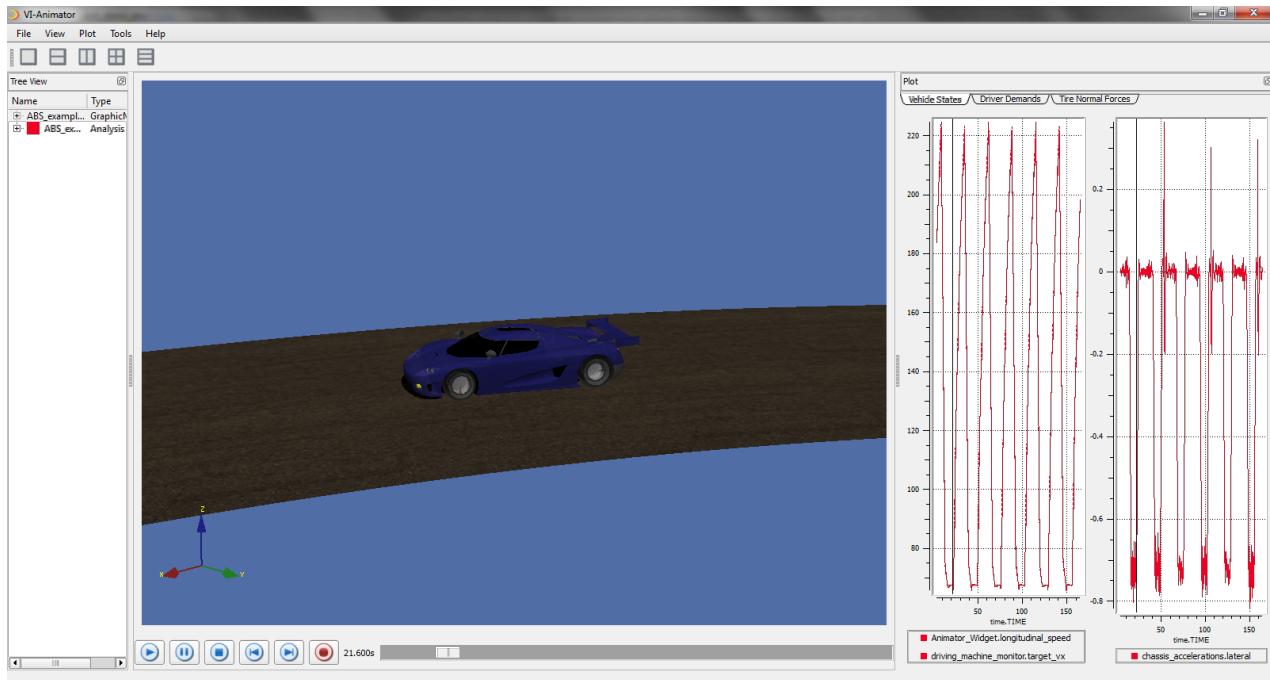
After deploying the application on the NI-PXI controller, start the VI-Animator, and set the remote machine IP address and the communication port.



After that, load the `ABS_example_vdd.xgr` graphic file generated in the local machine working directory and set the analysis via socket.



Now VI-Animator show in real time the vehicle state.



FMI with the NI-Veristand Interface

This tutorial shows how to run a VI-CarRealTime on the NI-PXI target by means of the VI-CarRealTime NI-Veristand interface.

The advantage of using this approach is that there is no need to use Simulink to cross-compile your model for the NI-PXI target. The functionality of VI-CarRealTime is made available through a couple of ready made models; those models only need to be added to the NI-Veristand project and be connected to the rest of the hardware-in-the-loop experiment. The couple of models can be described as follows:

- the first model implements the VI-CarRealTime solver
- the second model provides the default values for the solver inputs

As later explained, you will normally start by adding both those models to the project, then connect all the outputs of the default values provider to the solver inputs. Then, if needed, you will rewire the signals connections according to your specific needs (other models, acquisition peripherals, etc...).

This tutorial considers the case analyzed in the chapter FMI: External Driveline and provides instructions to run the same experiment on the NI-PXI target. It is divided in the following steps:

[Setting up the working directory](#)

[Running the FMI experiment locally](#)

[Preparing the import of the NI-Veristand Interface](#)

[Uploading to the target](#)

[Running the FMI experiment on the target with Veristand](#)

Setting up the working directory

In order to create the working directory, please double click on the following batch file:

```
%CRT_DIR%\acarrt\examples\NI_PXI\nipxi_prepare_tutorial.bat
```

The script will execute the following sequence of steps (that can be interrupted by pressing the key combination CTRL+C):

1. Collect information: local working directory for the tutorial, IP address of the target NI-PXI machine, FTP

VI-CarRealTime HIL Overlays

- credentials, VI-grade license file
- 2. Check that the target NI-PXI machine can be reached
- 3. Make an FTP connection
- 4. Prepare the local working directory
- 5. Copy VI-CarRealtime databases to the local working directory
- 6. Copy NI-Veristand demo project to the local working directory
- 7. Copy MATLAB demo project to the local working directory
- 8. Copy support python scripts...
- 9. Create script to launch NI-Veristand with the modified PATH env. variable so that the Simulink model DLL can be correctly added to the System Definition File
- 10. Create script for transferring VI-CarRealtime DLLs, VI-CarRealtime shared databases and license files to the target NI-PXI via FTP
- 11. Summary

There are **3 steps that deserve attention:** Step 1., Step 2. and Step 3..

During **Step 1.**, the user will be asked to type in the following information:

- **local working directory:** this directory will be used for copying all the files required for completing the tutorial - the working directory must be not existent (that's because it will be created automatically) and must be in the root of the drive (e.g., I:\vicrt)
- **IP address of the target:** this is the IP address of the NI-PXI machine where the model will be running
- **FTP credentials of the target:** these are the FTP username and password of the NI-PXI machine
- **license file for the target:** this is the absolute path to the license file that VI-grade has generated for the specific target NI-PXI machine

Please, collect all these pieces of information *before* running `nipxi_prepare_tutorial.bat`. Here is an example of running the script and typing in the required information:

```
...
1. Collect information: local working directory for the tutorial, IP address of the target
NI-PXI machine, FTP credentials, VI-grade license file

Please type in a valid directory path (the directory must be NOT existent and must be
located in the root of an hard drive): I:\vicrt

Please type in a valid IP address for the target NI-PXI machine: 192.168.0.190
Please type in the FTP username for accessing the target NI-PXI machine: admin
Please type in the FTP password:
Please type in the absolute path of a license file for the target NI-PXI machine: I:
\ViGRADE_9340.lic
...

```

During **Step 2.**, the target machine will be *ping-ed* in order to check that the specified IP address is reachable. During **Step 3.**, an FTP connection will be attempted with the specified user name and password. During these steps, if anything fails, you will be asked to check that the machine is on and that it is connected to the network; then, you will be able to make additional attempts.

Let's suppose that you have chosen the path I:\vicrt as your working directory. You will end up with the following directories/files located in I:\vicrt:

- `crt`: this is the directory where the maneuver file will be generated (see [Generating the maneuver file](#))
- `assets`: this directory contains the VI-CarRealTime database and a copy of the specified VI-grade license file for the NI-PXI target
- `demo_matlab`: this directory contains the demo MATLAB/Simulink model (see [Setting up the Simulink Model](#))

- `demo_veristand`: this directory contains the demo Veristand project (see [Monitoring the application with VeriStand](#))
- `scripts`: this directory contains support scripts
- `nivs`: this directory contains facilities for launching Veristand together with the VI-CarRealTime Veristand interface (see [FMI with the NI-Veristand Interface](#))
- `launch_veristand.bat`: this is the script to launch Veristand (see [Monitoring the application with VeriStand](#))
- `transfer_to_target.bat`: this is the script to transfer to the NI-PXI target machine all the files needed by VI-CarRealTime in order to execute the simulation (see [Uploading the maneuver to the target](#))
- `transfer_binaries_only.bat`: this is the script to transfer to the NI-PXI target machine only the binary files (this proves to be useful when the target NI-PXI machine has been previously setup and used and now needs to be updated with a new release of the VI-CarRealTime overlay for NI-PXI)

Running the FMI experiment locally

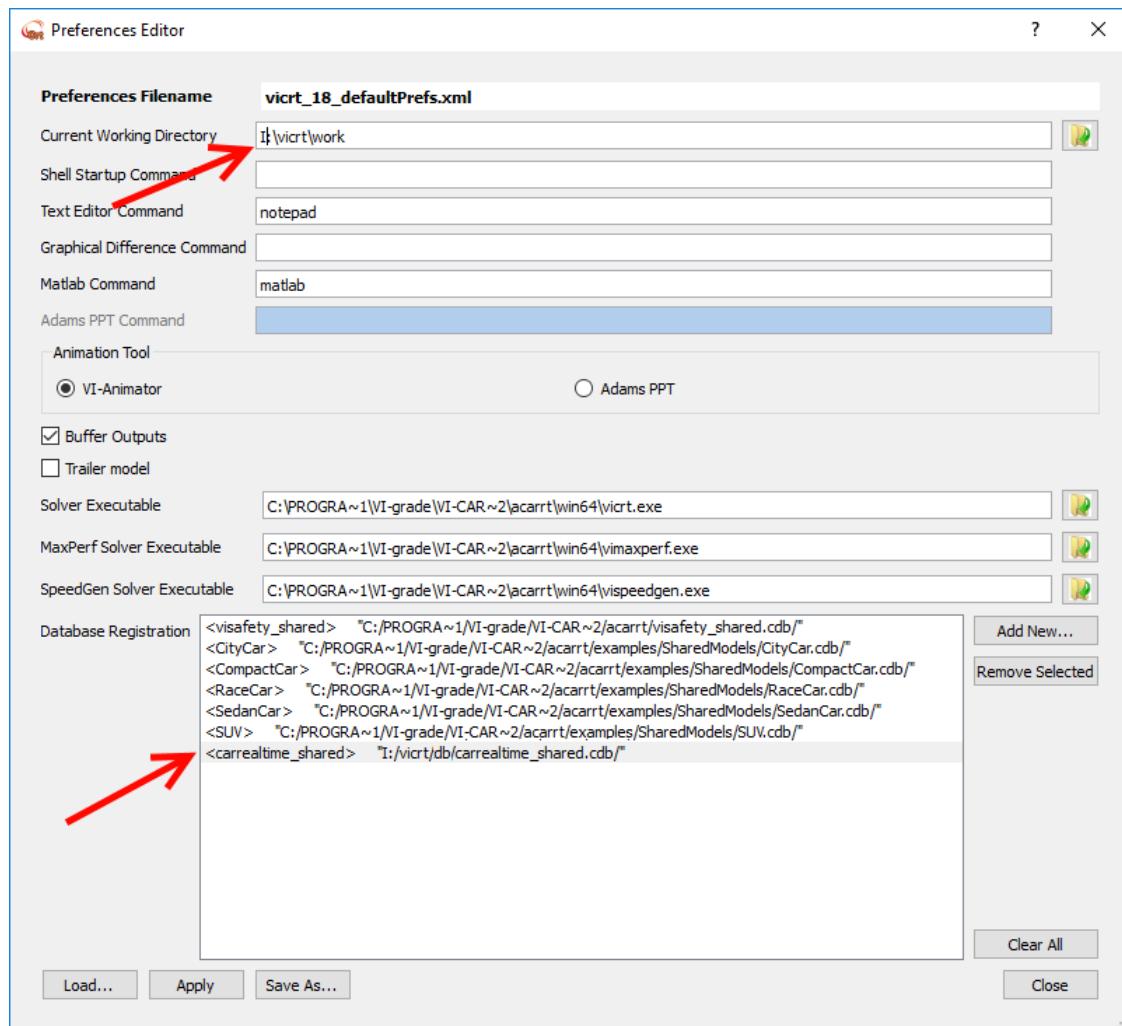
Once you have prepared your working directory in `I:\vicrt\crt`, it is highly recommended to run the FMI experiment on the desktop PC before running it on the NI-PXI target. That experiment is already documented in chapter FMI: External Driveline therefore we will not repeat it here. The only caveat is that you must execute the FMI experiment under your working directory so please, execute the following instructions.

Please, start a new VI-CarRealTime session using the **vicrt20** command from a dos shell or by using the shortcut from Windows Start Menu.

- Set up the VI-CarRealTime working directory to `I:\vicrt\crt`.
- Set up the VI-CarRealTime shared database in order to use the one placed under `I:\vicrt\assets\db\carrealtime_shared.cdb`.

Both of these operations can be done in VI-CarRealTime by choosing the menu Edit->Preferences and set the fields as highlighted in the following picture:

VI-CarRealTime HIL Overlays



Now, follow the instructions described in the chapter FMI: External Driveline. Once you have finished that tutorial, please, continue with the next steps of this tutorial.

Preparing the import of the NI-Veristand Interface

In order for the models of the VI-CarRealTime NI-Veristand Interface to be successfully imported, 2 operations must be done:

- the live animation must be disabled
- the path to the maneuver file must be provided

If you followed through the previous steps, the maneuver file should be located at the following:

```
I:\vicrt\crt\VI_CRT_Demo_vdd_send_svm.xml
```

Open the maneuver file with a text editor and make sure that the live animation is disabled as described in the following:

```
...
<IntParameter name="live_animation" active="true" userDefined="false" value="0"/>
...
```

Then, create a text file at the following location:

```
I:\vicrt\crt\vicrt_nivs.cfg
```

Open the `vicrt_nivs.cfg` file with a text editor and insert the following line containing the absolute path to the maneuver file:

```
I:\vicrt\crt\VI_CRT_Demo_vdd_send_svm.xml
```

Then, save and close the file.

Note: The file `vicrt_nivs.cfg` will be read by the VI-CarRealTime NI-Veristand Interface during the phase of importing the model into Veristand. This is needed in order to export the correct set of input and output channels.

Uploading to the target

If you followed through the previous steps of this tutorial, you ended up with a directory `I:\vicrt`. In order to upload everything needed to the target, these are the steps to do:

1) double click on the script `transfer_to_target.bat`; the script will take care of the following:

- transferring to the target the VI-CarRealTime DLLs
- transferring to the target the license file
- creating on the target the directory `/vicrt` that mirrors the one located at `I:\vicrt` (namely, the `db` directory and the `crt` directory)
- creating the file `/vicrt/crt/vicrt_cdb.cfg` customized for the target machine

2) then, **additional manual operations** that must be done on the target:

- a) open an FTP client and connect to the target machine
- b) transfer the local file `I:\vicrt\crt\VI_CRT_Demo_vdd_send_svm.xml` to the target machine at location `/vicrt/crt/VI_CRT_Demo_vdd_send_svm.xml`
- c) edit the `VI_CRT_Demo_vdd_send_svm.xml` on the target and apply the following changes:
 - i) change the `working_directory` field to the value `/vicrt/crt`, as in the following...

```
...
<StringParameter    name="working_directory"    active="true"    userDefined="false"
value="/vicrt/crt" />
...
```

- ii) change the `live_animation` field in order to make live animation available if needed, as in the following:

```
...
<IntParameter      name="live_animation"      active="true"      userDefined="false"
value="1" />
...
```

- iii) change the `FMI Master` field in order to instruct the solver on the correct location of the `Driveline2WD.vfm`, as in the following:

```
...
<FMIMaster        name="fmi_master"        active="true"        userDefined="false"
```

VI-CarRealTime HIL Overlays

- ```
configurationFile="file:///vicrt/crt/Driveline2WD.vfm"
library="vicrt_fmimaster.dll" activeFlag="true"/>
...
d) transfer the local file I:\vicrt\crt\Driveline2WD.fmu to the target machine at location /vicrt/crt/Driveline2WD.fmu
e) transfer the local file I:\vicrt\crt\Driveline2WD.vfm to the target machine at location /vicrt/crt/Driveline2WD.vfm
f) edit the Driveline2WD.vfm on the target and change the FMU_PATH field in order to instruct the solver on the correct location of the Driveline2WD.fmu, as in the following:
...
[FMU_1]
FMU_PATH='/vicrt/crt/Driveline2WD.fmu'
...
g) transfer the local file I:\vicrt\crt\vicrt_nivs.cfg to the target machine at location /nirt/system/vicrt_nivs.cfg
h) edit the vicrt_nivs.cfg on the target so that its contents are a single line with the absolute path to the maneuver file on the target, as in the following:
```

```
/vicrt/crt/VI_CRT_Demo_vdd_send_svm.xml
```

The target NI-PXI machine is now provided with all the files that are needed to run the experiment.

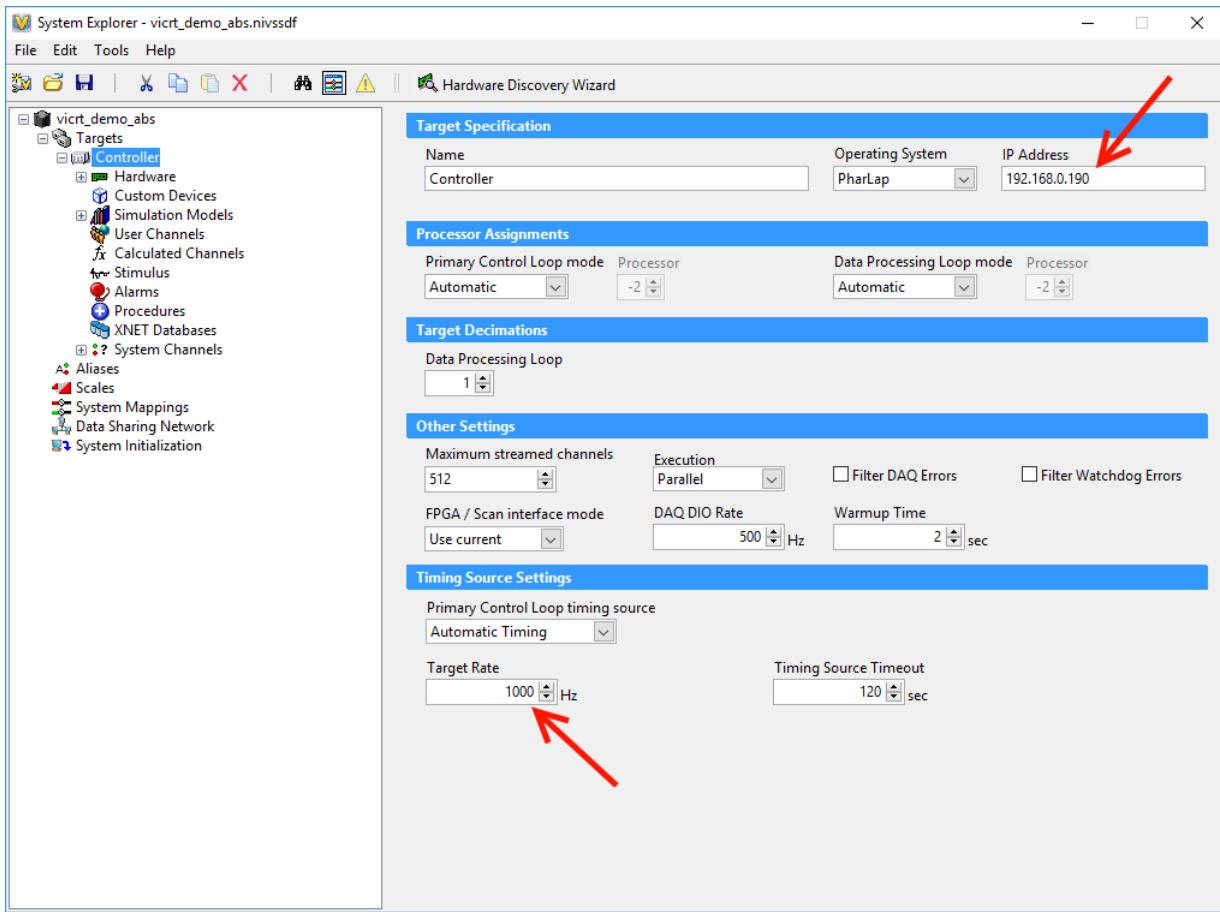
### Running the FMI experiment on the target with Veristand

Now, launch Veristand by double-clicking on the script located at the following:

```
I:\vicrt\nivs\launch_veristand.bat
```

The script will open the Veristand project located at I:\vicrt\demo\_veristand\vicrt\_demo\_abs.nivsproj. Please, notice that the script launches Veristand with a modified environment so that the simulation model created in Simulink can be correctly added to the system definition; launching Veristand without modifying the environment would produce a failure while adding the simulation model.

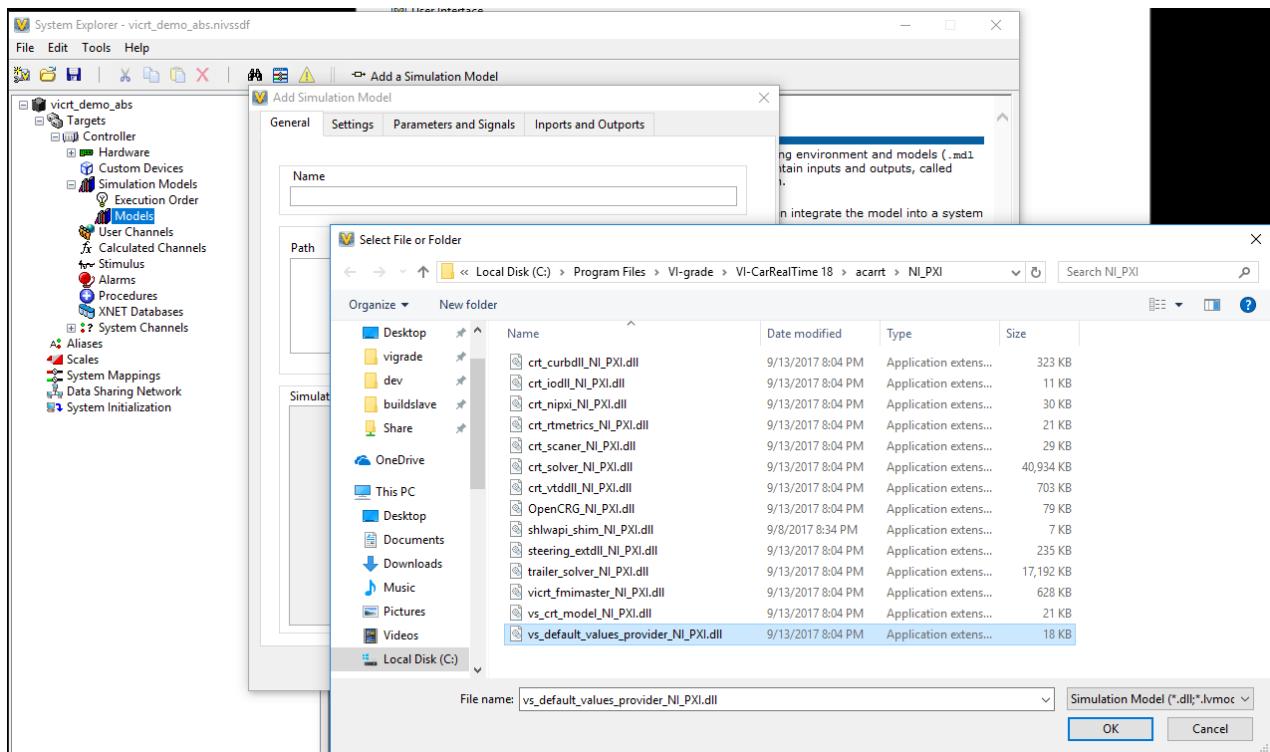
Then, edit the **System Definition File** (by double-clicking the **vicrt\_demo\_abs** item under **System Definition File**) and set the controller IP address in order to use the correct controller and set the **Target Rate** to 1000Hz.



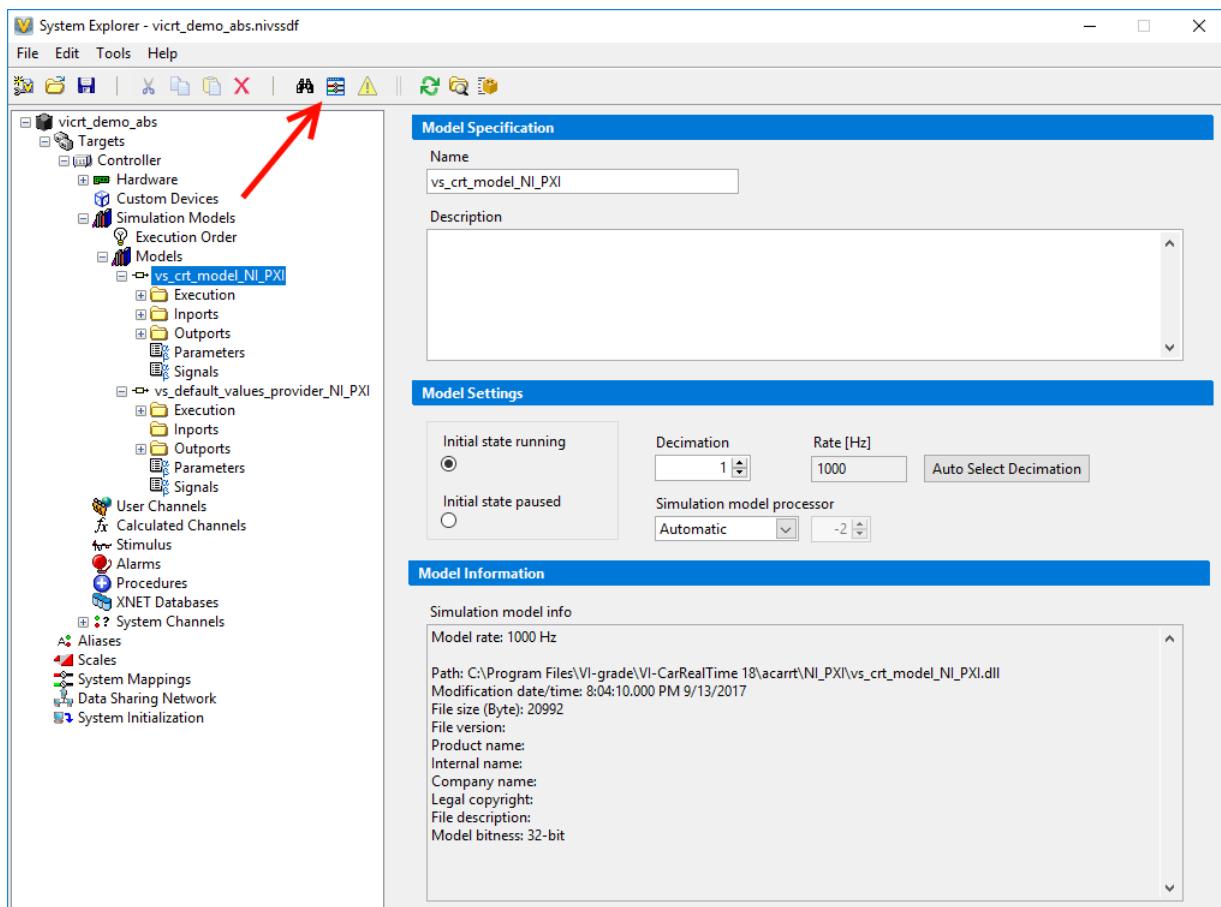
Now, using the **Add Simulation Model** command, add the following DLLs:

- %CRT\_DIR%\acarrt\NI\_PXI\vs\_crt\_model\_NI\_PXI.dll
- %CRT\_DIR%\acarrt\NI\_PXI\vs\_default\_values\_provider\_NI\_PXI.dll

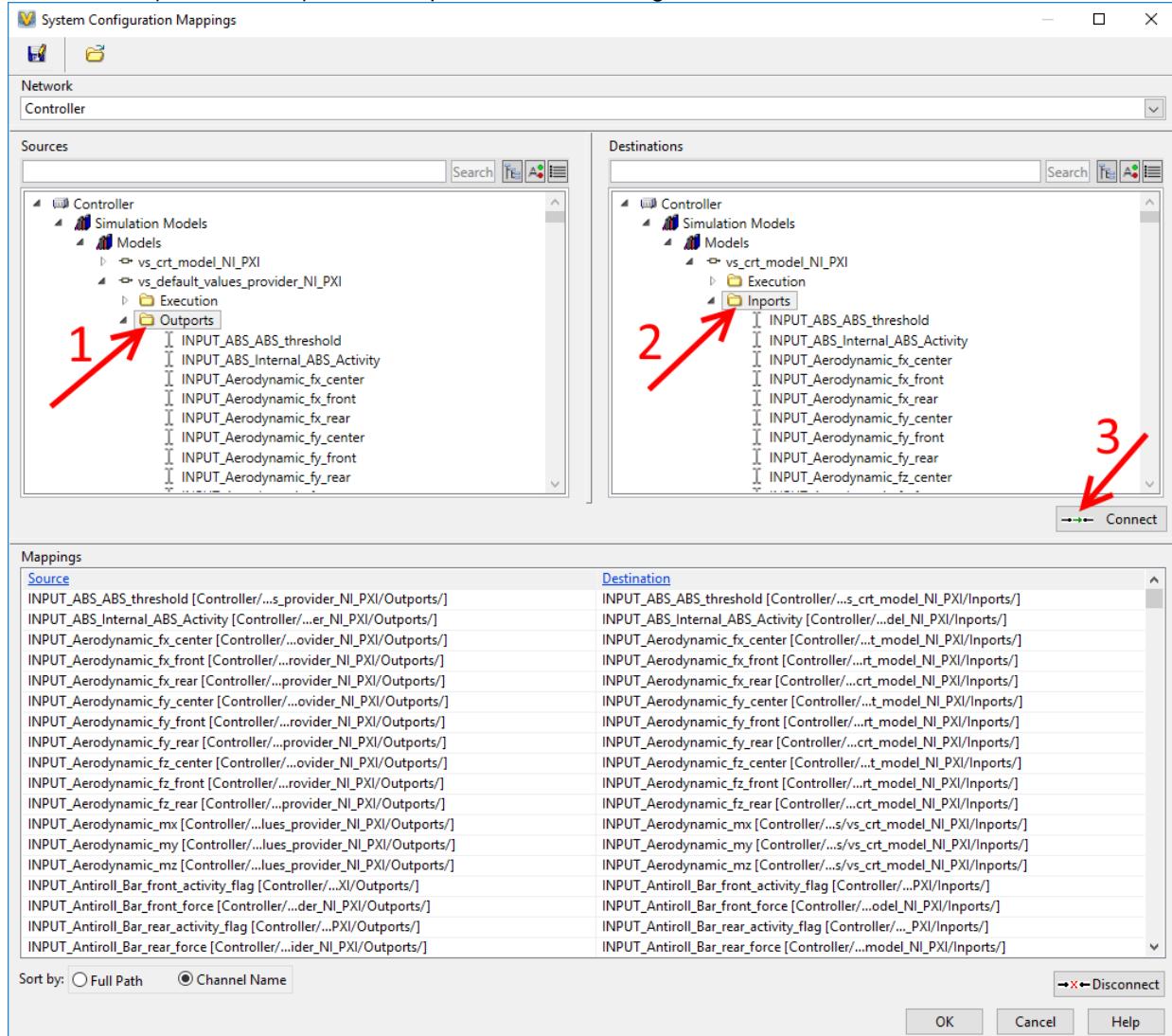
## VI-CarRealTime HIL Overlays



You will end up with 2 models added to your project. Those models input and output need to be connected. Therefore, open the *System Configuration Mappings* by clicking on the toolbar button highlighted in the following:

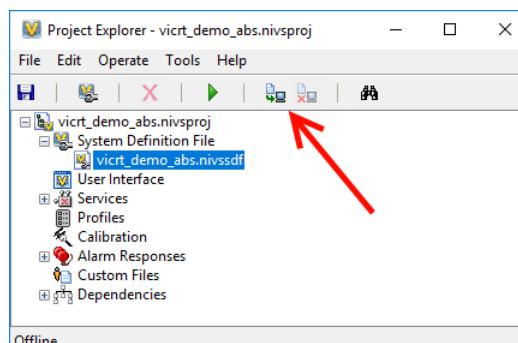


Then, expand the tree of the *Sources* and the tree of the *Destinations* so that the *Outports* of the model `vs_default_values_provider_NI_PXI` are selected on the *Sources* and the *Inports* of the model `vs_crt_model_NI_PXI` are selected on the *Destinations*. You can then click on the button *Connect* to map the selected *Inports* and *Outports* as depicted in the following.



Note: Please, notice that these initial mappings are only for the purpose of the tutorial. In a typical hardware-in-the-loop simulation, you will start from this mappings and then rewire a subset of the *Inports* and *Outports* to your specific models or acquisition peripherals.

You can now save and close the *System Configuration Mappings* and *System Definition File*: your NI-Veristand project is ready to be deployed and run on the target NI-PXI machine.



## 3.3 VI-CarRealTime Simulink Real-Time Overlay

### 3.3.1 Introduction

The VI-CarRealTime Simulink Real-Time Overlay is an additional tool that allows the execution of the VI-CarRealTime simulations on the Simulink Real-Time target environment.

After the installation, it is possible to build a Simulink Real-Time target application using the VI-CarRealTime Matlab/Simulink interface and the Simulink Real-Time target code generation options.

**Note:** The VI-CarRealTime solver library distributed with VI-CarRealTime Simulink Real-Time Overlay is generated using **Microsoft Visual C++ 2013 win32** compiler.

Please configure Simulink Real-Time to use the same compiler with the *mex -setup* command before the first build.

[Installation](#)

[Use the VI-CarRealTime Simulink Real-Time Overlay](#)

### 3.3.2 Installation

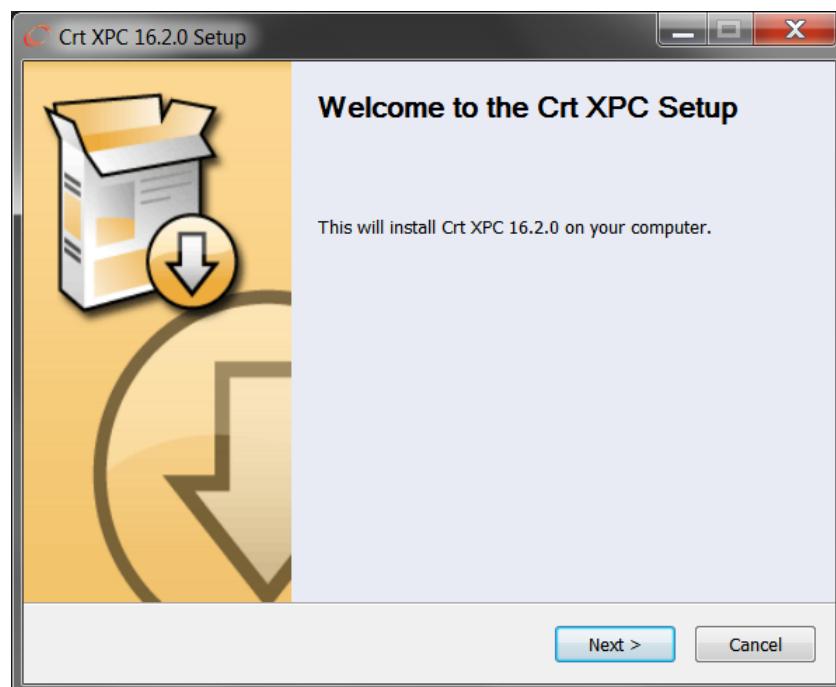
Welcome to the installation guide for VI-CarRealTime Simulink Real-Time Overlay. VI-CarRealTime Simulink Real-Time Overlay is currently for windows only.

In order to perform a complete installation of VI-CarRealTime Simulink Real-Time Overlay, you need to pickup the following files from the VI-grade [website support](#) area (registration is required):

- **VI\_CarRealTime\_xpc\_overlay\_20\_0\_x\_Setup.exe**: main product installer;
- **VI\_CarRealTime\_xpc\_overlay\_20\_0\_Installing.pdf**: this document;
- **VI\_CarRealTime\_xpc\_overlay\_20\_0\_Release\_Notes.pdf**: product information.

After reviewing the Release Notes document, the installation can start following the steps below:

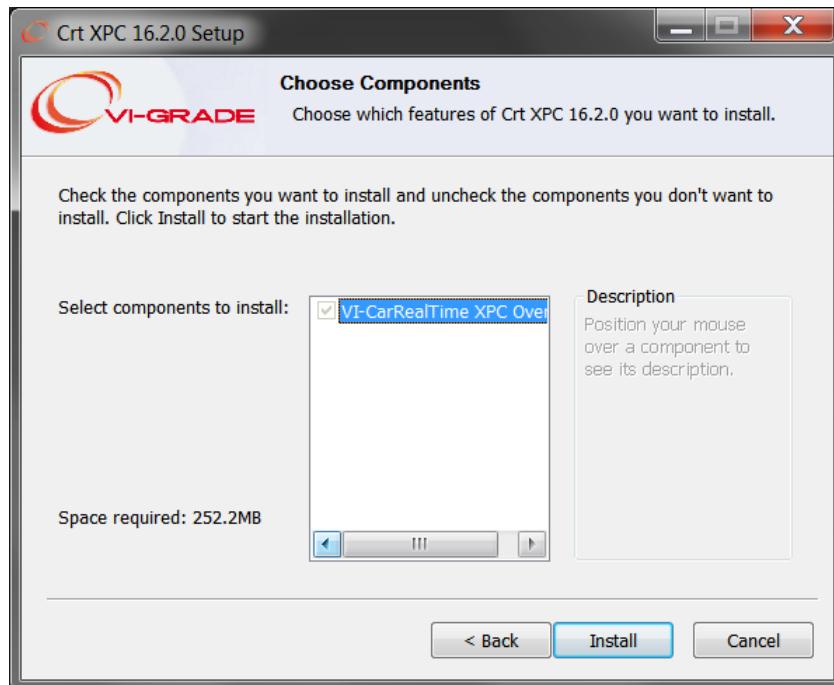
1. Double click on the **VI\_CarRealTime\_xpc\_overlay\_20\_0\_x\_Setup.exe** file, and follow the instructions that will appear on the screen:



The following component will be installed:

- **VI-CarRealTime Simulink Real-Time Overlay**

a specific VI-CarRealTime Simulink Real-Time library and other support files. Older VI-CarRealTime Simulink Real-Time Overlay 20 installation, already available on the target machine will be updated.



At the end of the installation the following window will appear:



Click "Finish" to complete the installation.

This procedure installs the Simulink Real-Time directory inside the VI-CarRealTime installation directory (or inside the one selected during the installation phase).

### 3.3.3 Limitations

The following table shows the features that are not currently supported by VI-CarRealTime Simulink Real-Time Overlay.

|           |                                                                              |
|-----------|------------------------------------------------------------------------------|
| Modelling | Advanced Steering<br>Trailer<br>External FMUs<br>FTire<br>MF-Tyre / MF-Swift |
| Events    | Max Performance<br>Press Maneuvers<br>SpeedGen<br>Sevenpostrig               |

Model extensions such as custom tires and custom aerodynamic require specific builds of the user code for Simulink Real-Time target.

### 3.3.4 Getting Started

This tutorial shows how to build a VI-CarRealTime application to run on the Simulink Real-Time target.

The main steps are:

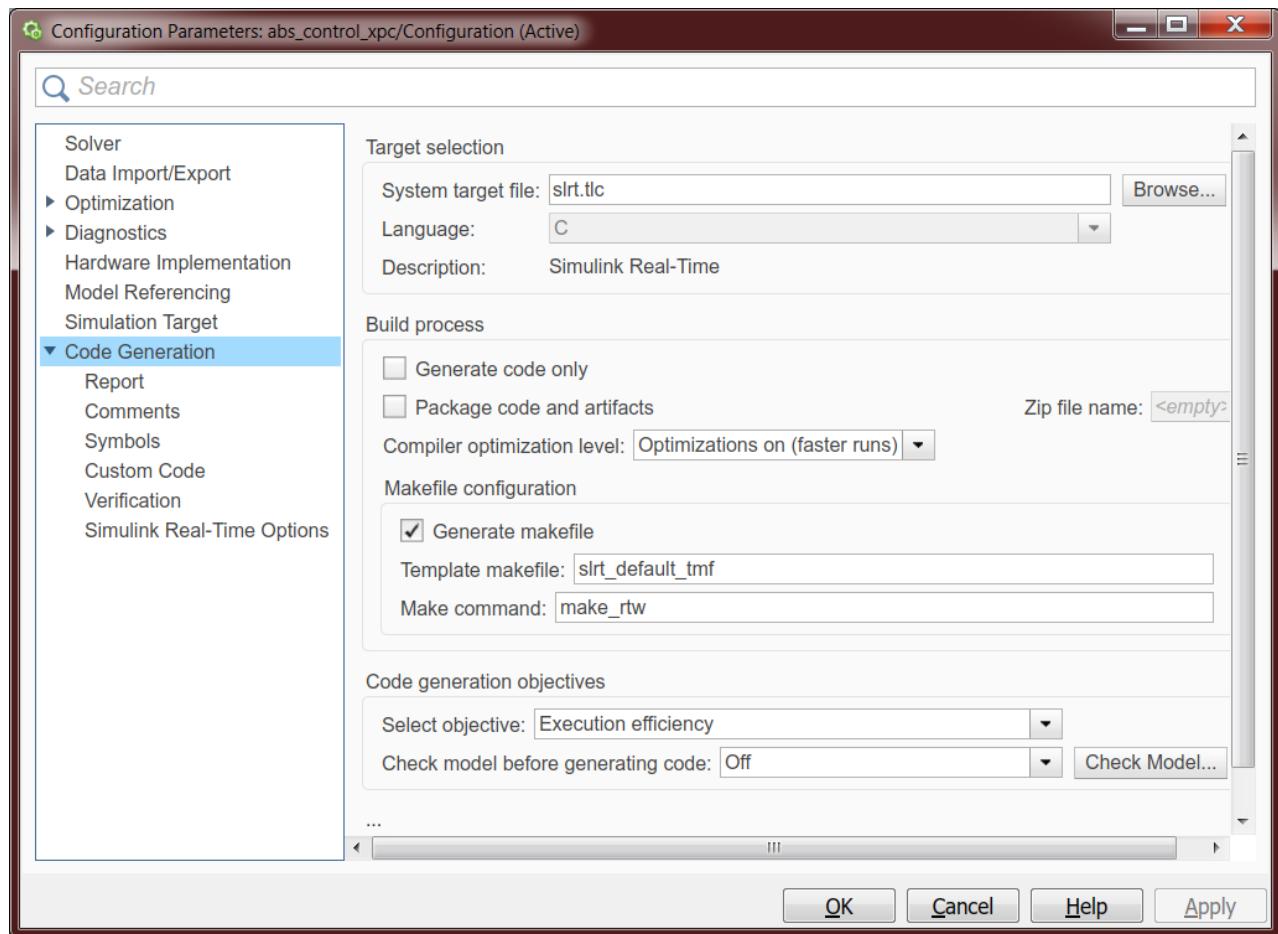
- [Set up the Simulink model](#) in order to use the Simulink Real-Time target tool chain.
- [Build and/or load the application](#).

#### Set Up Simulink Model

In order to build a Simulink Real-Time target application, the Simulink model must be configured to use the Simulink Real-Time target tool chain instead of the standard Generic Real -Time Target one.

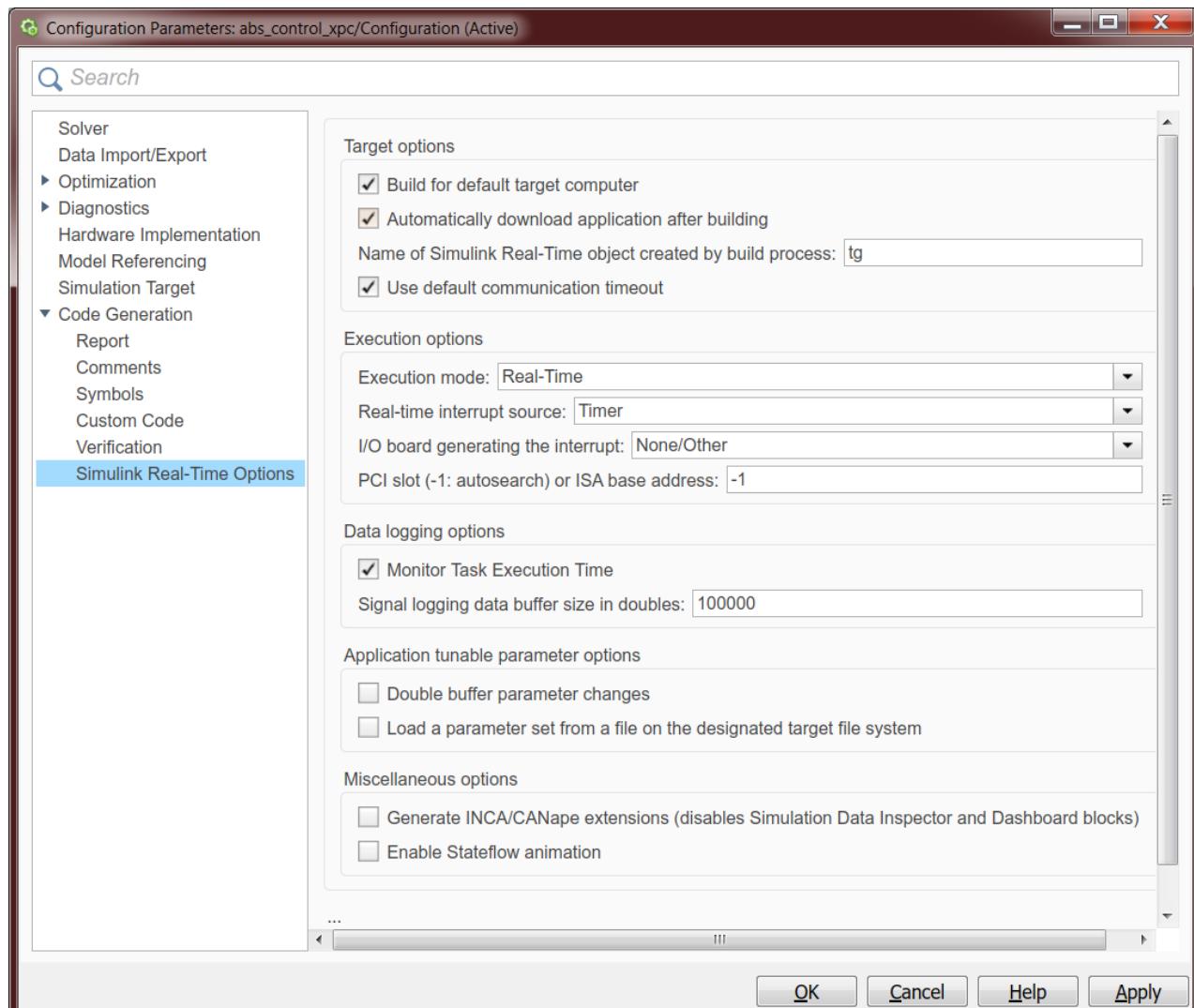
This can be done by going under the **Code Generation** section of the **Model Configuration Parameters** window placed under the **Simulation** menu (**Ctrl+E** shortcut).

In this panel set the **System target file** to **srt.tlc**.



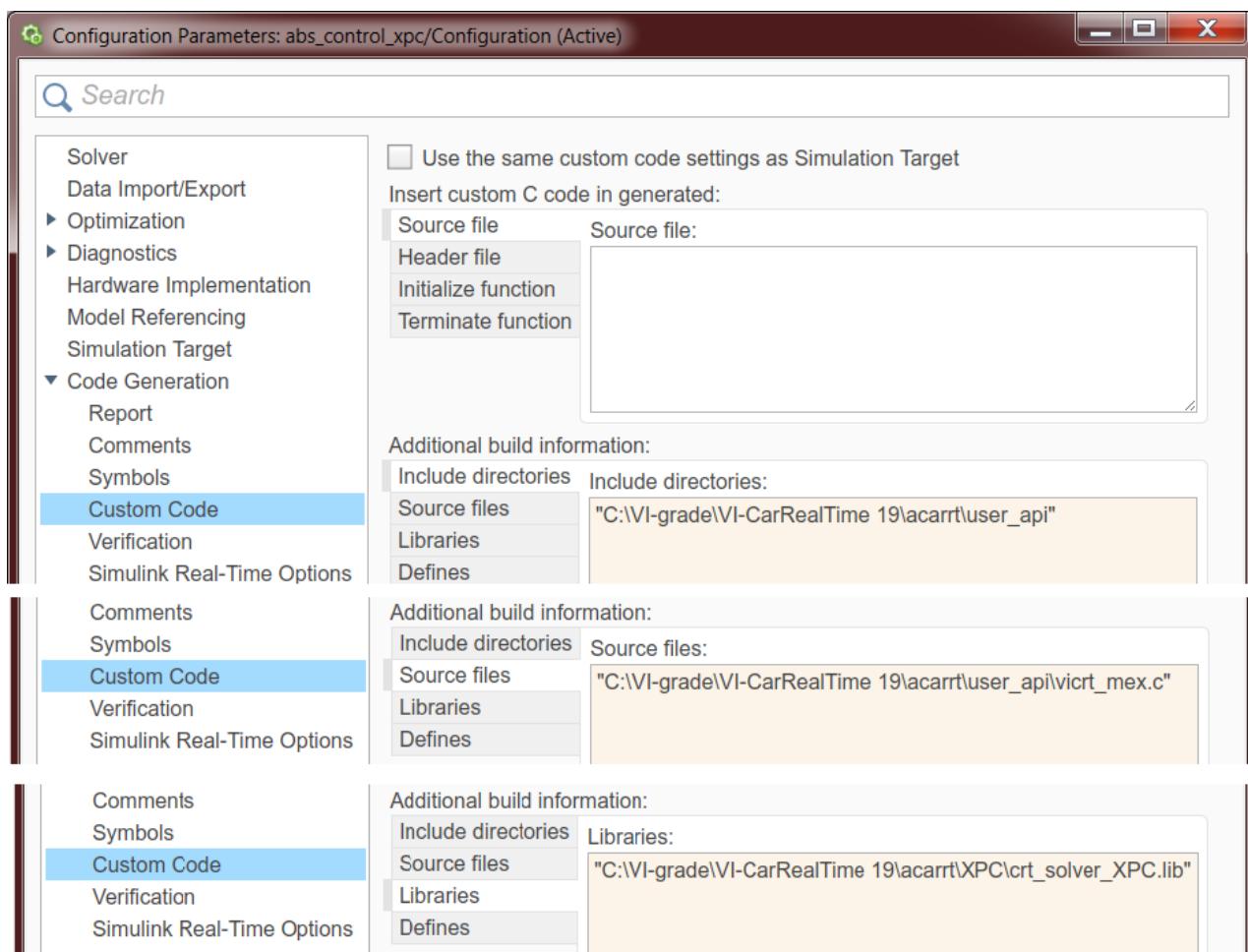
Then under the **Simulink Real-Time Target options** check that *Execution mode* is *Real-Time* and choose if you want to automatically download the application to the target computer after the build (for more details refer to the Simulink Real-Time documentation).

## VI-CarRealTime HIL Overlays



Finally, under the **Custom Code** pane set:

- the **Include directories** field to "`$ VI-CarRealTime install dir$\acarrt\user_api`"
- the **Source files** field to "`$ VI-CarRealTime install dir$\user_api\acarrt\vicrt_mex.c`"
- the **Libraries** field "`$ VI-CarRealTime install dir$\acarrt\XPC\crt_solver_XPC.lib`"



**Note:** In order to avoid system path resolution error, remember to enclose path string between ".

## Build Simulink Real-Time Application

After configuring the building environment needs, the Simulink model can be built using the **Build** button placed in the **Code Generation** pane of the **Configuration Parameters** window (or pressing the **Build Model** icon on the toolbar, or using the **Ctrl+B** shortcut).

The compile procedure reports on the Matlab Diagnostic Viewer the compile commands executed.

## VI-CarRealTime HIL Overlays

```

Diagnostic Viewer
abs_control_xpc

DERT_CORE C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\src\xpctarget.c
xpctarget.c
Compiling appmappingData.c
 cl -D_MT -MT /wd4996 /fp:fast /arch:SSE2 /W3 /Z7 /c /nologo /O2 /Oy- -DMODEL=abs_control_xpc -DRT -
DNUMST=2 -DTID01EQ=1 -DNCSTATES=5 -DMT=0 -DHAVESTDIO -DXPCMSVISUALC -DXPCCALLCONV=__cdecl -DUSE_RTMODEL -
DERT_CORE appmappingData.c
appmappingData.c
Compiling C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\src\legacy_stdio.c
 cl -D_MT -MT /wd4996 /fp:fast /arch:SSE2 /W3 /Z7 /c /nologo /O2 /Oy- -DMODEL=abs_control_xpc -DRT -
DNUMST=2 -DTID01EQ=1 -DNCSTATES=5 -DMT=0 -DHAVESTDIO -DXPCMSVISUALC -DXPCCALLCONV=__cdecl -DUSE_RTMODEL -
DERT_CORE C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\src\legacy_stdio.c
legacy_stdio.c
Compiling C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\xpcblocks\include\xpcimports.c
 cl -D_MT -MT /wd4996 /fp:fast /arch:SSE2 /W3 /Z7 /c /nologo /O2 /Oy- -DMODEL=abs_control_xpc -DRT -
DNUMST=2 -DTID01EQ=1 -DNCSTATES=5 -DMT=0 -DHAVESTDIO -DXPCMSVISUALC -DXPCCALLCONV=__cdecl -DUSE_RTMODEL -
DERT_CORE C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\xpcblocks\include\xpcimports.c
xpcimports.c
Compiling
C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\xpcblocks\include\xpcPCFunctions.c
 cl -D_MT -MT /wd4996 /fp:fast /arch:SSE2 /W3 /Z7 /c /nologo /O2 /Oy- -DMODEL=abs_control_xpc -DRT -
DNUMST=2 -DTID01EQ=1 -DNCSTATES=5 -DMT=0 -DHAVESTDIO -DXPCMSVISUALC -DXPCCALLCONV=__cdecl -DUSE_RTMODEL -
DERT_CORE C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\xpcblocks\include\xpcPCFunctions.c
xpcPCFunctions.c
 lib /nologo /out:xpcruntime.lib xpcimports.obj xpcPCFunctions.obj
Linking ...
 C:\PROGRA~1\MATLAB\R2017b\bin\win64\createResponseFile.exe 1 abs_control_xpc.lnk abs_control_xpc.obj
rt_matrix.obj rt_printf.obj rt_logging.obj abs_control_xpc_capi.obj abs_control_xpc_data.obj
abs_control_xpc_xcp.obj abs_control_xpc_xcp_TET.obj rtGetInf.obj rtGetNaN.obj rt_nonfinite.obj
xpc_datatype_ground.obj rt_logging_mmi.obj rtw_modelmap_utils.obj vicrt_mex.obj host_timer_x86.obj
xpc_code_profiling_utility_functions.obj xpctarget.obj appmappingData.obj legacy_stdio.obj
 link /NOLOGO /DLL /SUBSYSTEM:CONSOLE /DEF:xpcvcdll.def /Include:_malloc /MAP /DEBUG /IGNORE:4099
C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\lib\libatlas.a
C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\target\build\lib\libblas.a xpcruntime.lib
W:\software\intermediate\XPC\12.0\release\2005r2\vicrt\hil\crt_solver_XPC.lib @abs_control_xpc.lnk
@abs_control_xpc_ref.rsp -out:abs_control_xpc_slrt.dll
 Creating library abs_control_xpc_slrt.lib and object abs_control_xpc_slrt.exp
Created DLL abs_control_xpc_slrt.dll
 C:\PROGRA~1\MATLAB\R2017b\toolbox\rtw\targets\xpc\xpc\bin\mkusrdlm -c+ -q+ abs_control_xpc_slrt.dll
..\..\abs_control_xpc
RTTarget-32 DLM Processor 6.05 (c) 1996,2016 On Time Informatik GmbH
Created DLM ..\abs_control_xpc.dlm
Successful completion of build procedure for model: abs_control_xpc
Created MLDATX ..\abs_control_xpc.mldatx
Download process is disabled.

Build process completed successfully

```

At the end of the build if no error occurs the target *.mldatx* application is generated and automatically loaded to the target computer if the related option has been selected.

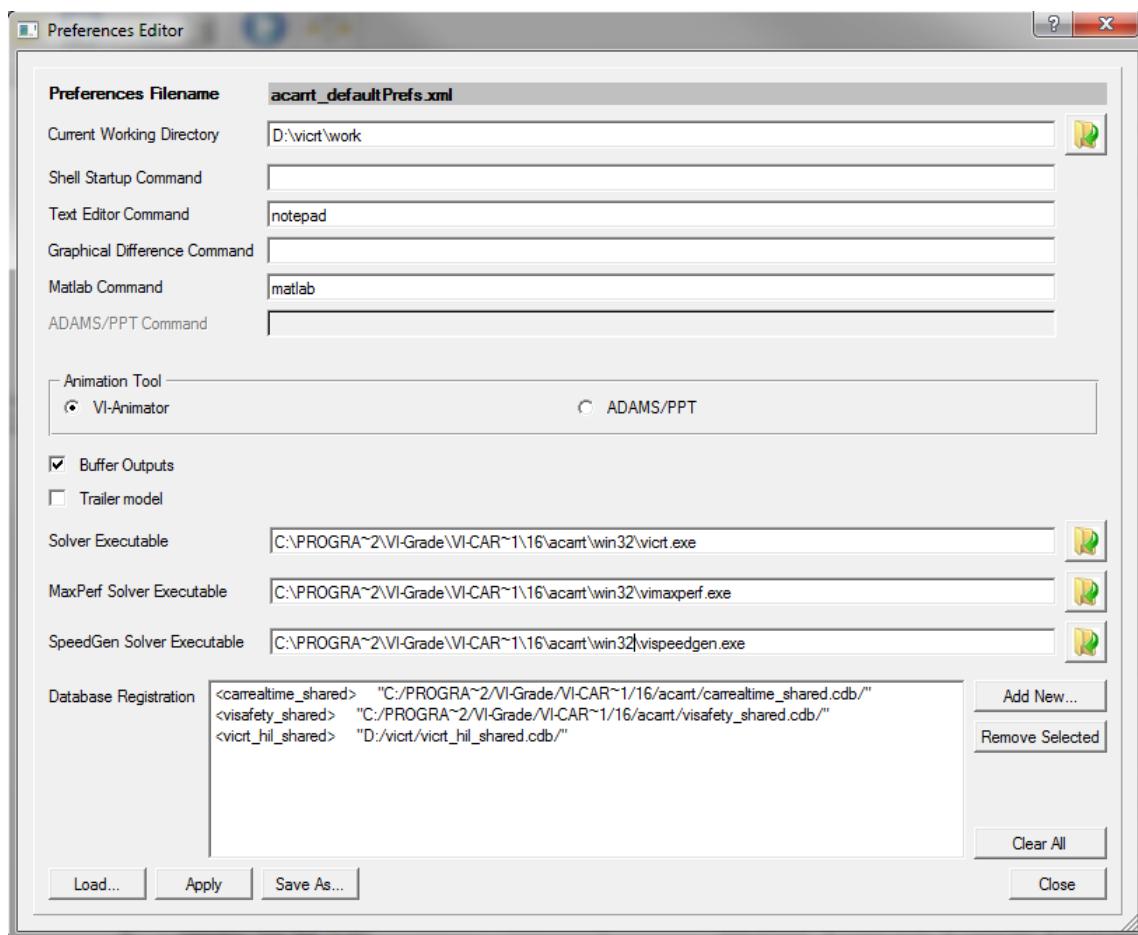
### 3.3.5 Tutorial

#### ABS Control

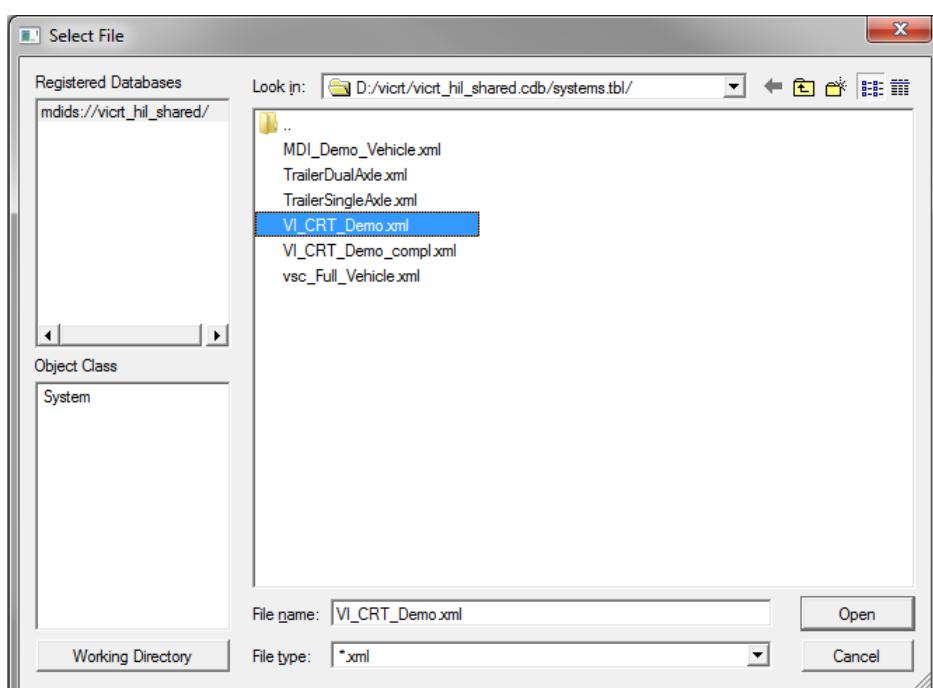
This tutorial shows how to run a standard VI-CarRealTime Simulink model on the Simulink Real-Time target computer. The Simulink model is a co-simulation between VI-CarRealTime and an external ABS control system.

The first step is about the generation of the input solver file including model and event data. Create a folder named **vicrt** under the root of your hard drive on the host machine and unzip inside it the **vicrt\_hil\_shared.cdb.zip** database which is under the directory "**\$VI-CarRealTime install dir\$\examples\XPC**". Then create a **work** folder at the same level and start from there a new VI-CarRealTime session using the **vicrt20** command from a dos shell or by using the the shortcut from Windows Start Menu.

Select **Edit** and **Preferences...** in the main toolbar to open the **Preferences Editor** dialog and register as VI-CarRealTime database the one placed under "**\$hard drive root\$\vicrt\vicrt\_hil\_shared.cdb**" (the existing databases can be unregistered).



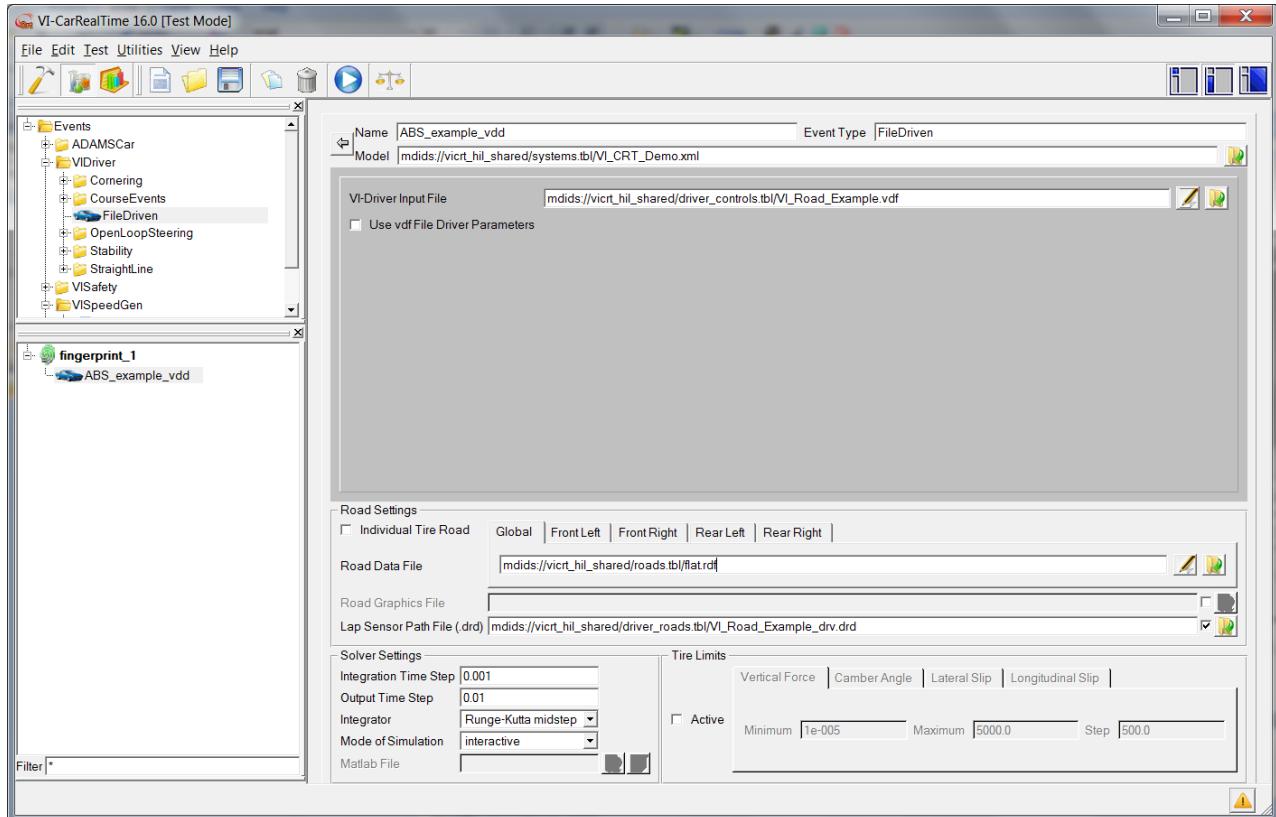
In **Build Mode** open the model **VI\_CRT\_Demo.xml** from the **vicrt\_hil\_shared** database.



## VI-CarRealTime HIL Overlays



Now you can switch to the **Test Mode** ( ) where the event specification can be defined. Create a new fingerprint, add a new VI-Driver event and rename it as *ABS\_example\_vdd* .



Please select from the `vicrt_hil_shared` database the `VI_Road_Example.vdf` event file. Choose the `VI_Road_example.rdf` as the road data file to be used. In order to activate the `lap_sensor`, needed by the Simulink model, then select from the `vicrt_hil_shared` shared database the `VI_Road_Example_drv.drd` file in the field **Lap Sensor Path File**.

**Note:** make sure that the internal ABS model is deactivated in `Properties > ABS > built_in_abs_active` and the circular buffer option is active in `Properties > system_properties > circular_buffer` (see circular buffer).



Now set the simulation mode to "**files only**" and run the event ( ) in order to generate the set of files needed by the VI-CarRealTime Matlab/Simulink interface.

Start a Matlab session from the current working directory on the host machine. The following step is to generate the same folders and database on the target machine. So create `vicrt` folder and `work` subfolder under the root of the target machine interactively with `slrteexplr` or manually with the proper Simulink Real-Time commands. It is available a `xpc_database_copy.m` script in `$VI-CarRealTime install dir$\examples\XPC` to copy the whole database to the target computer:

```
xpc_database_copy('C:\vicrt\vicrt_hil_shared.cdb')
```

will copy the database at the specified path on the host machine to the current target machine path.

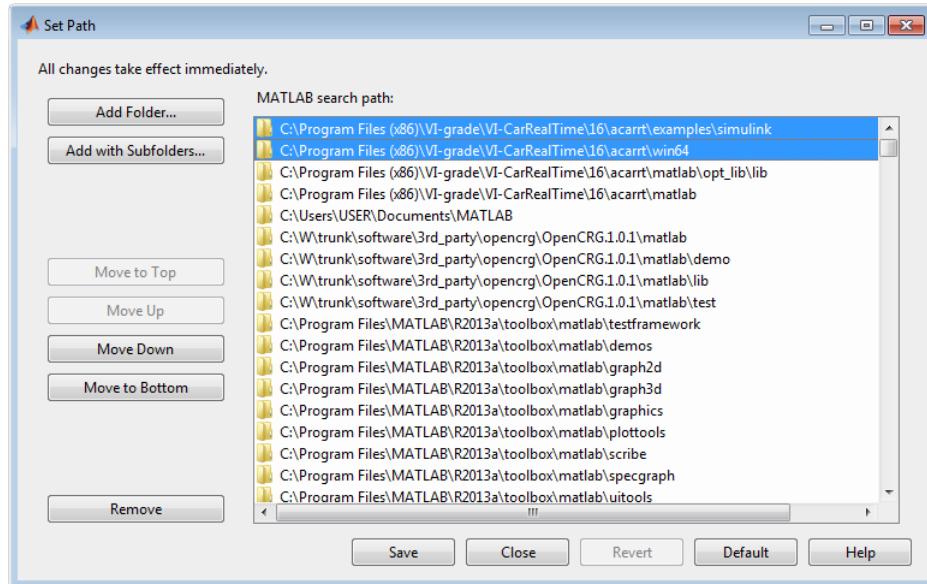
**Note:** it is preferable to use Simulink Real-Time target commands when working with the target file system instead of using `slrteexplr`.

Then upload also the `vicrt_cdb.cfg` file and the `ABS_example_vdd_send_svm.xml` event file to the `C:/vicrt/work` folder of the Simulink Real-Time target machine.

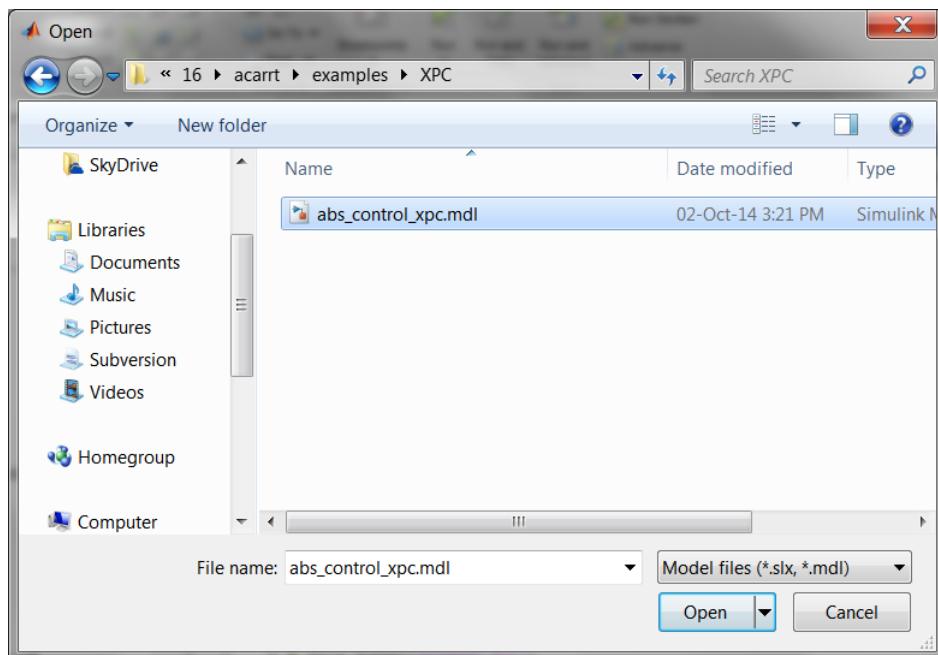
On the Simulink Real-Time target machine change the hard drive root in the Path of Database field of the vicrt\_cdb.cfg if necessary:

```
!-----!
! ***** VI-CarRealTime Database Configuration File *****
!-----!
! Database name Path of Database
!-----!
DATABASE vicrt_hil_shared C:/vicrt/vicrt_hil_shared.cdb
```

If you are opening a Matlab session for the first time, add to Matlab search paths the VI-CarRealTime paths by using the "addpath\_vicrt\_20" script.



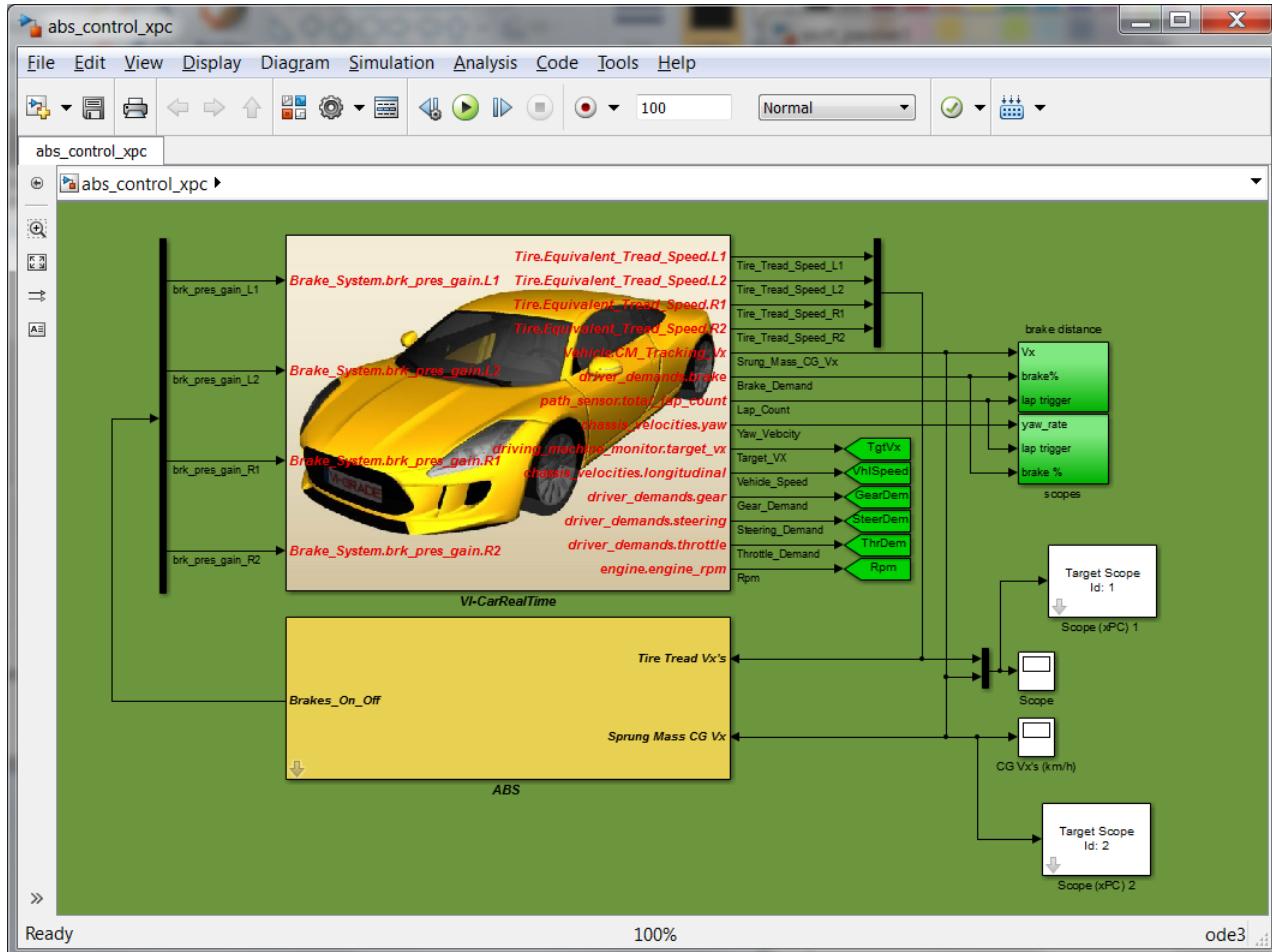
From your Matlab session open the file abs\_control\_xpc.mdl located in the sub folder examples\XPC of VI-CarRealTime installation folder.



## VI-CarRealTime HIL Overlays

This ABS control model contains two Simulink subsystems:

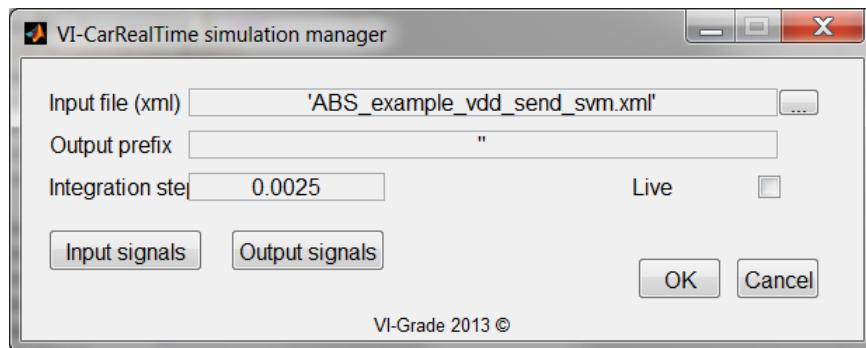
- VI-CarRealTime: the solver Simulink interface;
- ABS Controller: the ABS controller implementation.



Now set the VI-CarRealTime solver input file using the `vicrt_inputfile` Matlab variable and the `time_step` variable for the integration step.

```
Command Window
>>
>> vicrt_inputfile = 'ABS_example_vdd_send_svm.xml';
>> time_step = 0.0025;
>>
fx >>
```

or alternatively use the VI-CarRealTime Solver mask **Input file (xml)** and **Integration Step** fields.



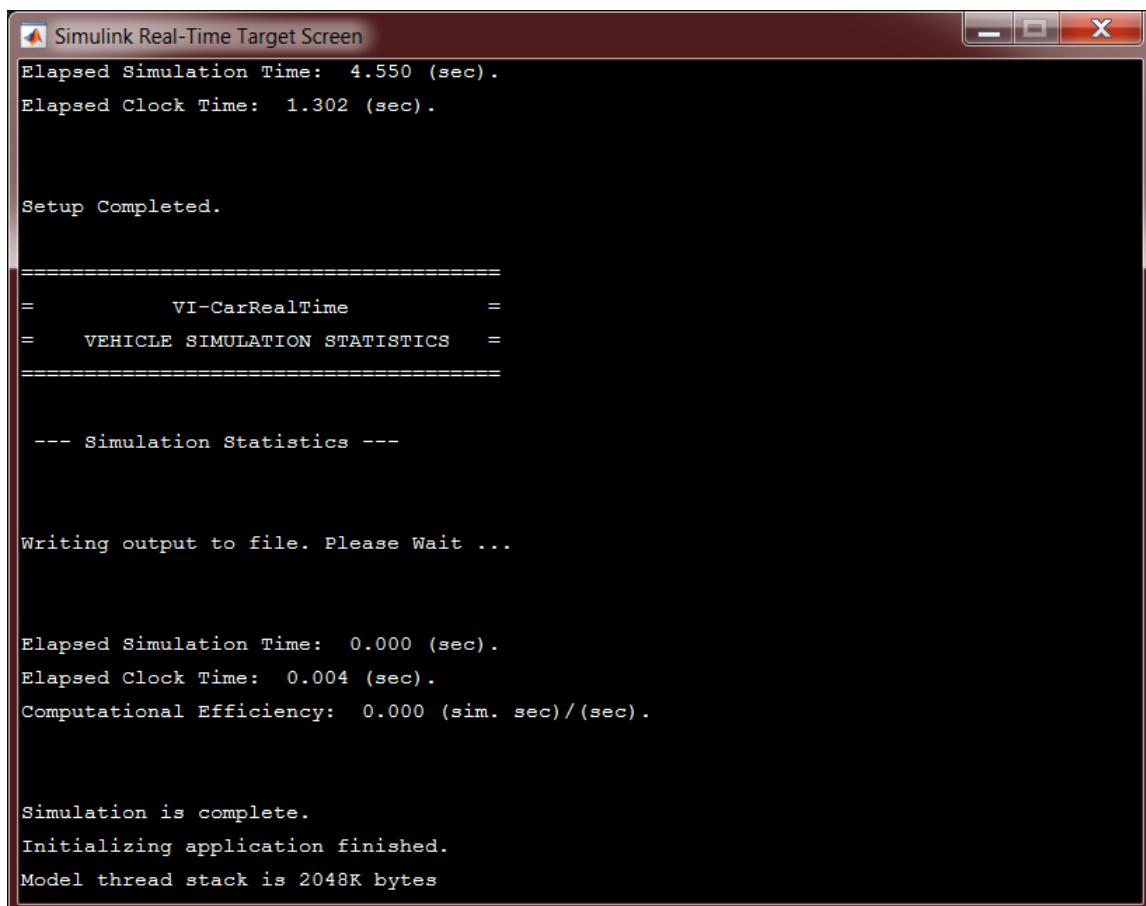
The Simulink model example is already configured to generate Simulink Real-Time target applications but you may need to change the paths for the custom source code and library according to the location used for the VI-CarRealTime Simulink Real-Time Overlay installation. See [set up simulink model](#) for these *Custom Code* settings or for the general Simulink Real-Time target options if needed.

The model is now ready to be built for the Simulink Real-Time target machine so you can start the build process ([see build Simulink Real-Time application](#)).

Typing `slrtepxlr` in the Matlab Command Window you can open the Simulink Real-Time interface and communicate with the target machine.

Alternatively the following commands can be executed:

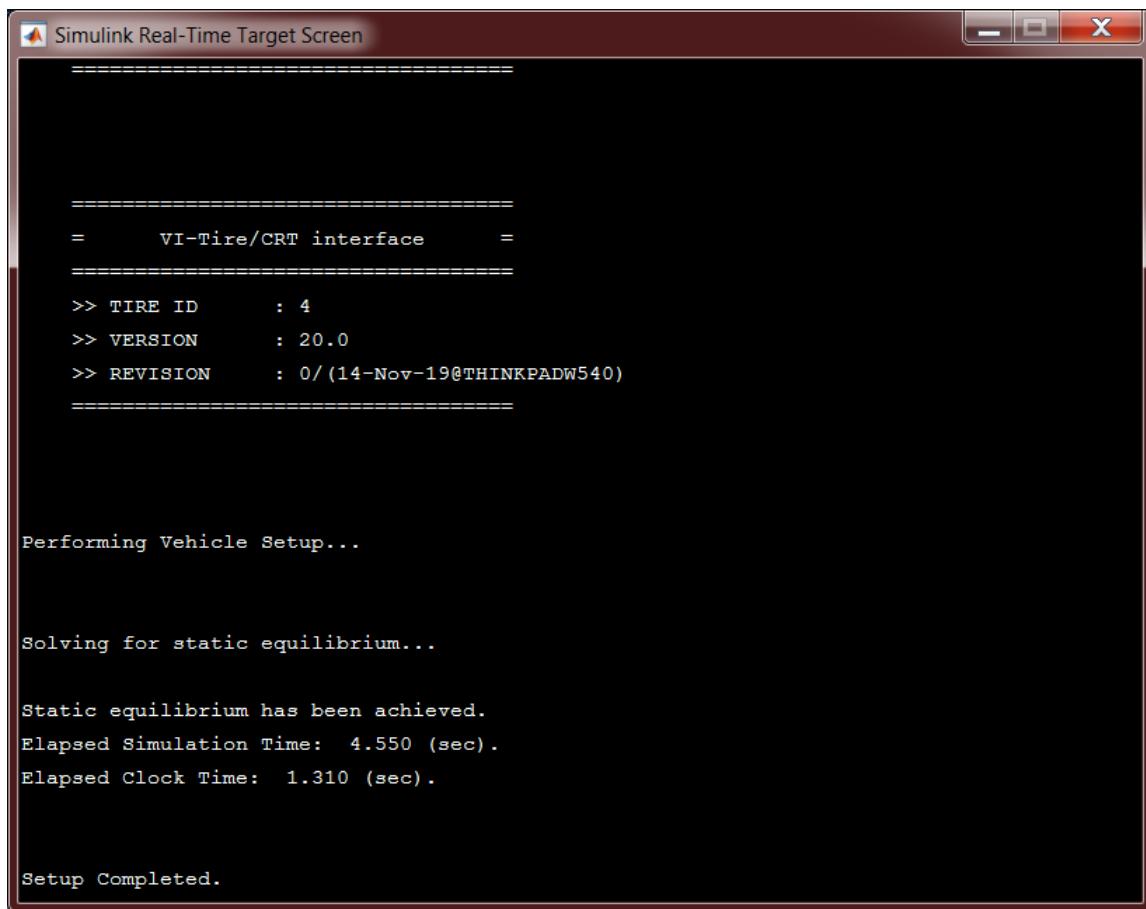
- create a target object: `tg = slrt` (IP address and port number are retrieved from the target machine)
- load the application: `tg.load('abs_control_xpc')` and wait for the model initialization



- move to the working directory: `fsys = SimulinkRealTime.fileSystem; fsys.cd('vicrt');`  
`fsys.cd('work');`

## VI-CarRealTime HIL Overlays

- start the simulation: tg.start



The screenshot shows a terminal window titled "Simulink Real-Time Target Screen". The output text is as follows:

```
=====
= VI-Tire/CRT interface =
=====
>> TIRE ID : 4
>> VERSION : 20.0
>> REVISION : 0/(14-Nov-19@THINKPADW540)
=====

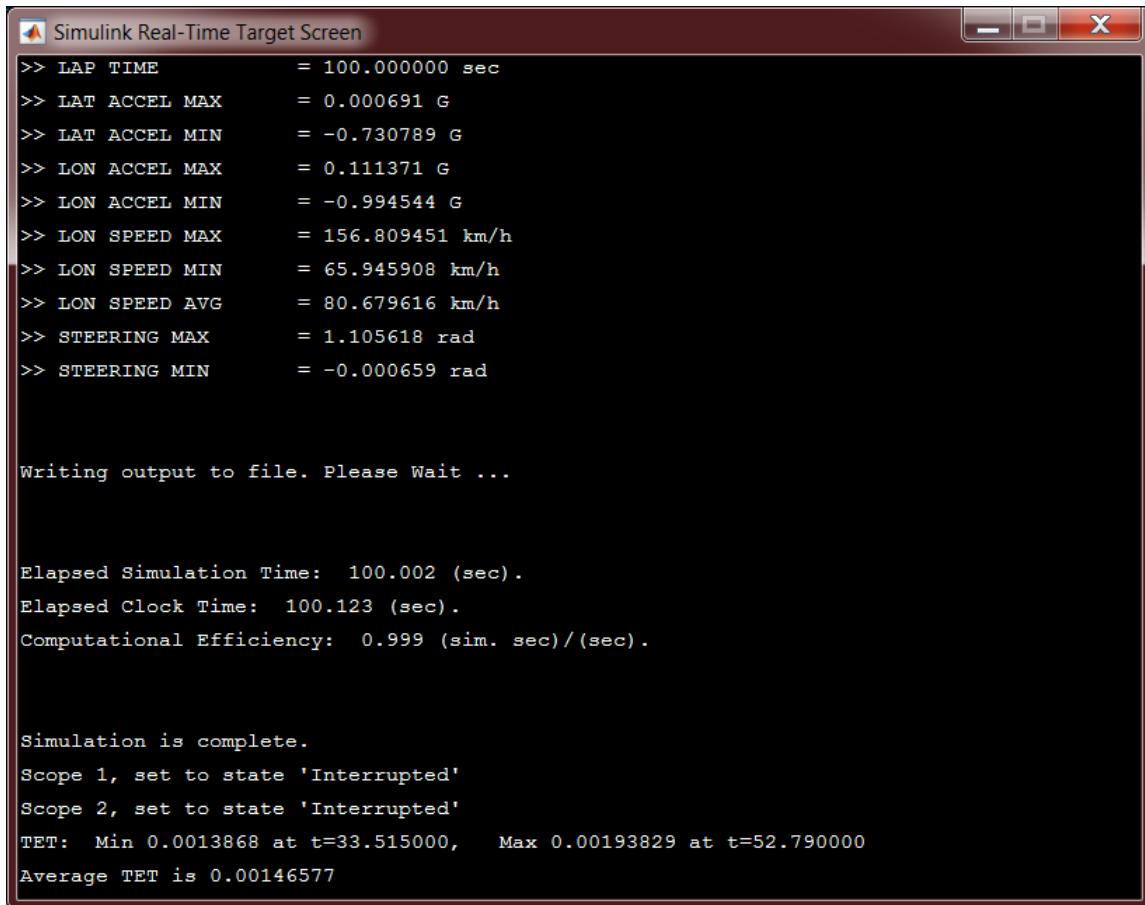
Performing Vehicle Setup...

Solving for static equilibrium...

Static equilibrium has been achieved.
Elapsed Simulation Time: 4.550 (sec).
Elapsed Clock Time: 1.310 (sec).

Setup Completed.
```

When the simulation is running you can see the plots update in the Target Scopes already defined into the model (tires and chassis velocities).



The screenshot shows a MATLAB command window titled "Simulink Real-Time Target Screen". It displays various simulation parameters and statistics:

```
>> LAP TIME = 100.000000 sec
>> LAT ACCEL MAX = 0.000691 G
>> LAT ACCEL MIN = -0.730789 G
>> LON ACCEL MAX = 0.111371 G
>> LON ACCEL MIN = -0.994544 G
>> LON SPEED MAX = 156.809451 km/h
>> LON SPEED MIN = 65.945908 km/h
>> LON SPEED AVG = 80.679616 km/h
>> STEERING MAX = 1.105618 rad
>> STEERING MIN = -0.000659 rad

Writing output to file. Please Wait ...

Elapsed Simulation Time: 100.002 (sec).
Elapsed Clock Time: 100.123 (sec).
Computational Efficiency: 0.999 (sim. sec) / (sec).

Simulation is complete.
Scope 1, set to state 'Interrupted'
Scope 2, set to state 'Interrupted'
TET: Min 0.0013868 at t=33.515000, Max 0.00193829 at t=52.790000
Average TET is 0.00146577
```

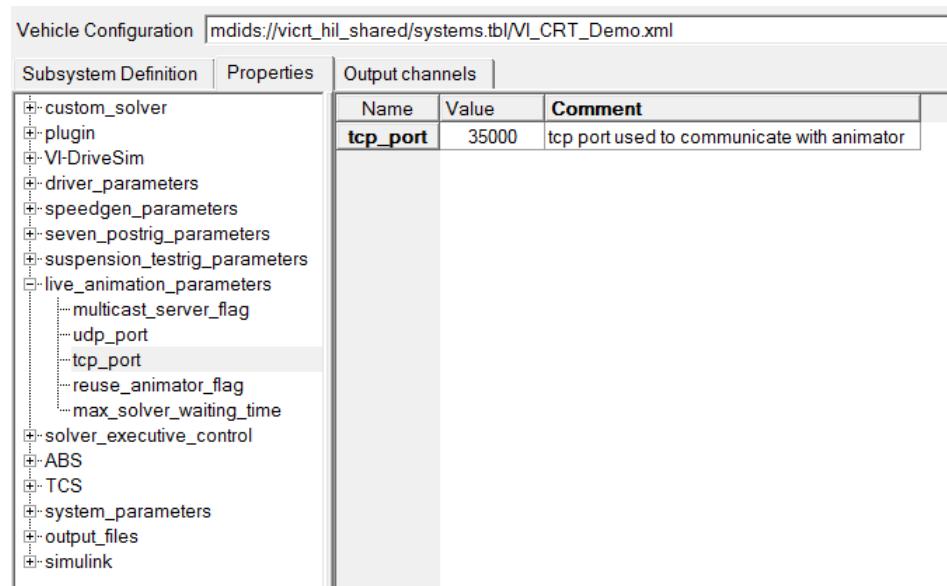
At the end of the simulation the .res results file (only the last seconds stored in the circular buffer) is saved in the target working directory and ready to be downloaded to the host machine.

## Live animation

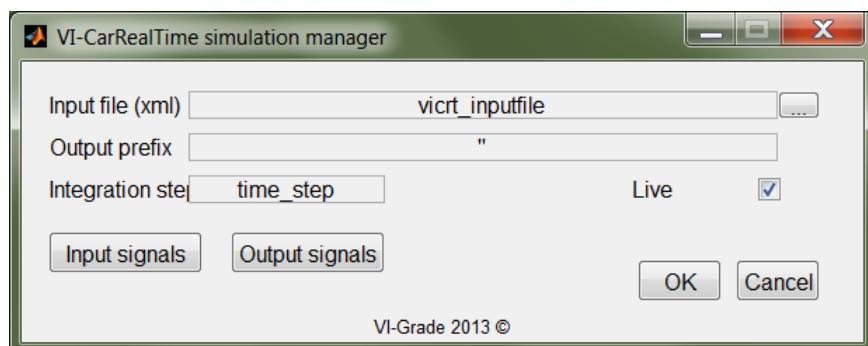
Some steps are required in order to set up a communication between VI-Animator and the Simulink Real-Time target machine.

- Before to generate the `ABS_example_vdd_send_svm.xml` input file from VI-CarRealTime, set the tcp port number that will be used for the communication in `Properties > live_animation_parameters > tcp_port`

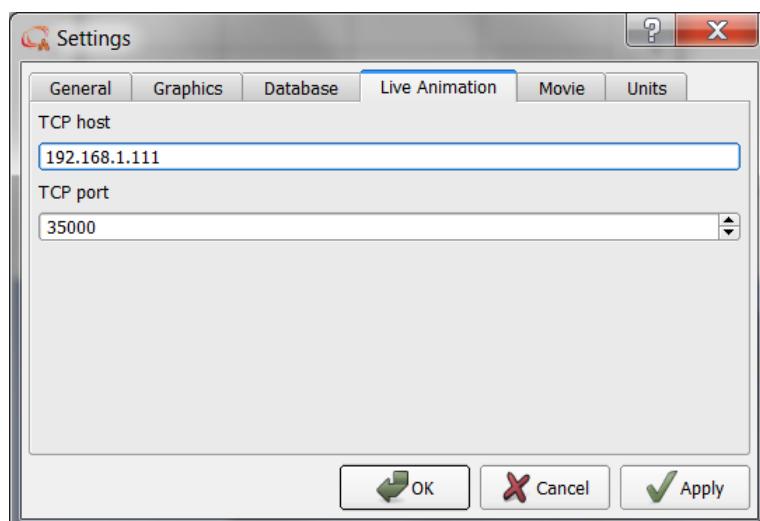
## VI-CarRealTime HIL Overlays



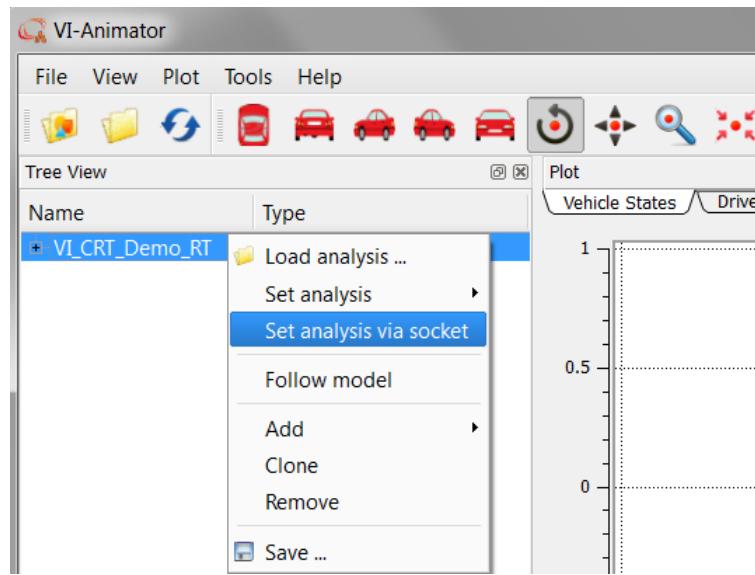
- Before to build the Simulink Real-Time target application in Simulink, click on the *Live* check box to activate the live animation.



- Launch VI-Animator and change the live animation settings in *Tools > Settings > Live Animation* tab. Write the Simulink Real-Time target machine TCP address in *TCP host* and the port number previously defined in *TCP port*.



- Load the vehicle graphics in VI-Animator and when the Simulink Real-Time target simulation is started right-click on the graphics tree item and select *Set analysis via socket*.



Once the communication is established the vehicle graphics will be animated and the plots updated with the data from the Simulink Real-Time target machine while the simulation is running.

## 3.4 VI-CarRealTime ETAS Overlay

### 3.4.1 Introduction

The VI-CarRealTime ETAS Overlay is an additional tool that allows the execution of the VI-CarRealTime model on an ETAS RTPC machine.

After the installation, it is possible to build an ETAS application using the VI-CarRealTime Matlab/Simulink interface and the ETAS LABCAR-OPERATOR.

The VI-CarRealTime ETAS Overlay is provided under a specific board locked license.

### 3.4.2 Installation

Welcome to the installation guide for VI-CarRealTime ETAS Overlay. VI-CarRealTime ETAS Overlay is currently for windows only.

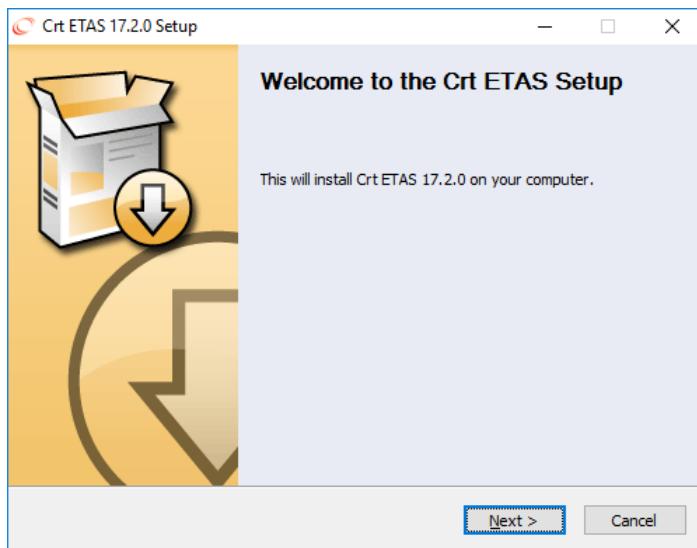
In order to perform a complete installation of VI-CarRealTime ETAS Overlay, you need to pickup the following files from the VI-grade [website support](#) area (registration is required):

- **VI\_Crt\_etas\_20\_0\_x\_Setup.exe**: main product installer;
- **VI\_Crt\_etas\_20\_0\_Installing.pdf**: this document;
- **VI\_Crt\_etas\_20\_0\_Release\_Notes.pdf**: product information.

After reviewing the Release Notes document, the installation can start following the steps below:

1. Double click on the **VI\_CarRealTime\_etas\_overlay\_20\_0\_x\_Setup.exe** file, and follow the instructions that will appear on the screen:

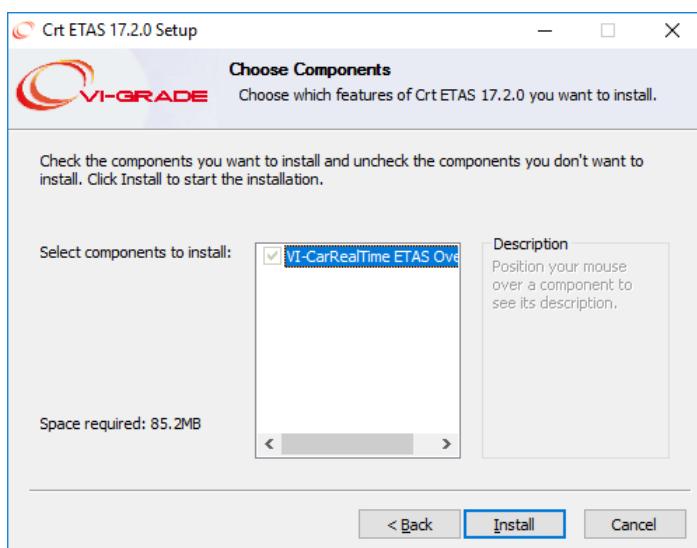
## VI-CarRealTime HIL Overlays



The following component will be installed:

- **VI-CarRealTime ETAS Overlay**

a specific VI-CarRealTime ETAS compliant library and other support files. Older VI-CarRealTime ETAS Overlay 20 installation, already available on the target machine will be updated.



At the end of the installation the following window will appear:



Click "Finish" to complete the installation.

This procedure install the etas directory inside the VI-CarRealTime installation directory (or inside the one selected during the installation phase).

### 3.4.3 Licensing

In order to execute VI-CarRealTime on the ETAS RTPC machine, a board specific license file is needed. The license file can be obtained sending to our [licensing support](#) the MAC address of the ETAS RTPC machine. You can retrieve the MAC address by executing the following steps on the host Windows PC (in the following, we assume that the ETAS RTPC machine is configured with the standard IP address 192.168.40.14):

1. open a command prompt
2. type in ping 192.168.40.14 and hit enter
3. check that the ETAS RTPC is answering
4. type in arp -a and hit enter
5. look for the line corresponding to the IP address 192.168.40.14:

|                                   |                   |         |  |
|-----------------------------------|-------------------|---------|--|
| ...                               |                   |         |  |
| Interface: 192.168.40.240 --- 0x7 |                   |         |  |
| Internet Address                  | Physical Address  | Type    |  |
| 192.168.40.14                     | 08-00-27-3b-b0-43 | dynamic |  |
| 192.168.40.255                    | ff-ff-ff-ff-ff-ff | static  |  |
| 224.0.0.22                        | 01-00-5e-00-00-16 | static  |  |
| 224.0.0.251                       | 01-00-5e-00-00-fb | static  |  |
| 224.0.0.252                       | 01-00-5e-00-00-fc | static  |  |
| 224.0.1.129                       | 01-00-5e-00-01-81 | static  |  |
| 225.1.1.1                         | 01-00-5e-01-01-01 | static  |  |
| 239.255.0.1                       | 01-00-5e-7f-00-01 | static  |  |
| 239.255.255.250                   | 01-00-5e-7f-ff-fa | static  |  |
| 255.255.255.255                   | ff-ff-ff-ff-ff-ff | static  |  |

The 12 hexadecimal code next to the IP address 192.168.40.14 is the MAC address of the ETAS RTPC machine.

As an alternative , you connect through ssh to the ETAS RTPC machine and launch the following command:

```
/sbin/ifconfig
```

This is the output that you should get with the highlighted MAC address.

```
eth0 Link encap:Ethernet HWaddr 08:00:27:3b:b0:43
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
 RX packets:851622 errors:0 dropped:148 overruns:0 frame:0
 TX packets:211315 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:375998198 (358.5 MiB) TX bytes:1618994237 (1.5 GiB)
...
...
```

### 3.4.4 Limitations

The current version of the VI-CarRealTime porting for ETAS does not support the full set of functionalities featured by the Windows version of the software. Please refer to the following table for the unsupported capabilities.

|           |                                                                   |
|-----------|-------------------------------------------------------------------|
| Modelling | Advanced Steering<br>External FMUs<br>FTire<br>MF-Tyre / MF-Swift |
| Events    | Max Performance<br>Press Maneuvers<br>SpeedGen<br>Sevenpostrig    |

Model extensions such as custom tires and aero require specific builds of the user code for the ETAS platform.

### 3.4.5 Tutorial

#### ABS Control

This tutorial shows how to run a standard VI-CarRealTime Simulink model on an ETAS RTPC machine. The Simulink model is a co-simulation between VI-CarRealTime and an external ABS control system.

The tutorial is divided into these steps:

- [Setting up the working directory](#)
- [Generating the maneuver file](#)
- [Creating the LABCAR-OPERATOR project](#)
- [Running the experiment](#)

In this tutorial, we will refer to the directory where VI-CarRealTime is installed with the symbol %CRT\_DIR%.

#### Setting up the working directory

A semi-automatic procedure is provided to prepare the working directory where the LABCAR-OPERATOR project will be created. This procedure is implemented in the batch-file located at the following

```
%CRT_DIR%\acarrt\examples\etas\etas_prepare_project.bat
```

Launch that batch file by opening Windows Explorer in the following directory

```
%CRT_DIR%\acarrt\examples\etas
```

...and double-clicking on `etas_prepare_project.bat`.

The semi-automatic procedure will create the working directory and a *auxiliary* subdirectory named `vigrade` containing the files that VI-CarRealTime needs to correctly run on the target RTPC machine.

During the procedure, your intervention will be required in 2 stages:

1. you will be asked to insert the **path of the working directory**: this must be a non existent directory
2. you will be asked to insert the **path of the VI-grade license file** (see [Licensing](#))

**Example:** in the following, you can see the output of the procedure in the case in which the user has inserted the following:

- **working directory:** `I:\etas_demo`
- **license file:** `C:\Users\vigrade\Desktop\VI-grade_5179_etas.lic`

STEPS:

1. Create working directory for LABCAR-OPERATOR project
2. Create subdirectory for VI-grade auxiliary files
3. Copy VI-grade license file
4. Copy VI-CarRealTime API
5. Copy Simulink Demo Model
6. Copy VI-CarRealTime core static library that will be linked into the generated executable for the RTPC machine
7. Copy batch files for prebuild step of LABCAR-OPERATOR project
8. Copy VI-CarRealTime Simulink components
9. Copy VI-CarRealTime basic binaries
10. Copy VI-CarRealTime MATLAB components
11. Copy VI-CarRealTime shared database
12. Move VI-CarRealTime mex C code
13. Setup .cfg file that will be uploaded to the target RTPC machine
14. Setup .cfg file for the prebuild step of LABCAR-OPERATOR project
15. Summary

CONFIGURATION:

```

CRT_API_LOCATION : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\user_api\
CRT_BIN_DIR : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\win64\
CRT_ETAS_LIB : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\etas\libcrt_solver_etas.a
CRT_ETAS_PREBUILD_BATCH : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\examples\etas\etas_pre_build.bat
CRT_ETAS_PREBUILD_PY : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\examples\etas\etas_pre_build.py
CRT_ETAS_SIMULINK_MODEL : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\examples\Simulink\abs_control_etas.mdl
CRT_MATLAB_DIR : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\matlab\
CRT_ROOT : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..
CRT_SHARED_CDB : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\carrealtime_shared.cdb\
CRT_SIMULINK_DIR : C:\Program Files\VI-grade\VI-CarRealTime
17\acarrt\examples\etas\..\..\..\acarrt\examples\Simulink\
LOG_FILE : E:\tmp\etas_prepare_project.log
PROJECT_NAME : CRT

```

Press Enter to start executing.

1. Create working directory for LABCAR-OPERATOR project

```
Please type in a valid directory path (the directory must be NON existent): I:\etas_demo
Creating directory I:\etas_demo...
```

2. Create subdirectory for VI-grade auxiliary files

```
Creating directory I:\etas_demo\vigrade...
```

3. Copy VI-grade license file

```
Please type in the path of the VI-grade license file associated to the target RTPC
machine: C:\Users\vigrade\Desktop\VI-grade_5179_etas.lic
Copying C:\Users\vigrade\Desktop\VI-grade_5179_etas.lic to file I:\etas_demo\vigrade...
```

4. Copy VI-CarRealTime API

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\user_api\ to I:\etas_demo\vigrade\user_api...
```

5. Copy Simulink Demo Model

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\examples\Simulink\abs_control_etas.mdl to I:\etas_demo\vigrade...
```

6. Copy VI-CarRealTime core static library that will be linked into the generated  
executable for the RTPC machine

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\etas\libcrt_solver_etas.a to I:\etas_demo\vigrade...
```

7. Copy batch files for prebuild step of LABCAR-OPERATOR project

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\examples\Simulink\abs_control_etas.mdl to I:\etas_demo\vigrade...
```

8. Copy VI-CarRealTime Simulink components

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\examples\Simulink\ to I:\etas_demo\vigrade\acarrt\examples\Simulink...
```

9. Copy VI-CarRealTime basic binaries

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\...\...\
\acarrt\win64\ to I:\etas_demo\vigrade\acarrt\win64...
```

10. Copy VI-CarRealTime MATLAB components

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\\...\...
\acarrt\matlab\ to I:\etas_demo\vigrade\acarrt\matlab...
```

11. Copy VI-CarRealTime shared database

```
Copying C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\...\\...\...
\acarrt\carrealtime_shared.cdb\ to I:\etas_demo\vigrade\carrealtime_shared.cdb...
```

12. Move VI-CarRealTime mex C code

```
Moving I:\etas_demo\vigrade\user_api\vicrt_mex.c to I:\etas_demo\vigrade\acarrt\win64...
```

13. Setup .cfg file that will be uploaded to the target RTPC machine

14. Setup .cfg file for the prebuild step of LABCAR-OPERATOR project

15. Summary

The procedure for creating the VI-CarRealTime demo project has completed.

- the **working directory** has been created in **I:\etas\_demo**
- the **auxiliary directory** has been created in **I:\etas\_demo\vigrade**

Please, press Enter to exit.

As you can see, at the end of the procedure, a *Summary* is printed with 2 important pieces of information that will be required later:

- the **working directory** : this is where you will create the LABCAR-OPERATOR project
- the **auxiliary directory** : this is where you will create the maneuver file

Take note of these 2 paths and proceed with the tutorial.

### Generating the maneuver file

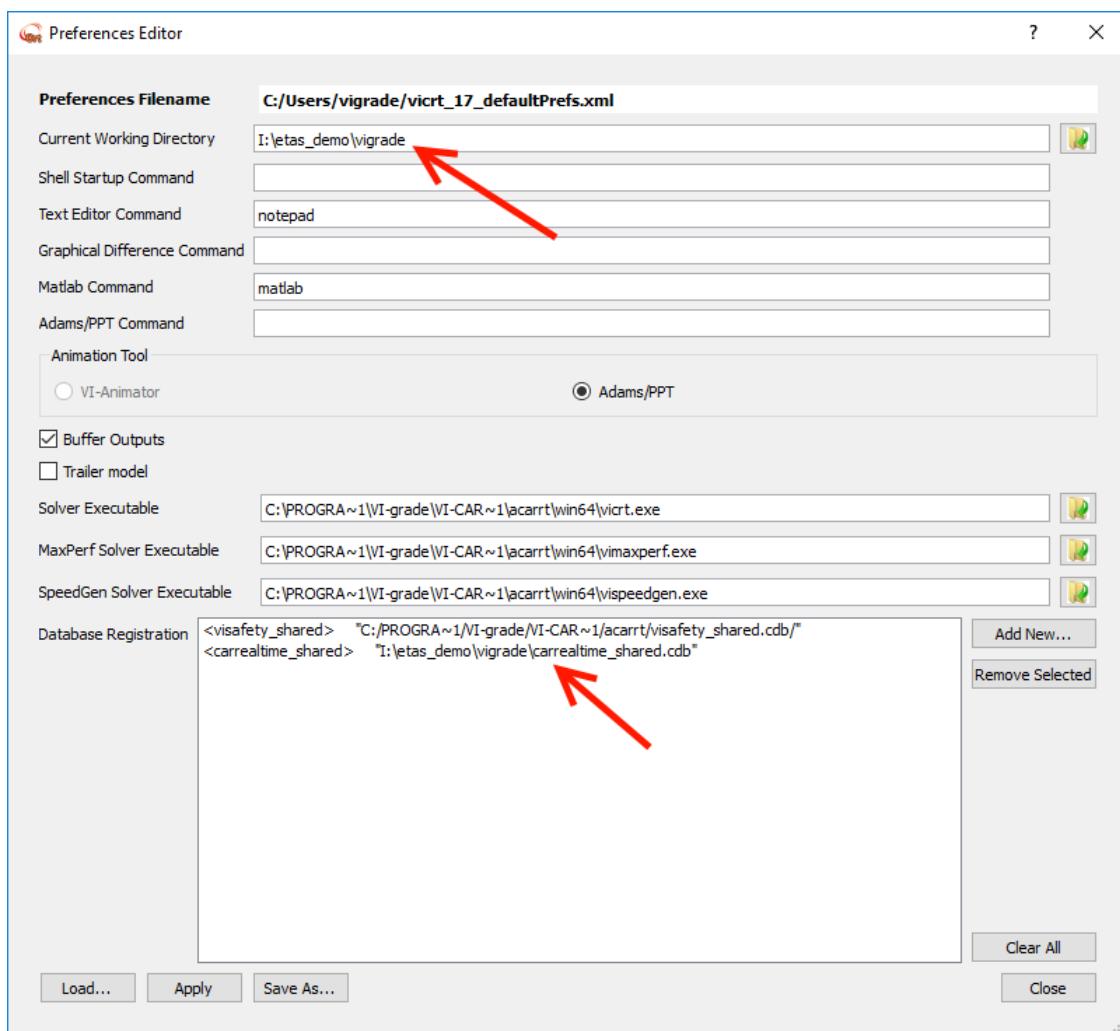
In the following, we suppose that the auxiliary directory (see Setting up the working directory) is located at the following: **I:\etas\_demo\vigrade**.

Please, open VI-CarRealTime, open the menu Edit->Preferences and do the following:

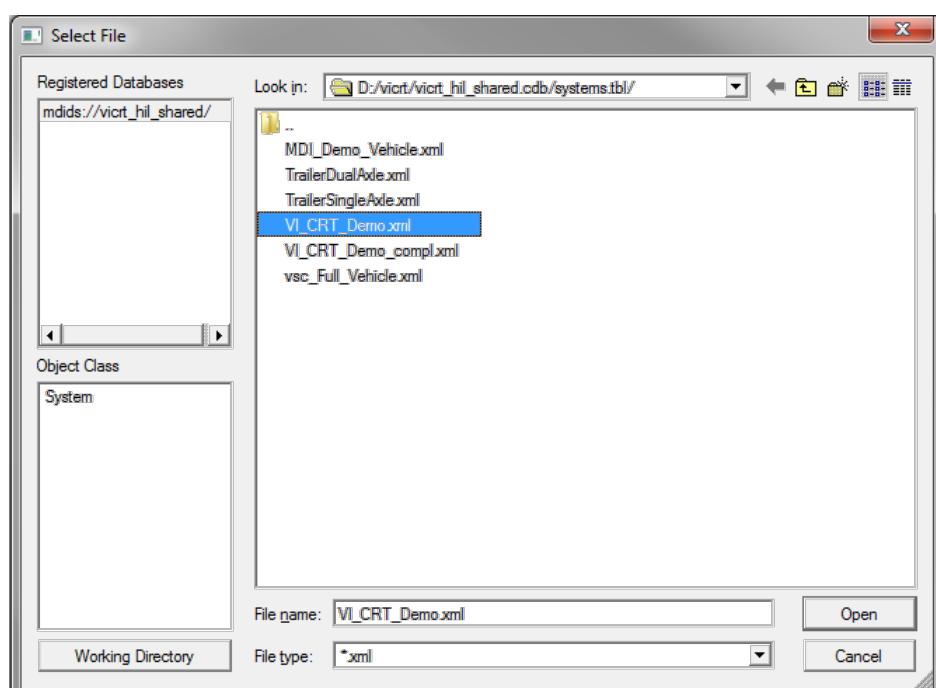
- change the VI-CarRealTime current working directory to **I:\etas\_demo\vigrade**
- change the default shared database <carrealtime\_shared> with the one located at **I:\etas\_demo\vigrade\carrealtime\_shared.cdb**
- click **Apply** button
- click **Close** button

You should end up with the following preferences:

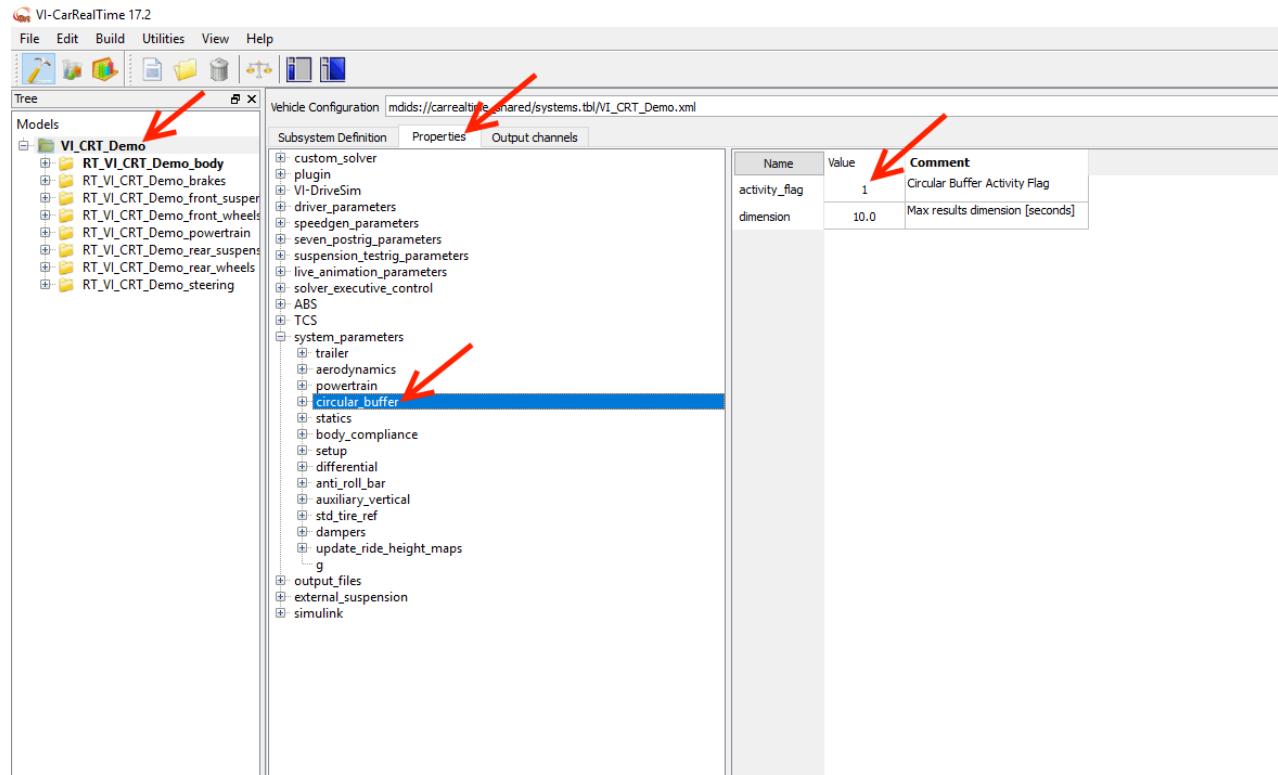
## VI-CarRealTime HIL Overlays



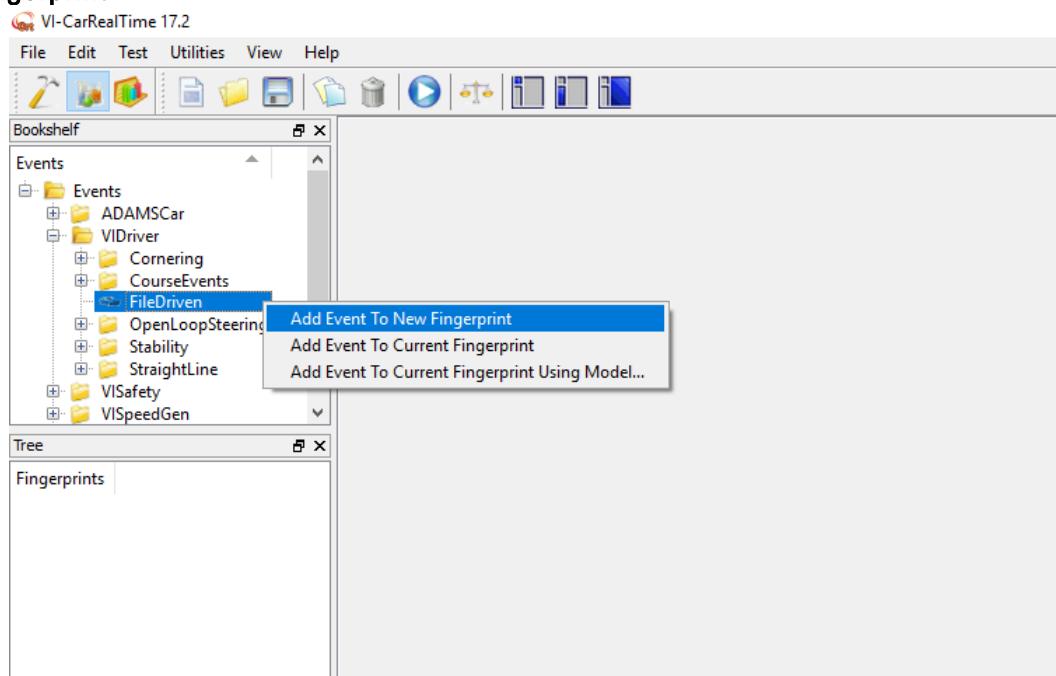
Now, open the model `VI_CRT_Demo.xml` from the `vicrt_hil_shared` database.



Then, **activate the circular buffer** by clicking on the root node of the VI\_CRT\_Demo project, switch to the **Properties** tab, activate the system\_parameters->circular\_buffer node and, then, set the activity flag to the value 1.

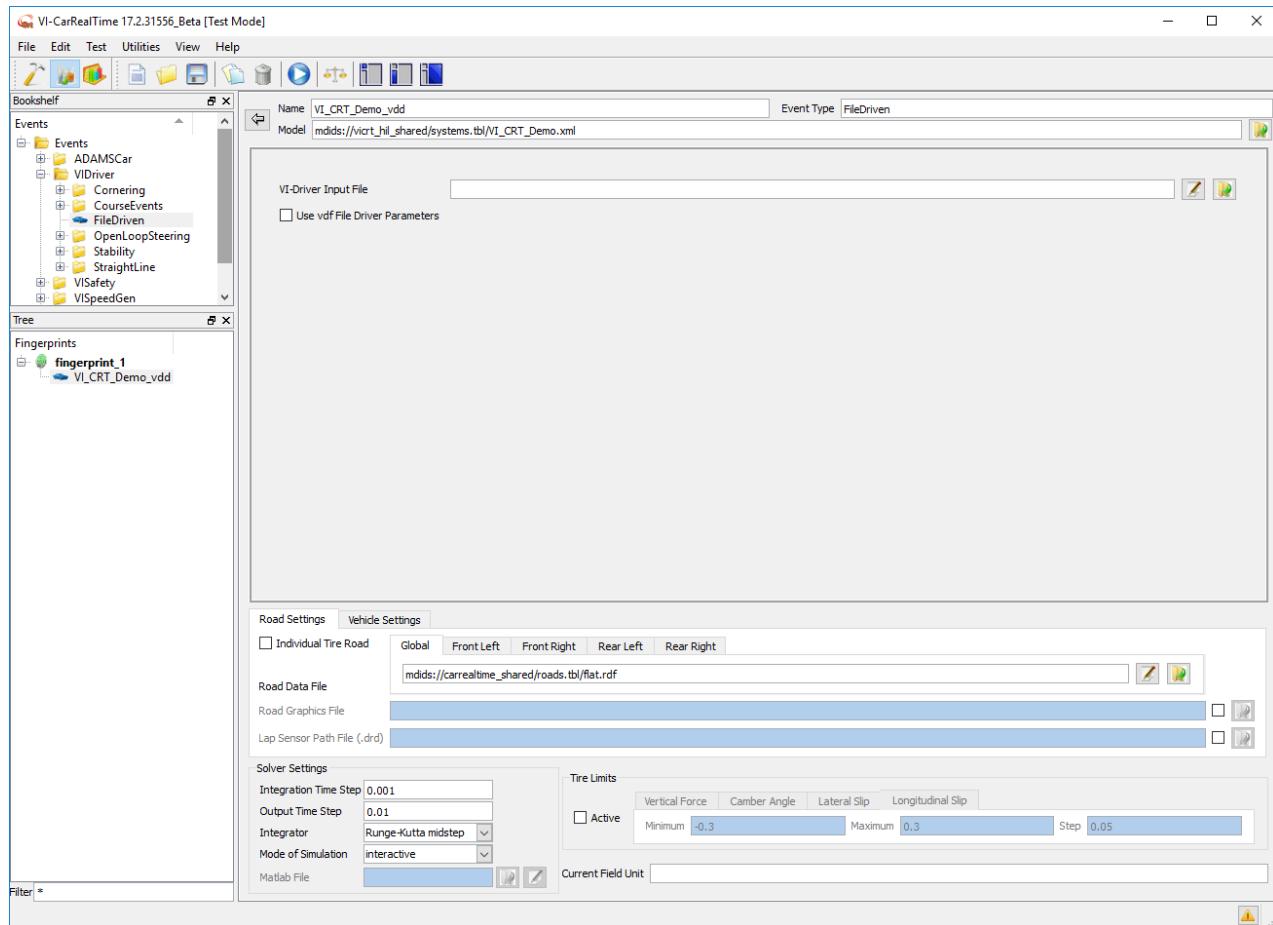


Now, you can switch to the **Test Mode** (  ) where the event specification will be defined. Create a new fingerprint by expanding the **Events** hierarchy, right click on the **FileDriven** item and select **Add Event to New Fingerprint**.



You will end up with a new event specification as depicted in the following.

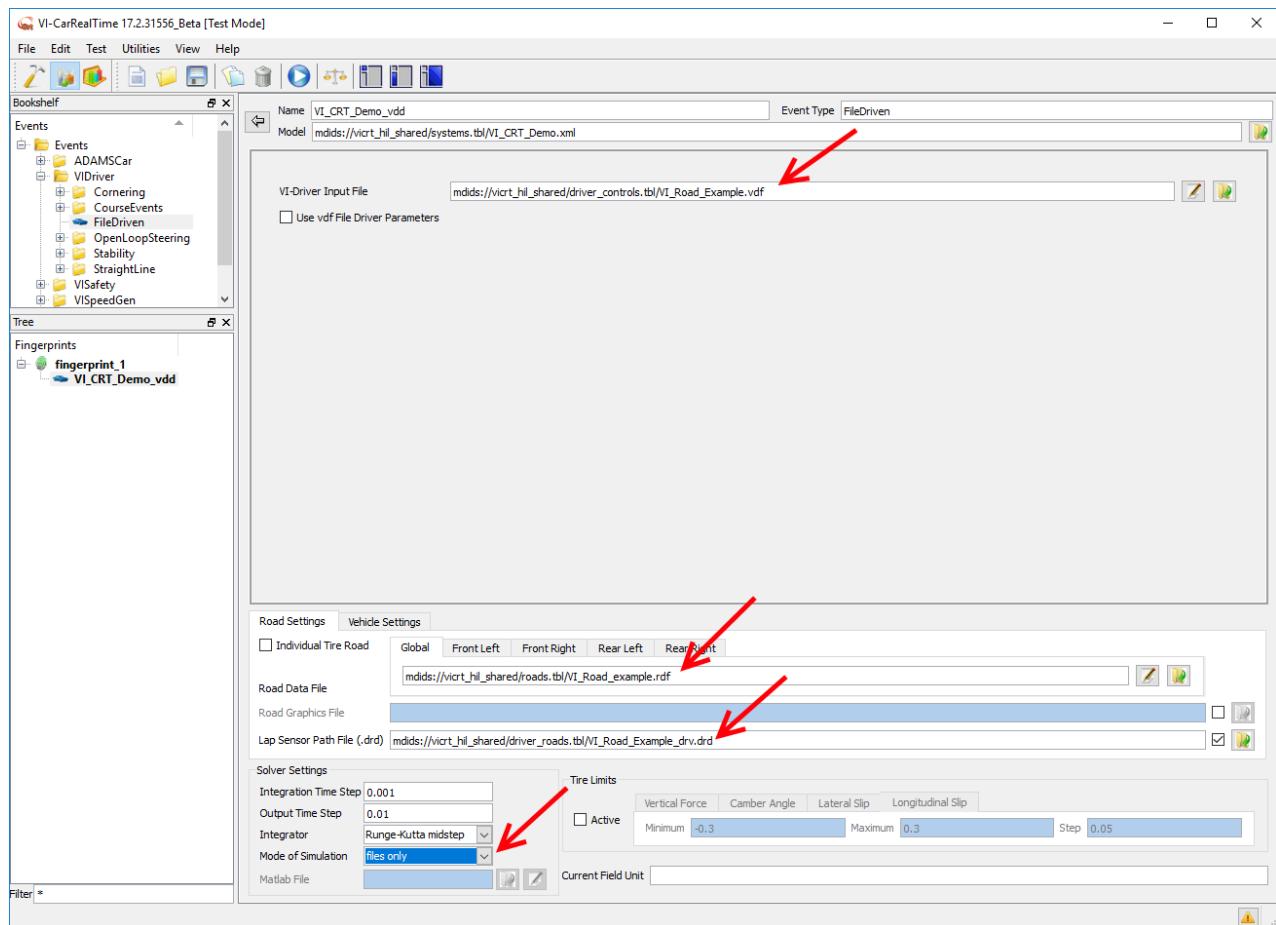
## VI-CarRealTime HIL Overlays



Now, the event specification must be completed as follows:

1. Select from the `vicrt_hil_shared` database the `VI_Road_Example.vdf` as the **VI-Driver input file**
2. Select from the `vicrt_hil_shared` database the `VI_Road_Example.rdf` as the **Road Data File**
3. Select from the `vicrt_hil_shared` database the `VI_Road_Example_drv.drd` file as the **Lap Sensor Path File**
4. Set the simulation mode to **files only**

You will end up with the following event specification:



You can now run the maneuver (  ) in order to generate the set of files needed by the VI-CarRealTime Matlab/Simulink interface. A console will appear with the following message:

```
File VI_CRT_Demo_vdd_send_svm.xml was successfully created..
```

The following files will be generated in the directory I:\etas\_demo\vigrade:

```
VI_CRT_Demo_vdd.cmd
VI_CRT_Demo_vdd.obj
VI_CRT_Demo_vdd_graphics.xgr
VI_CRT_Demo_vdd_road_graphics.xgr
VI_CRT_Demo_vdd_send_svm.xml
VI_CRT_Demo_vdd_viani_live_startup.py
vicrt_cdb.cfg
viroad.mtl
```

### Creating the LABCAR-OPERATOR project

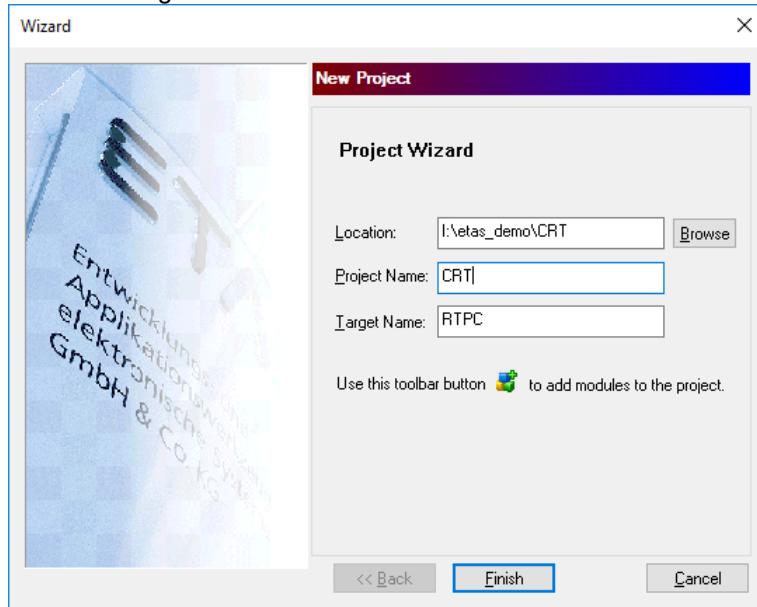
In the following, we suppose that the working directory (see Setting up the working directory) is located at the following: I:\etas\_demo.

You can now launch LABCAR-OPERATOR and create a new project. When asked, choose the following:

- **Location:** I:\etas\_demo\CRT
- **Project Name:** CRT

For the moment, do not try to configure a different Project Name. For more information, see [Configuring the OnPreBuild event](#).

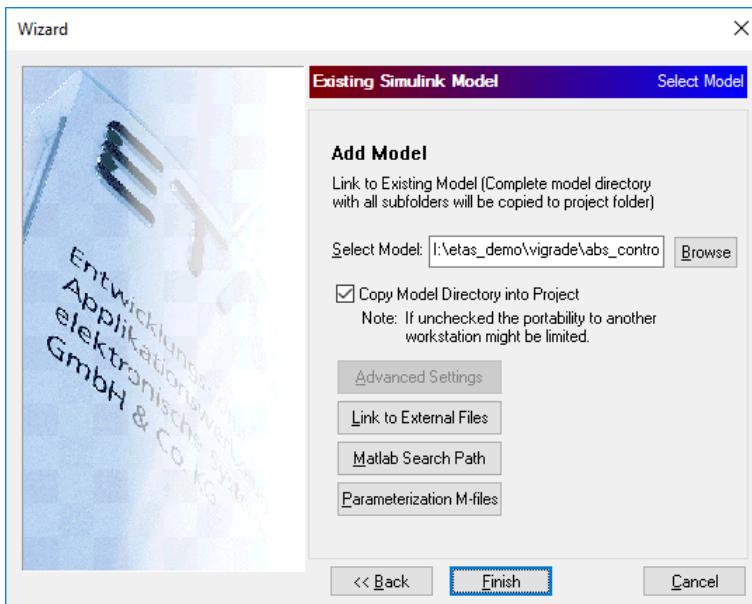
You should end up with the following:



Then, press **Finish**.

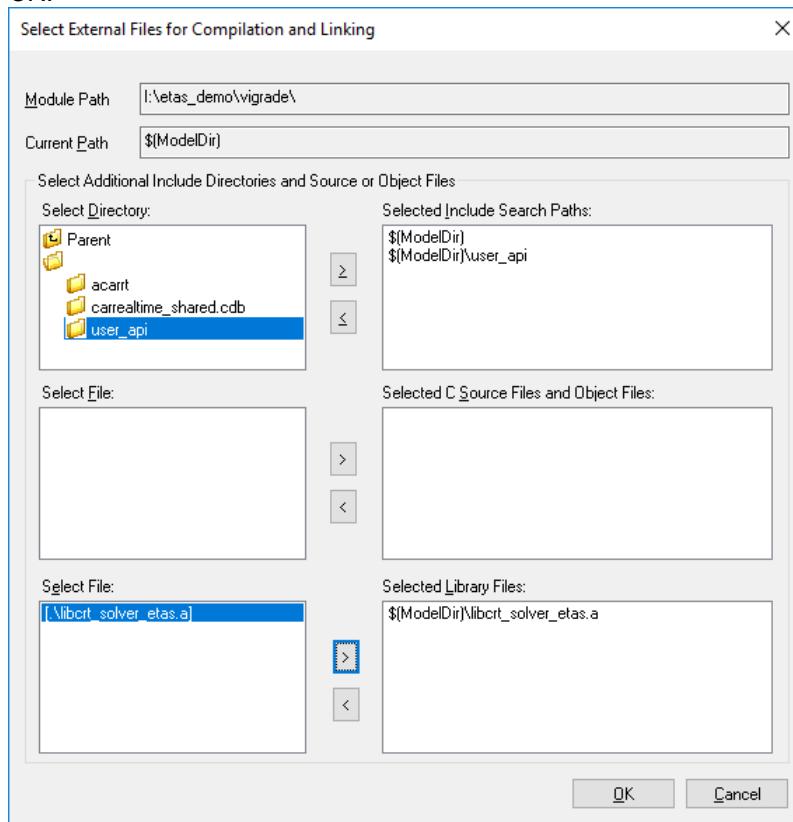
Now, click on menu *Project->Add Module...* and go through the wizard as detailed in the following:

- choose *Add Simulink(TM) Module* and click *Next*
- choose *Use Existing Simulink(TM) Module* and click *Next*
- click on the *Browse* button and select the module located at I:\etas\_demo\vigrade\abs\_control\_etas.mdl; you should end up with the following:



- now, click on the *Link to External Files* button and add the subdirectory user\_api to the *Selected Include Search Paths* and the file libcrt\_solver\_etas.a to the *Selected Library Files*; then, click the button

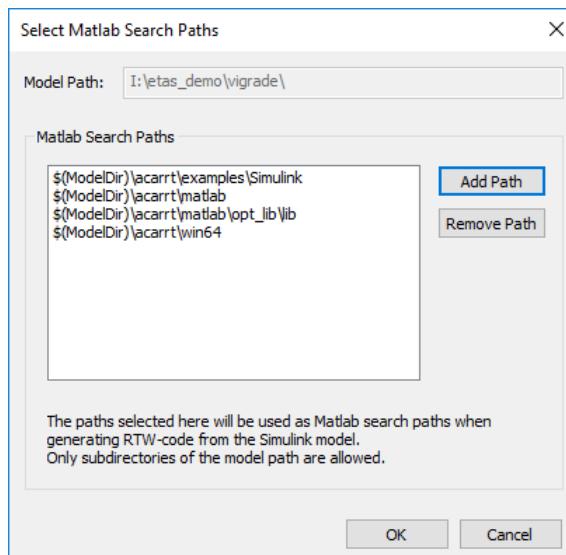
**OK:**



- now click on the *Matlab Search Paths* button and add the following subdirectories:

```
acarrt\examples\Simulink
acarrt\matlab
acarrt\matlab\opt_lib\lib
acarrt\win64
```

You should end up with the following:

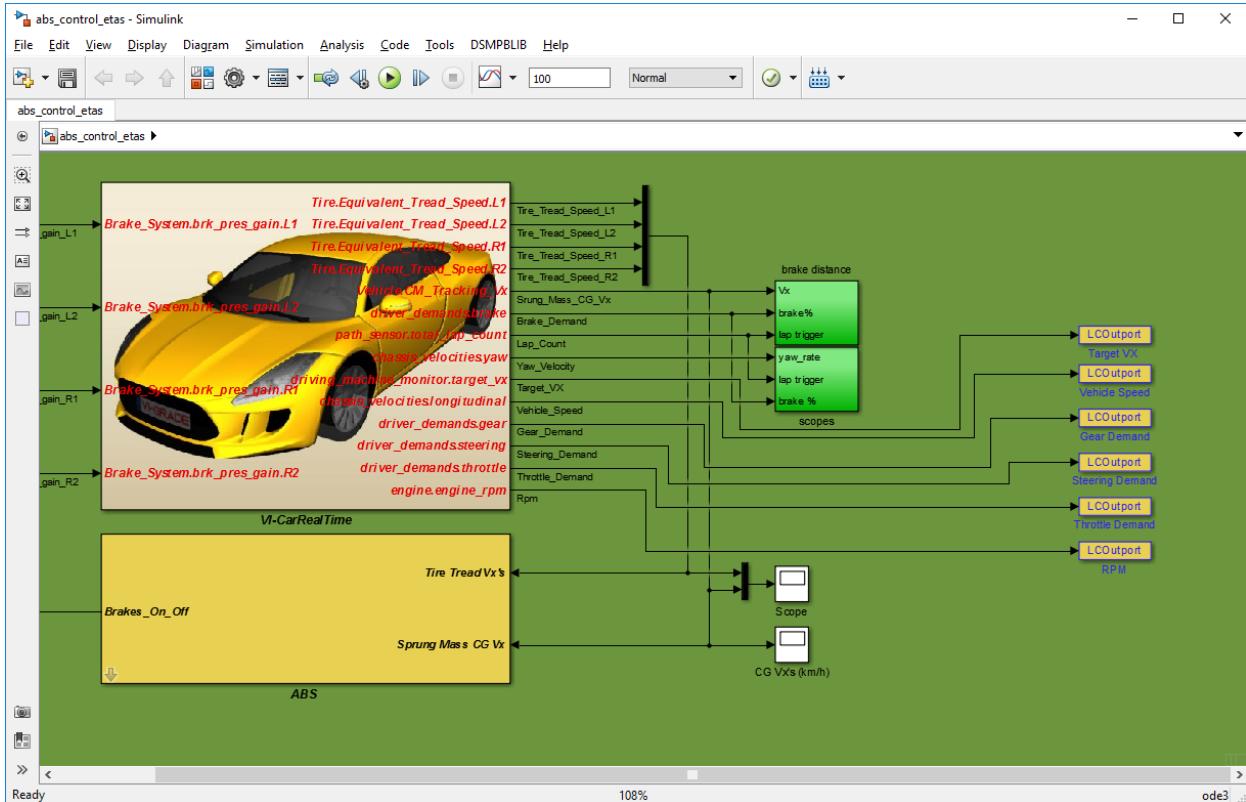


- You can now click the *OK* button and, then click the *Finish* button. By doing so, LABCAR-OPERATOR will now examine the Simulink model `abs_control_eta.sldl` that we just added. When the analysis has

## VI-CarRealTime HIL Overlays

finished you can save the project selecting the menu *File->Save All*.

**Note:** if interested in examining the just added Simulink module, you can now right click on it in the tree on the left of the application window and select the menu *Edit*. Simulink will open up with the correct setting (e.g., the *Matlab Search Paths*). You should see something like this:

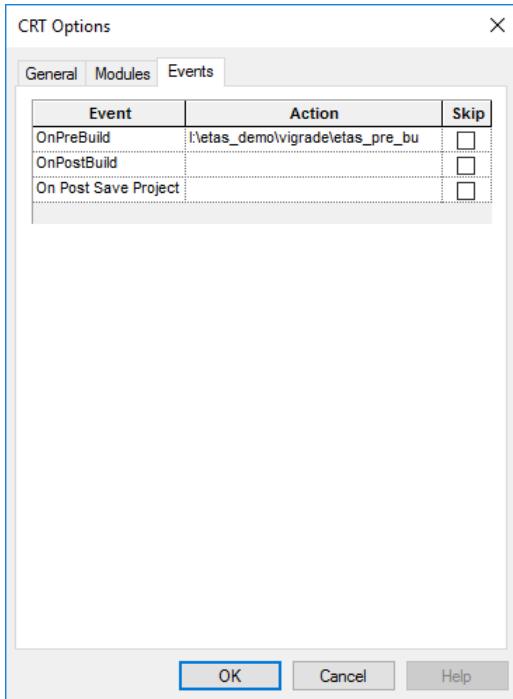


Once finished, please close Simulink without saving.

- Now, select the menu *Project->Options*, choose the tab *Events* and choose, for the *OnPreBuild* event the following batch file:

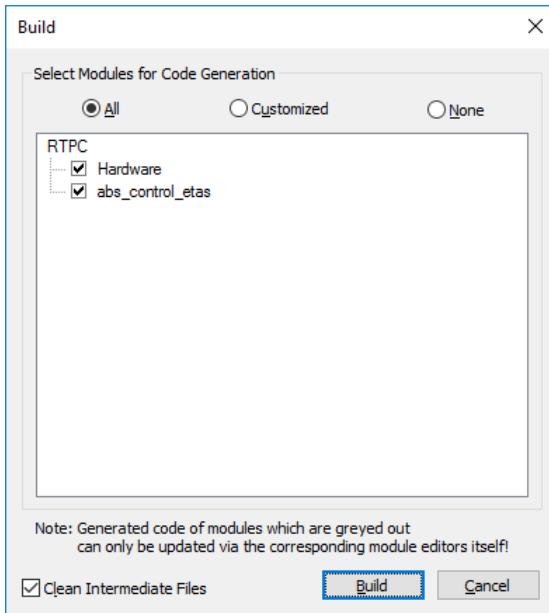
```
I:\etas_demo\vigrade\etas_pre_build.bat
```

You should end up with the following:



If you want to know more about the configuration of *OnPreBuild* event, please see [Configuring the OnPreBuild event](#).

- Now, you can save the project selecting the menu *File->Save All* and, then, select the menu *Project->Build*. You will be asked to confirm to start the build procedure:



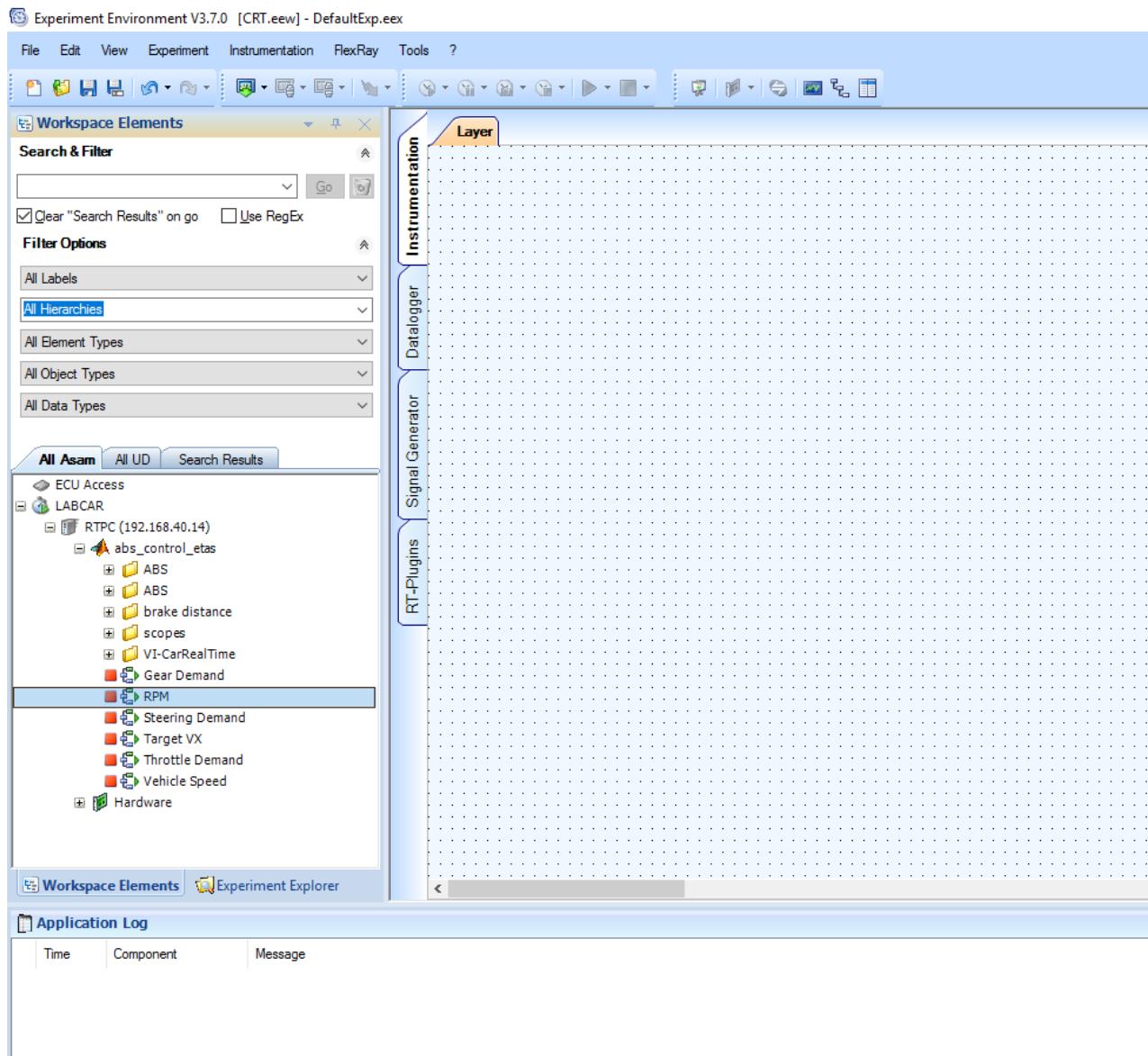
Click the *Build* button.

The build procedure will start and, at the end, a default experiment will be created in the ETAS Experiment Environment.

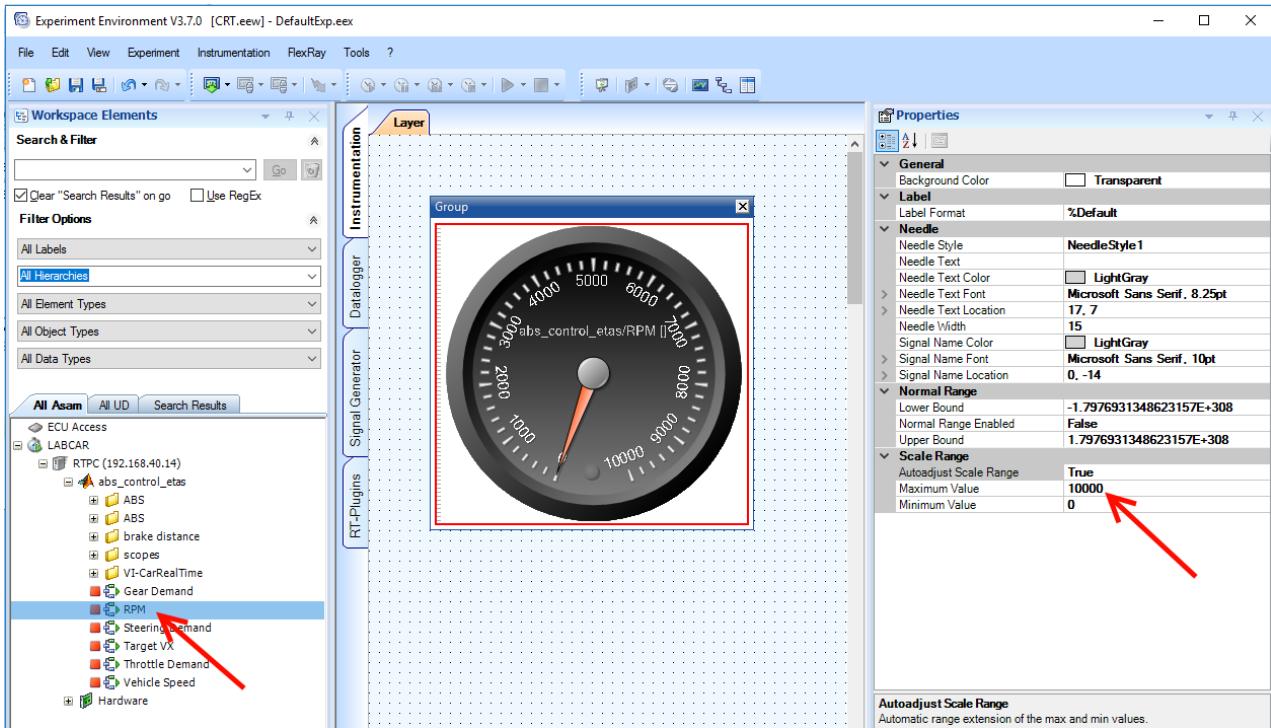
## VI-CarRealTime HIL Overlays

## Running the experiment

Once LABCAR-OPERATOR has completed the build procedure, ETAS Experiment Environment will be opened with a new project already pre-configured.

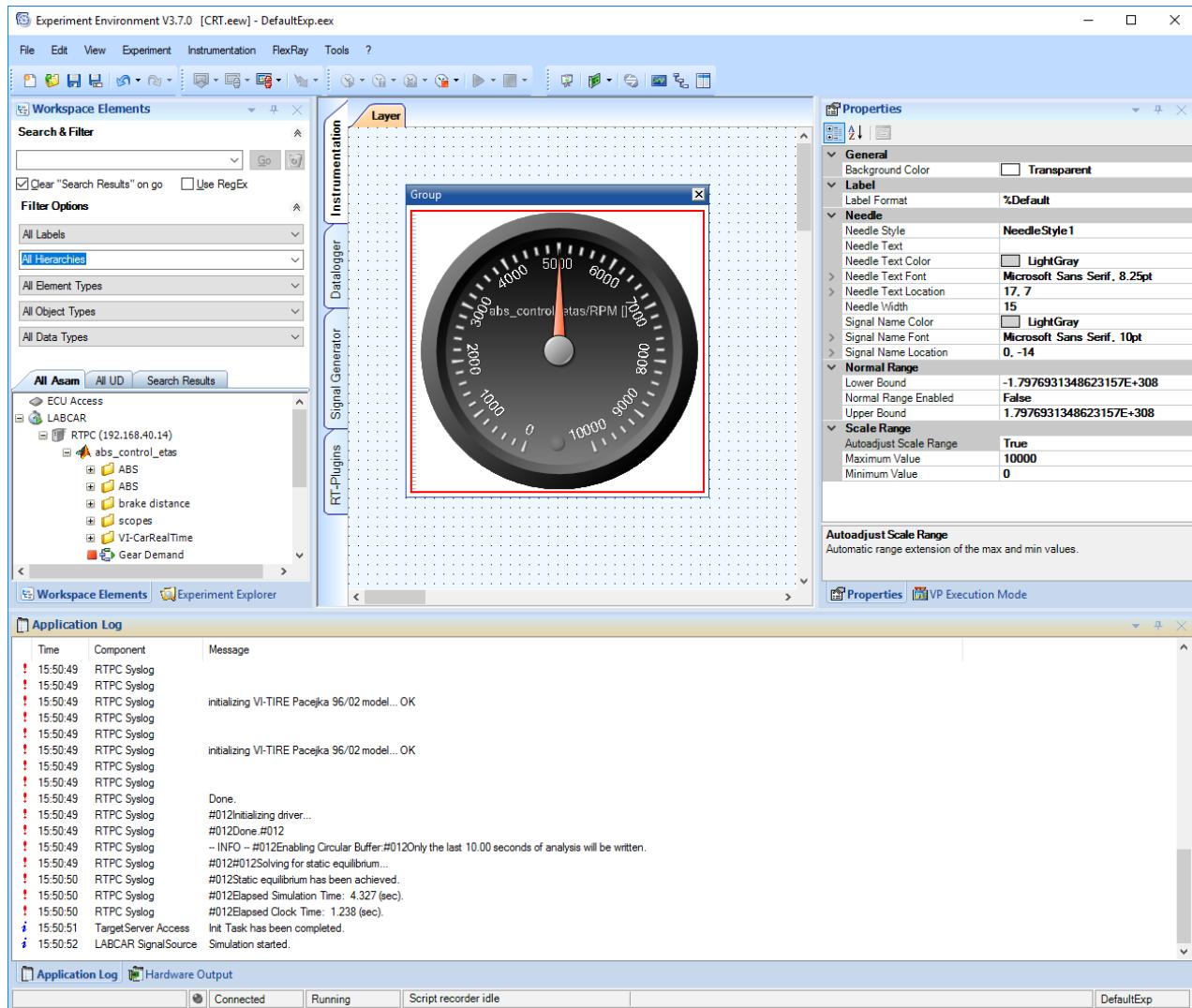


You can now add some of the available outputs to the observed variables by dragging and dropping the output to the available *Layer*. In the following, we dragged and dropped the *RPM* output, chose the *Circular Gauge* instrument and set its *Maximum Value* to 10000.



Now, in order to run the experiment, just push the *Download* button and, then, the *Start Simulation* button. The simulation will start and the Simulink model will run in conjunction with the VI-CarRealTime solver.

## VI-CarRealTime HIL Overlays



### 3.4.6 Configuring the OnPreBuild event

The ETAS build procedure is composed by several steps. The *OnPreBuild* event is the first step of this procedure and it is the step that we use to upload and synchronize to the target RTPC machine all the files that are needed by VI-CarRealTime. This is implemented in the tutorial by setting the batch file `etas_pre_build.bat` as the *OnPreBuild* event (see [Creating the LABCAR-OPERATOR project](#)).

The batch file `etas_pre_build.bat` takes care of uploading the files in the *auxiliary directory* to their corresponding location on the RTPC target machine as described in the following:

```
I:\etas_demo\vigrade\hil_license.lic
-> /home/user/hil_license.lic

I:\etas_demo\vigrade\VI_CRT_Demo_vdd_send_svm.xml
-> /home/user/var/VI_CRT_Demo_vdd_send_svm.xml

I:\etas_demo\vigrade\carrealtime_shared.cdb
-> /home/user/var/carrealtime_shared.cdb

I:\etas_demo\vigrade\vicrt_cdb_hil.cfg
-> /home/user/var/vicrt_cdb.cfg
```

Keep in mind that `etas_pre_build.bat` can be run also standalone in order to make a synchronization without using LABCAR-OPERATOR. Every time the script is run, the log `I:\etas_demo\vigrade\etas_pre_build.log` will be created. You can examine it in case of issues.

**Note:** in order to make its synchronization, the `etas_pre_build.bat` step will use the `rsync` utility provided by ETAS in its toolchain.

Normally, you do not need to configure the script `etas_pre_build.bat` but, if needed, this can be done. The script `etas_pre_build.bat` is accompanied by the text file `etas_pre_build.cfg` whose contents are, by default, as follows:

```
CRT_ROOT=C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\..\..\..
TARGET_DIR=I:\etas_demo\CRT
```

The file `etas_pre_build.cfg` can be edited in order to change some of the parameters of the synchronization. As an example, you can set `AnotherSendFile.xml` as the send file to be used by adding the line with argument `CRT_SEND_FILE` (keep in mind that, anyway, it must reside in the *auxiliary directory*)

```
CRT_ROOT=C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\..\..\..
TARGET_DIR=I:\etas_demo\CRT
CRT_SEND_FILE=AnotherSendFile.xml
```

The most used parameters (and their defaults values) are the following:

| Parameter        | Default Value                                                                                |
|------------------|----------------------------------------------------------------------------------------------|
| TARGET_DIR       | I:\etas_demo\CRT                                                                             |
| CRT_SEND_FILE    | VI_CRT_Demo_vdd_send_svm.xml                                                                 |
| ETAS_IP          | 192.168.40.14                                                                                |
| ETAS_LCO_ROOT    | C:\Program Files (x86)\ETAS\LABCAR-OPERATOR5.4                                               |
| ETAS_RSYNC_SHARE | user                                                                                         |
| ETAS_WORKDIR     | var                                                                                          |
| LOGFILE          | I:\etas_demo\vigrade\etas_pre_build.log                                                      |
| CRT_ROOT         | C:\Program Files\VI-grade\VI-CarRealTime 17\acarrt\examples\etas\..\..\..                    |
| RSYNC_PATH       | C:\Program Files (x86)\ETAS\LABCAR-OPERATOR5.4\BuildSystemTools\mingw\msys\1.0\bin\rsync.exe |

# 4 Appendix A: VI-Tire

## 4.1 Overview

The VI-Tire module is the framework used by VI-grade products to access both native and user-defined tire models.

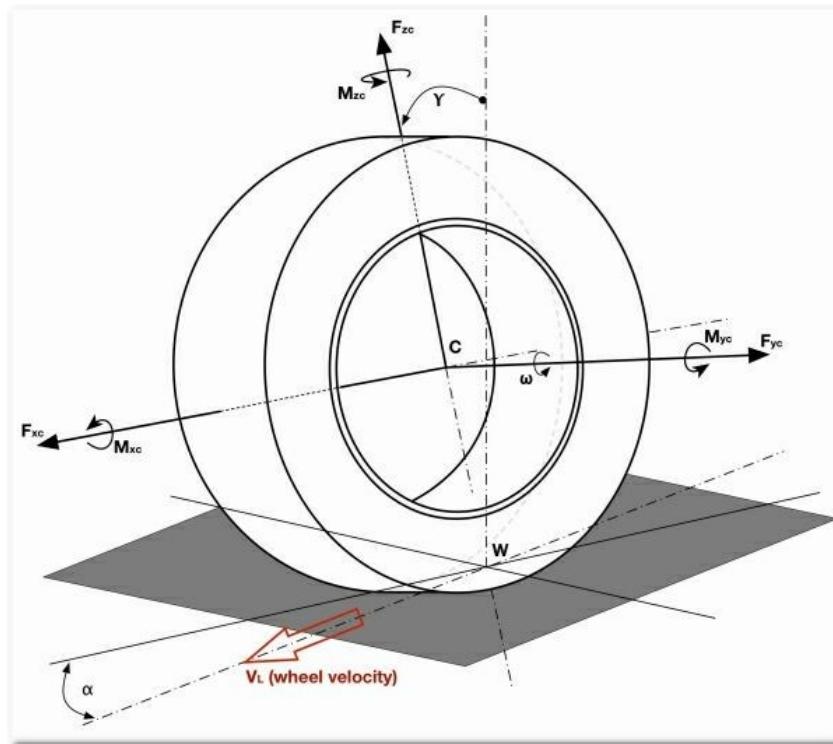
The tire models currently integrated into VI-Tire module can be divided into three families:

1. **Native tire models**, entirely developed and maintained by VI-grade.  
This category includes PAC2002 and PAC-MC formulations.
2. **External 3rd-party tire models**, interfaced directly to VI-Tire.  
In this case only the interface to the tire routines is maintained by VI-grade. The Cosin FTire, Fraunhofer CDTire, Michelin TameTire and Siemens MF-Tyre/MF-Swift belong to this family.
3. **External user-defined 3rd-party tire models**, based on the STI/TYDEX interface. Examples are provided to help the users to interface their own tire models with the VI-grade simulation software.

## 4.2 Tire Axis System

The VI-Tire basic interface is based on STI conventions. The most of the tire input parameters are referred to the wheel center (C axis system), while the output forces and moments are computed at the tire-road contact patch (W axis system).

The forces and moments at the contact patch are then transformed back to the wheel center. Results for both systems are available as standard output.

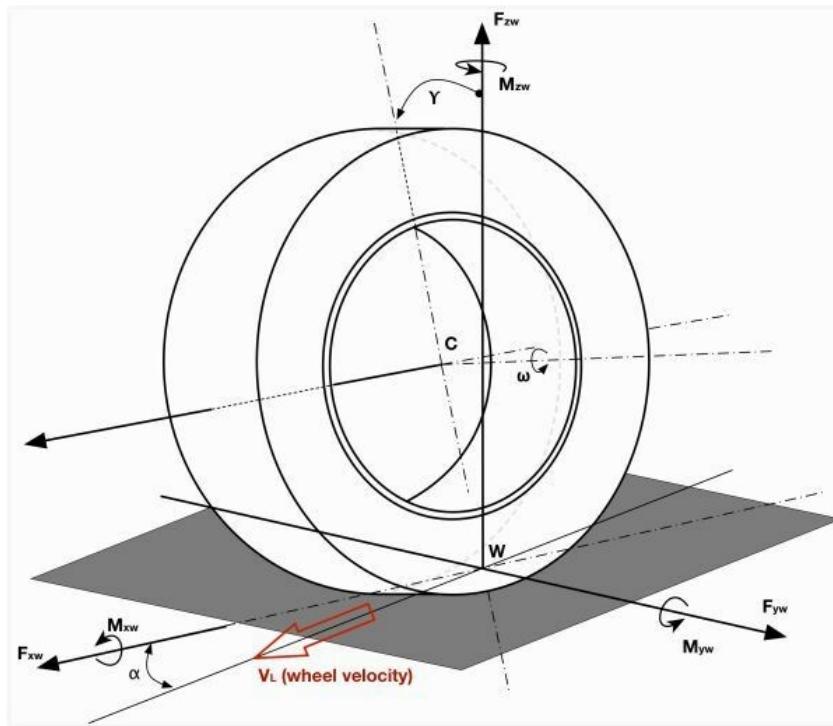


The tire "C" (center) axis system has the origin fixed in the center of the wheel.

The  $\mathbf{X}_c$  axis lies in the wheel central plane and it is parallel to the ground.

The  $\mathbf{Y}_c$  axis is the spin axis of the wheel. It follows the inclination angle  $\gamma$ .

The  $\mathbf{Z}_c$  axis points upwards, lies in the wheel central plane and follows the inclination angle  $\gamma$ .



The tire "W" (wheel) axis system has the origin fixed in the tire-surface contact patch.

The  $X_w$  axis results from the intersection of the wheel central plane with the track surface.

The  $Y_w$  axis is the projection of the spin axis onto the ground.

The  $Z_w$  axis is normal to the ground and points upwards.

**Notes:**

1. according to STI conventions the track surface can be oriented (local normal of the track surface).
2. the inclination angle results from the angle between the wheel central plane and the  $X_w$ - $Y_w$  plane.

## 4.3 Tire Slip Definition

The tire slip parameters are all computed at the W axis system.

The longitudinal slip velocity is defined as:  $V_{sx} = V_x - \omega r_e$

where:

$V_x$  = wheel longitudinal velocity;

$\omega$  = wheel rotational (spinning) velocity;

$r_e$  = effective rolling radius =  $V_x / \omega$  at free rolling condition.

The longitudinal slip follows as:  $\kappa = -V_{sx} / V_x$

The lateral slip velocity is equal to the lateral speed of the wheel with respect to the track surface, thus:

$$V_{sy} = V_y$$

The resultant lateral slip is:

$$\alpha = \tan(V_{sy} / |V_x|)$$

The actual rolling speed is function of the effective radius:

$$V_r = \omega r_e$$

## 4.4 Effective Radius

The tire effective radius takes into account the deflection of the tire, and its effect on the actual tire velocities.

All the Adams 2011r1 native (handling) tires adopt the Pacejka [1] non-linear equation, expressed in the general form:

$$r_e = r_0 + \Delta r - \{ q_1 \rho_z + D \tan(B\rho_z) \}$$

where:

$r_0$  = unloaded tire radius

$\Delta r$  = tire radial growth

$\rho_z$  = tire normal deflection =  $(r_0 - r) / (F_{Znom}/C_z)$

Which, simplified, leads to:

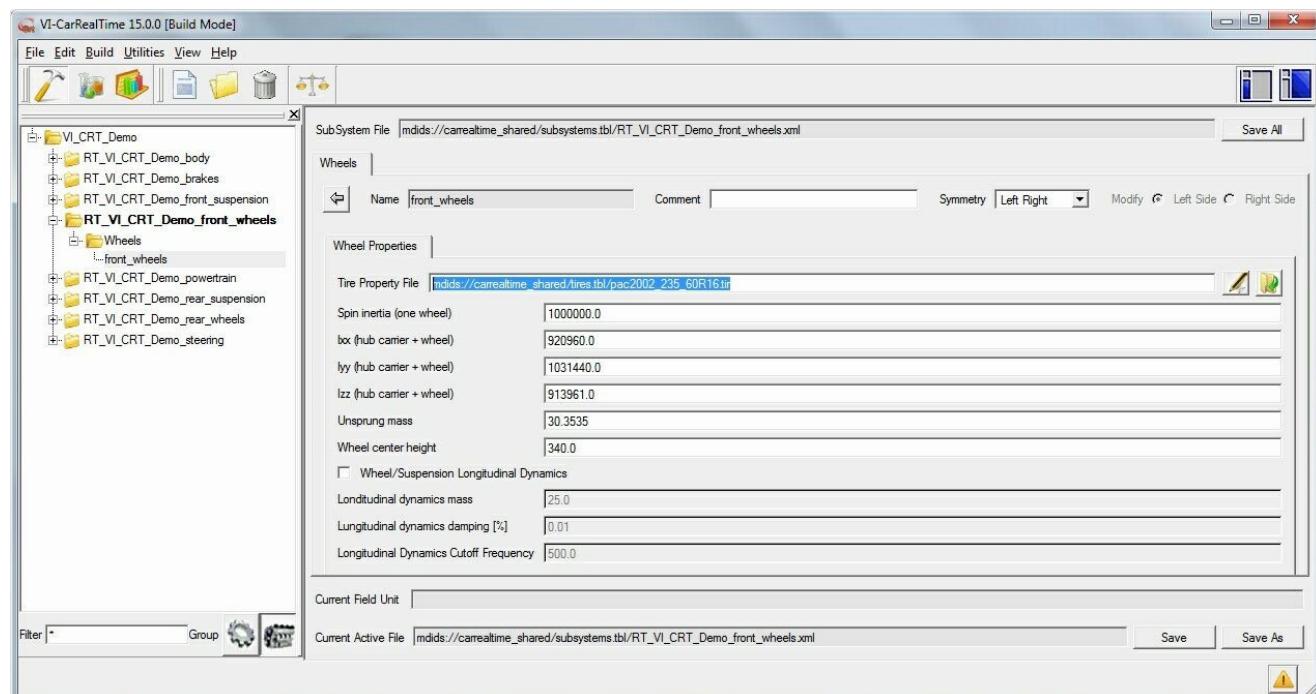
$$r_e = r_0 - (F_{Znom}/C_z) \{ F\rho_z + D \tan(B\rho_z) \}$$

The related coefficients can be found in the [VERTICAL](#) data block of the properties file.

## 4.5 Selecting The Tire Model

The tire model is entirely defined by its properties file.

The tire properties file has to be defined in the front/rear Wheels subsystem of VI-CarRealTime:



## 4.6 Tire Models Dispatching

The VI-Tire supported tire models selection mechanism is based on a combination of specific parameters inside the properties file MODEL section.

|                      |                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------|
| PROPERTY_FILE_FORMAT | defines the data format and its value is checked as first and, if not set to 'USER', identifies the tire model directly. |
| FUNCTION_NAME        | defines the (shared) library and function name whenever a USER tire model is selected by PROPERTY_FILE_FORMAT.           |
| MODEL_TYPE           | defines the tire model type whenever a USER tire model is selected by PROPERTY_FILE_FORMAT                               |
| USER_SUB_ID          | additional routine identifier, quite obsolete and rarely used.                                                           |

| TIRE MODEL               | VENDOR     | NOTES                 | PROPERTY_FILE_FORM_AT | FUNCTION_NAME   | MODEL_TYPE | USER_SUB_ID |
|--------------------------|------------|-----------------------|-----------------------|-----------------|------------|-------------|
| PAC96 ADV                | VI-grade   |                       | MF_05                 |                 |            |             |
| PAC96 ADV                | VI-grade   |                       | PAC2002               |                 |            |             |
| FTire                    | Cosin      | Separate Installation | FTIRE                 |                 |            |             |
| FTire                    | Cosin      | Separate Installation | USER                  | CTI::TYR510     |            |             |
| MF-TYRE/MF-SWIFT v6.2    | Siemens    | Separate License      | USER                  | TYR815          |            |             |
| MF-TYRE/MF-SWIFT v6.2    | Siemens    | Separate License      | USER                  |                 |            | 815         |
| MF-TYRE/MF-SWIFT v2020.1 | Siemens    | Separate License      | USER                  | mtyre_mfswift_7 |            |             |
| CDTire                   | Fraunhofer | Separate Installation | CDTIRE                |                 |            |             |
| TameTire                 | Michelin   | Separate Installation | TAMETIRE              |                 |            |             |

## 4.7 Internal models

Enter topic text here.

### 4.7.1 PAC2002

Enter topic text here.

### 4.7.2 MF\_O5

Enter topic text here.

### 4.7.3 PACMC

Enter topic text here.

### 4.7.4 Legacy Tire Models

The basic VI-Tire native models for vehicle simulations (PAC2002/MF05) have been extensively modified in order to improve computational stability and overall efficiency, implementing new features (obfuscation) and improved save and restore capabilities.

These revised implementations of existing tire models are now used by default and even if the algorithms are exactly the same of the previous versions, minor differences may appear under some conditions.

Additionally, some rarely used features like contact mass and parking torque have been dropped to allow general efficiency improvements.

To avoid any problems with legacy results comparisons, the older (legacy) implementation is still accessible with a minor modification to the tire property file. To get back the legacy implementation of PAC2002 and MF05 tire models the new `LEGACY_FORMULATION` key has been added to the `MODEL` data block of the TIR file.

```
[MODEL]
LEGACY_FORMULATION = 0
```

Setting `LEGACY_FORMULATION` equal to 1 (the default is 0) will cause VI-Tire to switch to the legacy implementation.

Please note that tire property file configured for the legacy mode are not compatible with the obfuscation process.

## 4.8 Cosin FTire interface

The Cosin FTire interface (CTI) makes it possible to run simulations using FTire tire model with VI-CarRealTime. The interface is completely transparent to the user, being the tire and road selection mechanism exactly the same of the native tires.

Please note that you must first download and install the appropriate Cosin software package from the Cosin website.

The selection of an FTire model in your vehicle is entirely delegated to the tire file (see [Selecting The Tire Model](#) section).

### Limitations:

1. the VI-grade native roads are supported in addition to the native FTire ones, but the rgr is recommended in order to achieve the best computational efficiency. Please refer to FTire documentation for supported road models.

**Licensing:** this tire model requires a separate license and license server. Please contact your VI-grade representative for more information.

To understand which road formats this tire model is compatible with, please refer to the [Tire/Road Compatibility](#) section.

For more information on using FTire with VI-CarRealTime, please refer to the chapter Using FTire At Best In VI-CarRealTime

## 4.9 Siemens MF-Tyre/MF-Swift interface

The MF-Tyre/MF-Swift interface makes it possible to run simulations using the Siemens PLM Software MF-Tyre/MF-Swift tire model with VI-CarRealTime. The interface is completely transparent to the user, and the tire and road selection mechanism is exactly the same of the native tires.

Two complementary versions of the MF-Tyre/MF-Swift library are supported:

- MF-Tyre/MF-Swift v2020.1 is the current reference version. It grants real-time compatibility including parallel evaluation
- MF-Tyre/MF-Swift v6.2 is a legacy product, shipped for compatibility, but not offering real-time capabilities

To understand which road formats this tire model is compatible with, please refer to the [Tire/Road Compatibility](#) section.

The tire model is completely defined by the parameters in the tire properties file, and particular attention has to be paid to the three digits operating mode (see related documentation included in the package):

`USE_MODE = BCD`, where:

### B = contact method

- 0/1 = smooth road contact, single contact point <default> (\*\*)  
2 = smooth road contact, circular cross section (motorcycle tyre) (\*)

## Appendix A: VI-Tire

- 3 = moving road contact, flat surface (NOT SUPPORTED)
- 4 = road contact for 2D roads (using traveled distance) (\*)
- 5 = road contact for 3D roads

**C = dynamics**

- 0 = Steady-state evaluation (< 1 Hz) <default>
- 1 = Transient effects included, relaxation behavior (< 10 Hz, linear)
- 2 = Transient effects included, relaxation behavior (< 10 Hz, nonlinear)
- 3 = Rigid ring dynamics included (< 100 Hz, nonlinear) (\*\*\*)
- 4 = Rigid ring dynamics + initial statics (same as 3, but with finding static equilibrium of the tyre belt) (\*)

**D = Magic Formula Force Evaluation**

- 0 = No Magic Formula evaluation (Fz only)
- 1 = longitudinal forces/moment only (Fx,My)
- 2 = lateral forces/moment only (Fy,Mx,Mz)
- 3 = uncombined forces/moment (Fx,Fy,Mx,My,Mz)
- 4 = combined forces/moment (Fx,Fy,Mx,My,Mz)
- 5 = combined forces/moment (Fx,Fy,Mx,My,Mz) + turnslip

(\*) option not supported in MF-Tyre/MF-Swift 2020.1.

(\*\*) in MF-Tyre/MF-Swift v2020.1 the single point contact (=0) and smooth road contact (=1) are distinct options. The single point contact will be deprecated in the future, it is advised to always use smooth road contact if possible.

(\*\*\*) in MF-Tyre/MF-Swift v2020.1 there is a single Rigid ring option (=3) which always includes the initialization of the belt dynamics.

No special option for the MF-Tyre/MF-Swift tire are provided in the solver GUI, so the TIR files can be selected in the same way as the other (native) tire models

**Limitations:** the MF-Tyre/MF-Swift tire cannot be used over moving roads.

**Notes:**

1. The tire model user has to subtract the belt mass and inertia from the mass of the wheel body manually when rigid ring dynamics is activated (dynamics mode 3, 4).
2. When the MF-Tyre/MF-Swift tire is used together with VI-grade road models only contact methods 0/1/2 and 5 should be used (a specific warning is issued at runtime).
3. The only PROPERTY\_FILE\_FORMAT supported keyword is 'USER' (see MODEL data block). The 'MF-TYRE' is not supported, although some files may contain it.

**Licensing:**

- Some MF-Tyre features can be used free of charge.

For a more details regarding the MF-Tyre/MF-Swift models, please refer to:

- [MF-Tyre/MF-Swift v2020.1 reference guide](#)
- [MF-Tyre/MF-Swift v6.x reference guide](#)

In order to enable the realtime mode for MF-Tyre/MF-Swift, the environment variable:

VICRT\_MFTYRE\_CALL\_MODE=HIL

This should be set in the execution environment. VI-DriveSim defines it by default.

## 4.10 Michelin Tametire interface

VI-CarRealTime works in combination with TameTire 6.1 provided that the related libraries, data and licenses from Michelin are installed. Please contact your VI-grade representative for additional information about getting access to Michelin Tametire package.

The Tametire model is automatically selected by VI-CarRealTime when the specific property (.tir) file is present in the vehicle model. The file itself is provided directly by Michelin and can be identified by the key PROPERTY\_FILE\_FORMAT = 'TAMETIRE' within the [MODEL] data block. It contains a Michelin proprietary encrypted data section, so manual editing should be avoided.

The generic tire temperature output channels of VI-grade simulation applications are mapped to TameTire actual output as follows:

Tire\_Temperature\_1 = CORE TEMPERATURE (Tgi)  
Tire\_Temperature\_2 = AVERAGE CORE TEMPERATURE (TempCore)  
Tire\_Temperature\_3 = SURFACE TEMPERATURE (TempSurf)

To understand which road formats this tire model is compatible with, please refer to the [Tire/Road Compatibility](#) section.

## 4.11 Fraunhofer CDTire

VI-CarRealTime works in combination with CDTire 4.2.8 provided that the related libraries, data and licenses from Fraunhofer are installed. Please contact your VI-grade representative for additional information about getting access to the Fraunhofer CDTire package.

The CDTire model is automatically selected by VI-CarRealTime when the specific property (.tir) file is present in the vehicle model. The file itself can be identified by the key PROPERTY\_FILE\_FORMAT = 'CDTIRE' within the [MODEL] data block.

To understand which road formats this tire model is compatible with, please refer to the [Tire/Road Compatibility](#) section.

## 4.12 User Defined Tire Model

Custom tire equations can be connected to VI-CarRealTime wrapping the custom code as a user tire model. The following chapters explain how the user tire interface works.

### 4.12.1 Tire User Interface

There are two kind of tire interface types to create user tire models, one is STI/TYDEX based and the other is CRT specific. The main differences reside in the amount of solver information directly available through the interface itself, but using one or another is a matter of convenience, looking at the model features and required input data.

The interface type has to be addressed using the INTERFACE\_TYPE parameter in the MODEL data block of the (user) tire data file.

The INTERFACE\_TYPE = 'STI' selects the standard STI/TYDEX interface, while INTERFACE\_TYPE = 'CRT' instructs the solver to call the user tire using the CRT interface.

Attention has to be made in selecting the correct interface, matching the user tire function actual "signature". Failing in setting the proper INTERFACE\_TYPE will result in solver crash since the solver will call the user function with the wrong parameters list structure.

STI interface call (C programming language).:

## Appendix A: VI-Tire

```
tire_c_sti_user(//
 // INPUT PARAMETERS
 //
 int* outputDevice, int* modelSwitch,
 int* jobFlag, int* tireID,
 double* simulationTime, double* wheelCarrierLocation,
 double* transformationMatrix, double* wheelAngle,
 double* wheelCarrierVelocity, double* wheelCarrierOmega,
 double* wheelOmega, int* diffeqNumber,
 double* diffeqVariables, int* tireParNumber,
 double* tireParametersArray, int* tireFileNameLength,
 char* tireFileName, void* externalRoad,
 int* roadModel, int* roadParNumber,
 double* roadParametersArray, int* roadFileNameLength,
 char* roadFileName,
 //
 // OUTPUT PARAMETERS
 //
 double* wheelForces, double* wheelTorques,
 double* diffeqInitialValues, double* diffeqDerivatives,
 char* tireModel, int* outputValuesNumber,
 double* outputValuesArray, int* realWorkArraySize,
 double* realWorkArray, int* intWorkArraySize,
 int* intWorkArray, int* errorID
);
```

| VARIABLE NAME        | TYPE        | SPECIFICATIONS                                                                                                                                                                                                                                                                                                                                |
|----------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INPUT                |             |                                                                                                                                                                                                                                                                                                                                               |
| outputDevice         | int*        | the output device for error and information messages                                                                                                                                                                                                                                                                                          |
| modelSwitch          | int*        | the tire model type switch<br>0 = static analysis model<br>1 = dynamic analysis model<br>2 = static solver analysis model (e.g.VI-SpeedGen calls to tire)                                                                                                                                                                                     |
| jobFlag              | int*        | the tire model required task:<br>-1 = no evaluation (skip computations)<br>0 = normal mode<br>1 = inquire (should return arrays dimensions)<br>2 = first initialization<br>3 = re-initialization during simulation<br>4 = solver successful step taken<br>5 = differencing<br>10 = save<br>11 = restore<br>99= model termination (final call) |
| tireID               | int*        | the tire numerical identifier                                                                                                                                                                                                                                                                                                                 |
| simulationTime       | double*     | the current simulation time<br>NOTE: the simulation time is always increasing its value according to simulation time step, regardless of the analysis type (statics, dynamics or else).                                                                                                                                                       |
| wheelCarrierLocation | double* [3] | the wheel carrier location coordinates at the wheel center w.r.t the global reference system                                                                                                                                                                                                                                                  |
| transformationMatrix | double* [9] | the wheel carrier to global reference system transformation matrix                                                                                                                                                                                                                                                                            |
| wheelAngle           | double*     | the rotation angle of the wheel w.r.t the wheel carrier                                                                                                                                                                                                                                                                                       |
| wheelCarrierVelocity | double* [3] | the wheel carrier translational velocities at the wheel center to the global reference system expressed in the wheel carrier reference system                                                                                                                                                                                                 |
| wheelCarrierOmega    | double* [3] | the wheel carrier rotational velocities at the wheel center to the global reference system expressed in the wheel carrier                                                                                                                                                                                                                     |

| VARIABLE NAME       | TYPE           | SPECIFICATIONS                                                                                                                                                                                                                                                                                                              |
|---------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| wheelOmega          | double*        | the rotational velocity of the wheel w.r.t the wheel carrier                                                                                                                                                                                                                                                                |
| diffeqNumber        | int*           | the number of differential equations variables (DEQN)                                                                                                                                                                                                                                                                       |
| diffeqVariables     | double* [DEQN] | the integrated values of the diffeqDerivatives array (initial values are provided by diffeqInitialValues array)                                                                                                                                                                                                             |
| tireParNumber       | int*           | the number of tire parameters (TPN)                                                                                                                                                                                                                                                                                         |
| tireParametersArray | double* [TPN]  | the tire parameters array                                                                                                                                                                                                                                                                                                   |
| tireFileNameLength  | int*           | the tire properties file name string length (TFN)<br>(no file name is supplied if zero)                                                                                                                                                                                                                                     |
| tireFileName        | char* [TFN]    | the tire properties file name string<br>(maybe NULL if no file name is supplied)                                                                                                                                                                                                                                            |
| externalRoad        | void*          | the external road module (function) pointer                                                                                                                                                                                                                                                                                 |
| roadModel           | int*           | the road model type                                                                                                                                                                                                                                                                                                         |
| roadParNumber       | int*           | the number of road parameters (RPN)                                                                                                                                                                                                                                                                                         |
| roadParametersArray | double* [RPN]  | the road parameters array                                                                                                                                                                                                                                                                                                   |
| roadFileNameLength  | int*           | the road properties file name string length (RFN)<br>(no file name is supplied if zero)                                                                                                                                                                                                                                     |
| roadFileName        | char* [RFN]    | the road properties file name string<br>(maybe NULL if no file name is supplied)                                                                                                                                                                                                                                            |
| <b>OUTPUT</b>       |                |                                                                                                                                                                                                                                                                                                                             |
| wheelForces         | double* [3]    | the forces applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system                                                                                                                                                                                                            |
| wheelTorques        | double* [3]    | the torques applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system                                                                                                                                                                                                           |
| diffeqInitialValues | double* [DEQN] | the initial values of the diffeqDerivatives array                                                                                                                                                                                                                                                                           |
| diffeqDerivatives   | double* [DEQN] | the derivatives of diffeqVariables array (to be integrated by the calling solver)                                                                                                                                                                                                                                           |
| tireModel           | char* [256]    | the tire model info string (version, manufacturer, etc.)                                                                                                                                                                                                                                                                    |
| outputValuesNumber  | int*           | the number of output array values (OUTN)                                                                                                                                                                                                                                                                                    |
| outputValuesArray   | double* [OUTN] | the tire model internal variables array. Dynamics (forces, torques) and kinematics (longitudinal and lateral slip, inclination angle) values computed by the tire are stored in this array.<br>The output values strictly required by VI-GRADE post-processing packages are listed in the next paragraph (tire output map). |
| realWorkArraySize   | int*           | the REAL (double) work array size (RWN)                                                                                                                                                                                                                                                                                     |
| realWorkArray       | double* [RWN]  | the tire real storage/work array (for tire model internal use)                                                                                                                                                                                                                                                              |
| intWorkArraySize    | int*           | the INTEGER (int) work array size (IWN)                                                                                                                                                                                                                                                                                     |
| intWorkarray        | int* [IWN]     | the tire integer storage/work array (for tire model internal use)                                                                                                                                                                                                                                                           |
| erroid              | int*           | the error code (0 => no error)<br>NOTE: in case of error the tire model should return error code only. Program STOP/EXIT must NOT be called by the tire module.                                                                                                                                                             |

CRT interface (C/C++ programming language):

```
tire_c_sti_user(//
 // INPUT PARAMETERS
 //

tire_cpp_crt_user(//
 // INPUT PARAMETERS
 //
 int tireID, int simulationMode,
```

## Appendix A: VI-Tire

```

int calcMode, double simulationTime,
double* tireParametersArray, char* tireFileName,
int roadModel, char* roadFileName,
double* wheelCarrierLocation, double* transformationMatrix,
double wheelAngle, double* wheelCarrierVelocity,
double* wheelCarrierOmega, double wheelOmega,
double* diffeqVariables, double frictionScalingFactor,
//
// OUTPUT PARAMETERS
//
double* diffeqDerivatives, double* wheelForces,
double* wheelTorques, double* outputValuesArray
)

```

| VARIABLE NAME         | TYPE           | SPECIFICATIONS                                                                                                                                                                                                                 |
|-----------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INPUT</b>          |                |                                                                                                                                                                                                                                |
| tireID                | int            | the tire numerical identifier                                                                                                                                                                                                  |
| simulationMode        | int            | the solver simulation mode<br>0 = STATICS<br>1 = DYNAMICS                                                                                                                                                                      |
| calcMode              | int            | the tire computation mode<br>0 = initialization/reset<br>1 = standard computations (normal mode)<br>2 = multi-thread computations<br>3 = differencing<br>4 = model termination (final call)                                    |
| simulationTime        | double         | the current simulation time                                                                                                                                                                                                    |
| tireParametersArray   | double* [TPN]  | the tire parameters array (TPN = VT_MAX_PARAMS_NUM)                                                                                                                                                                            |
| tireFileName          | char*          | the tire properties file name string<br>(may be NULL if no file name is supplied)                                                                                                                                              |
| roadModel             | int            | the road model type                                                                                                                                                                                                            |
| roadFileName          | char*          | the road properties file name string<br>(may be NULL if no file name is supplied)                                                                                                                                              |
| wheelCarrierLocation  | double* [3]    | the wheel carrier location coordinates at the wheel center w.r.t<br>the global reference system                                                                                                                                |
| transformationMatrix  | double* [9]    | the wheel carrier to global reference system transformation<br>matrix                                                                                                                                                          |
| wheelAngle            | double         | the rotation angle of the wheel w.r.t the wheel carrier                                                                                                                                                                        |
| wheelCarrierVelocity  | double* [3]    | the wheel carrier translational velocities at the wheel center to<br>the global reference system expressed in the wheel carrier<br>reference system                                                                            |
| wheelCarrierOmega     | double* [3]    | the wheel carrier rotational velocities at the wheel center to the<br>global reference system expressed in the wheel carrier<br>reference system                                                                               |
| wheelOmega            | double         | the rotational velocity of the wheel w.r.t the wheel carrier                                                                                                                                                                   |
| diffeqVariables       | double* [DEQN] | the integrated values of the diffeqDerivatives array<br>(DEQN = VT_MAX_STATES_NUM)                                                                                                                                             |
| frictionScalingfactor | double         | the road parameters array                                                                                                                                                                                                      |
| <b>OUTPUT</b>         |                |                                                                                                                                                                                                                                |
| diffeqDerivatives     | double* [DEQN] | the derivatives of diffeqVariables array (to be integrated by the<br>calling solver) (DEQN = VT_MAX_STATES_NUM)                                                                                                                |
| wheelForces           | double* [3]    | the forces applied by the tire to the wheel at the wheel center,<br>expressed in the wheel carrier reference system                                                                                                            |
| wheelTorques          | double* [3]    | the torques applied by the tire to the wheel at the wheel center,<br>expressed in the wheel carrier reference system                                                                                                           |
| outputValuesArray     | double* [OUTN] | the tire model internal variables array. Dynamics (forces,<br>torques) and kinematics (longitudinal and lateral slip,<br>inclination angle) values computed by the tire are stored in this<br>array (OUTN = VT_MAX_OUTPUT_NUM) |

| VARIABLE NAME | TYPE | SPECIFICATIONS                                                                                                               |
|---------------|------|------------------------------------------------------------------------------------------------------------------------------|
|               |      | The output values strictly required by VI-GRADE post-processing packages are listed in the next paragraph (tire output map). |

## 4.12.2 Tire Utility Functions

The tire user interface allows access to a set of specific utility functions, to compute basic kinematics or access solver data.

C\_ACTCLC2: computes the tire contact patch kinematics (velocities and slip quantities) from wheel carrier dynamic states.

```
c_actclc2(//
 // INPUT PARAMETERS
 //
 double* transformationMatrix, double* wheelCarrierVelocity,
 double* wheelCarrierOmega, double* wheelOmega,
 double* effectiveRadius, double* roadNormal,
 //
 // OUTPUT PARAMETERS
 //
 double* vx, double* vxCP,
 double* vyCP, double* psiOmega,
 double* vzCP, double* lateralSlip,
 double* inclinationAngle, double* longitudinalSlip,
 double* radiusDirection, double* sae2globalMatrix,
 double* vxSign, double* vxFactor);

```

| VARIABLE NAME        | TYPE        | SPECIFICATIONS                                                                                                                                |
|----------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INPUT</b>         |             |                                                                                                                                               |
| transformationMatrix | double* [9] | the wheel carrier to global reference system transformation matrix                                                                            |
| wheelCarrierVelocity | double* [3] | the wheel carrier translational velocities at the wheel center to the global reference system expressed in the wheel carrier reference system |
| wheelCarrierOmega    | double* [3] | the wheel carrier rotational velocities at the wheel center to the global reference system expressed in the wheel carrier reference system    |
| wheelOmega           | double*     | the rotational velocity of the wheel w.r.t the wheel carrier                                                                                  |
| effectiveRadius      | double*     | the tire effective rolling radius                                                                                                             |
| roadNormal           | double* [3] | the road surface normal vector at tire-road contact patch location                                                                            |
| <b>OUTPUT</b>        |             |                                                                                                                                               |
| vx                   | double*     | the longitudinal velocity of the tire w.r.t. the TYDEX-C (wheel carrier) reference system                                                     |
| vxCP                 | double*     | the contact patch longitudinal velocity w.r.t. the TYDEX-W (contact patch) SAE reference system                                               |
| vyCP                 | double*     | the contact patch lateral velocity w.r.t. the TYDEX-W (contact patch) SAE reference system                                                    |
| psiOmega             | double*     | the tire steering angular velocity w.r.t. the TYDEX-W (contact patch) SAE reference system                                                    |
| vzCP                 | double*     | the contact patch vertical velocity w.r.t. the TYDEX-W (contact patch) SAE reference system                                                   |
| lateralSlip          | double*     | the lateral slip angle (SAE)                                                                                                                  |

## Appendix A: VI-Tire

| VARIABLE NAME    | TYPE        | SPECIFICATIONS                                                                                                                      |
|------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| inclinationAngle | double*     | the inclination angle                                                                                                               |
| longitudinalSlip | double*     | the longitudinal slip                                                                                                               |
| radiusDirection  | double* [3] | the tire loaded radius direction unit vector, directed from wheel center to contact patch, expressed in the global reference system |
| sae2globalMatrix | double* [9] | the transformation matrix from the TYDEX-W (contact patch) SAE reference system to the global reference system                      |
| vxSign           | double*     | the tire longitudinal velocity sign                                                                                                 |
| vxfactor         | double*     | the tire longitudinal velocity scaling factor                                                                                       |

C\_ACTFZ: computes the tire vertical force given the tire deflection and stiffness parameters.

```
c_actfz(//
 // INPUT PARAMETERS
 //
 double* vzCP, double* loadedRadius,
 double* verticalStiffness, double* auxiliaryKZ,
 double* verticalDamping, double* unloadedRadius,
 double* kzScaling, double* nominalFZ,
 int* splineID,
 //
 // OUTPUT PARAMETERS
 //
 double* verticalForce
);
```

| VARIABLE NAME     | TYPE    | SPECIFICATIONS                                                                                   |
|-------------------|---------|--------------------------------------------------------------------------------------------------|
| <b>INPUT</b>      |         |                                                                                                  |
| vzCP              | double* | the contact patch vertical velocity w.r.t. the TYDEX-W (contact patch) SAE reference system      |
| loadedRadius      | double* | the tire loaded radius                                                                           |
| verticalStiffness | double* | the tire vertical stiffness                                                                      |
| auxiliaryKZ       | double* | the tire auxiliary/secondary vertical stiffness (usually ZERO). This term is multiplied by DR^2. |
| verticalDamping   | double* | the tire vertical damping                                                                        |
| unloadedRadius    | double* | the tire unloaded (free rolling) radius                                                          |
| kzScaling         | double* | the tire vertical stiffness scaling factor                                                       |
| nominalFZ         | double* | the tire nominal vertical load                                                                   |
| splineID          | int*    | the auxiliary spline identifier (usually ZERO)                                                   |
| <b>OUTPUT</b>     |         |                                                                                                  |
| verticalForce     | double* | the tire vertical force                                                                          |

C\_XCP2HB: transforms the tire contact patch forces and torques to the wheel center (hub). The resultant forces and torques are expressed in the global reference system

```
c_xcp2hb(//
 // INPUT PARAMETERS
 //
 double* forcesCP, double* torquesCP,
 double* radiusVector, double* sae2globalMatrix,
 //
 // OUTPUT PARAMETERS
 //
 double* forcesWC, double* torquesWC
);
```

| VARIABLE NAME    | TYPE        | SPECIFICATIONS                                                                                                       |
|------------------|-------------|----------------------------------------------------------------------------------------------------------------------|
| <b>INPUT</b>     |             |                                                                                                                      |
| forcesCP         | double* [3] | the contact patch tire forces, expressed in the TYDEX-W SAE reference system                                         |
| torquesCP        | double* [3] | the contact patch tire torques, expressed in the TYDEX-W SAE reference system                                        |
| radiusVector     | double* [3] | the tire loaded radius vector, directed from wheel center to contact patch, expressed in the global reference system |
| sae2globalMatrix | double* [9] | the transformation matrix from the TYDEX-W (contact patch) SAE reference system to the global reference system       |
| <b>OUTPUT</b>    |             |                                                                                                                      |
| forcesWC         | double* [3] | the tire forces at wheel center, expressed in the global reference system                                            |
| torquesWC        | double* [3] | the tire torques at wheel center, expressed in the global reference system                                           |

C\_M3T1:multiples transposed 3x3 matrix with 3x1 vector. Note that for historical reasons (Fortran legacy) the matrices are considered column ordered.

This means that C\_M3T1 is actually a direct matrix to vector product for a C (row ordered) matrix.

```
c_m3t1(//
 // OUTPUT PARAMETERS
 //
 double* outputVector,
 //
 // INPUT PARAMETERS
 //
 double* inputMatrix,
 double* inputVector);
```

| VARIABLE NAME | TYPE        | SPECIFICATIONS                           |
|---------------|-------------|------------------------------------------|
| <b>OUTPUT</b> |             |                                          |
| outputVector  | double* [3] | the output (3x1) vector (product result) |
| <b>INPUT</b>  |             |                                          |
| inputMatrix   | double* [9] | the input (3x3) matrix                   |
| inputVector   | double* [3] | the input (3x1) vector                   |

### 4.12.3 Tire-Road Interface

The tire may access the VI-Road solver module by mean of specific calls that respect a single interface type. The road routines have to be explicitly called as function of the road model identifier, being the external road function pointer currently not supported by VI-Tire.

Even if the road interface is common, three different basic road "families" may be addressed. It is suggested to include calls for all the three families (VI\_ROAD, VI\_ROADSINGLE and C\_ARC901).

The VI\_ROAD function is taken as sample, being all parameters the same for all the other road routines.

```
vi_road(//
 // INPUT PARAMETERS
 //
 int* jobFlag, int* roadID,
 double* simulationTime, double* wheelCarrierLocation,
 double* transformationMatrix, int* roadModel,
```

## Appendix A: VI-Tire

```

int* roadParNumber,
int* roadFileNameLength,
int* sectionShapeSize,
double* unloadedRadius,
//
// OUTPUT PARAMETERS
//
int* roadContact, double* effectiveVolume,
double* effectivePenetration, double* roadContactPatch,
double* roadNormal, double* frictionValue,
int* errorID, char* errorMessage
);

```

| VARIABLE NAME        | TYPE          | SPECIFICATIONS                                                                                                                                                                                                                                                                                                   |
|----------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INPUT</b>         |               |                                                                                                                                                                                                                                                                                                                  |
| jobFlag              | int*          | the road model required task:<br>-1 = no evaluation (skip computations)<br>0 = normal mode<br>1 = inquire (should return arrays dimensions)<br>2 = first initialization<br>3 = re-initialization during simulation<br>4 = solver successful step taken<br>5 = differencing<br>99= model termination (final call) |
| roadID               | int*          | the road (or calling tire) numerical identifier                                                                                                                                                                                                                                                                  |
| simulationTime       | double*       | the current simulation time                                                                                                                                                                                                                                                                                      |
| wheelCarrierLocation | double* [3]   | the wheel carrier location coordinates at the wheel center w.r.t<br>the global reference system                                                                                                                                                                                                                  |
| transformationMatrix | double* [9]   | the wheel carrier to global (road) reference system<br>transformation matrix                                                                                                                                                                                                                                     |
| roadModel            | int*          | the road model type                                                                                                                                                                                                                                                                                              |
| roadParNumber        | int*          | the number of road parameters (RPN)                                                                                                                                                                                                                                                                              |
| roadParametersArray  | double* [RPN] | the road parameters array                                                                                                                                                                                                                                                                                        |
| roadFileNameLength   | int*          | the road properties file name string length (RFN)<br>(no file name is supplied if zero)                                                                                                                                                                                                                          |
| roadFileName         | char* [RFN]   | the road properties file name string<br>(maybe NULL if no file name is supplied)                                                                                                                                                                                                                                 |
| sectionShapeSize     | int*          | the tire section shape size (SSN) (*)                                                                                                                                                                                                                                                                            |
| sectionShape         | double* [SSN] | the tire section shape parameters (*)                                                                                                                                                                                                                                                                            |
| unloadedRadius       | double*       | the tire unloaded radius                                                                                                                                                                                                                                                                                         |
| tireWidth            | double*       | the tire width                                                                                                                                                                                                                                                                                                   |
| <b>OUTPUT</b>        |               |                                                                                                                                                                                                                                                                                                                  |
| roadContact          | int*          | the tire-road contact flag (0/1 => no contact/contact)                                                                                                                                                                                                                                                           |
| effectiveVolume      | double*       | the tire-road penetrated volume (*)                                                                                                                                                                                                                                                                              |
| efectivePenetration  | double*       | the tire-road effective penetration (tire deflection)                                                                                                                                                                                                                                                            |
| roadContactPatch     | double* [3]   | the tire-road contact patch location, expressed in the global<br>(road) reference system                                                                                                                                                                                                                         |
| roadNormal           | double* [3]   | the road normal direction cosines at tire-road contact patch,<br>expressed in the global (road) reference system                                                                                                                                                                                                 |
| frictionValue        | double*       | the tire-road friction coefficient at contact patch location                                                                                                                                                                                                                                                     |
| errorID              | int*          | the error code (0 => no error)<br>NOTE: in case of error the road model should return error code<br>only. Program STOP/EXIT must NOT be called by the road<br>module.                                                                                                                                            |
| errorMessage         | char* [256]   | the error message related to the error code                                                                                                                                                                                                                                                                      |

(\*) not used for the time being

## 4.12.4 Tire-Road General Interface

The multi-thread capable user tire models should access the VI-ROAD solver computations using the new interface (roadGEN) provided by the external road function pointer passed to the tire module (externalRoad).

The older VI-ROAD interface API calls (VI\_ROAD, VI\_ROADSINGLE and C\_ARC901) are not guaranteed to be thread-safe and so should be avoided if the tire is supposed to be used for multi-thread simulations.

Features of the VI-ROAD general interface:

1. thread-safety: it is the only interface can be used by user tire to access VI-ROAD tire-road contact during multi-thread simulations.
2. allows multiple point contact computations, up to **50** concurrent points. The computations inputs points can be passed to the interface in a single call and are elaborated much more efficiently than using a loop calling the single point interface.

Limitations of the VI-ROAD general interface:

1. it is supported only by the following road models: FLAT, MESH, EXTERNAL, OPEN-CRG

Description of the interface parameters follows. Please note the most of the definitions are implemented in the `comp_publicdef.h` file provided with user tire samples.

```
roadGEN(//
 // INPUT PARAMETERS
 //
 const int COMPUTATION_MODE
 const int ROAD_ID,
 const int ROAD_DISPATCHER_ID,
 const int SENSOR_ID,
 const int SIMULATION_CALC,
 const int SIMULATION_MODE,
 const double SIMULATION_TIME,
 const int POINTS_NUMBER,
 const int INPUT_FORMAT,
 const double* INPUT_DATA,
 const double* AUX_PARAMETERS,
 const char* ROAD_FILENAME,
 const int OUTPUT_FORMAT,
 //
 // OUTPUT PARAMETERS
 //
 double* outputData,
 double* auxOutput,
 int* errorID
);
```

| VARIABLE NAME      | TYPE | SPECIFICATIONS                                                                                                                                                          |
|--------------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INPUT              |      |                                                                                                                                                                         |
| COMPUTATION_MODE   | int  | the road (interface) computation mode. Currently only ZERO (VCR COMPUTATION_MODE BASIC) is supported.                                                                   |
| ROAD_ID            | int  | the road numerical identifier. It is suggested to pass a ZERO value, to allow automatic setting of the road ID                                                          |
| ROAD_DISPATCHER_ID | int  | the road model dispatcher ID. It should be ZERO to allow VI-ROAD to automatically select the road model                                                                 |
| SENSOR_ID          | int  | the road sensor identifier. A completely independent set (thread) of computations is performed for each sensor. It is a good practice to pass the TIRE identifier here. |
| SIMULATION_CALC    | int  | the solver simulation calc type. This parameter is related to the STI interface job flag.<br>0 (VC_CALC_INIT) = initialization call                                     |

## Appendix A: VI-Tire

| VARIABLE NAME   | TYPE          | SPECIFICATIONS                                                                                                                                                                                                                                               |
|-----------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 |               | 1 (VC_CALC_STANDARD) = standard (normal) computations<br>3 (VC_CALC_DIFF) = differencing computations<br>4 (VC_CALC_TERMINATE) = termination (final) call<br>11(VC_CALC_SAVE) = data save call<br>12(VC_CALC_RESTORE) = data restore call                    |
| SIMULATION_MODE | int           | the solver simulation mode.<br>0 (VC_SIM_STATIC) = statics analysis<br>1 (VC_SIM_DYNAMIC) = dynamics analysis                                                                                                                                                |
| SIMULATION TIME | double        | the solver simulation time                                                                                                                                                                                                                                   |
| POINTS NUMBER   | int           | the number of input/output points to compute                                                                                                                                                                                                                 |
| INPUT_FORMAT    | int           | the input format type. This value defines what input parameters are provided to the road computations.<br>Available format for the current version:<br>0 (VCR_INPUT_BASIC) = minimal input optimized for multiple points contact (MPC) simulations           |
| INPUT DATA      | double* [...] | the road input parameters array. Description is provided below.                                                                                                                                                                                              |
| AUX PARAMETERS  | double* [...] | the the road auxiliary (input) parameters (*)                                                                                                                                                                                                                |
| ROAD FILENAME   | char* [...]   | the road data file name.                                                                                                                                                                                                                                     |
| OUTPUT_FORMAT   | int           | the output format type. This value defines what output values are returned after the road computations.<br>Available output format for the current version:<br>0 (VCR_OUTPUT_BASIC) = minimal output optimized for multiple points contact (MPC) simulations |
| OUTPUT          |               |                                                                                                                                                                                                                                                              |
| outputData      | double* [...] | the road output parameters array. Description is provided below.                                                                                                                                                                                             |
| auxOutput       | double* [...] | the the road auxiliary (output) parameters (*)                                                                                                                                                                                                               |
| errorID         | int*          | the error code. A value of ZERO means there were no errors during computations.                                                                                                                                                                              |

(\*) not used for the time being

NOTE, all the input and output positional data are referred to the road reference system.

The input data array (INPUT\_DATA) size and parameters type depends on the input format type (INPUT\_FORMAT).

#### VCR\_INPUT\_BASIC format

The array is made of sequential data block containing input points Cartesian coordinates (x, y, z values) so the final size of the array will be VCR\_INP\_BSC\_SIZE \* POINTS\_NUMBER.

Using convenience definitions (comp\_publicdef.h):

```
INPUT_DATA[VCR_INP_BSC_SIZE * POINTS_NUMBER] =
{
 X_sp1, Y_sp1, Z_sp1,
 ...
 X_spn, Y_spn, Z_spn
}
```

Where

X, Y, Z = sensor point (sp) input coordinates

The output data array (outputData) size and parameters type depends on the output format type (OUTPUT\_FORMAT).

#### VCR\_OUTPUT\_BASIC format

The basic (minimal) output consists of the road contact patch coordinates and related friction value, for a total of four values each input sensor point. The resulting size of the output array is `VCR_OUT_BSC_SIZE * POINTS_NUMBER`.

Using the provided C defines:

```
outputData[VCR_OUT_BSC_SIZE * POINTS_NUMBER] =
{
 Xcp1, Ycp1, Zcp1, MUcp1,
 ...
 Xcpn, Ycpn, Zcpn, MUcpn
}
```

Where

X, Y, Z = contact point (cp) coordinates.

MU = contact point friction

**WARNING: use always the provided I/O size and channels definitions, as declared in `comp_publicdef.h`. Using definitions will ensure code portability among different version of VI-grade products.**

#### 4.12.5 Tire Output Map

The standard output from the VI-Tire models may vary depending on the model itself. All the native and user-defined tire models, however, should conform to the following output table to have their results extracted and analyzed after a simulation. Note that the output for each tire is a single array with a maximum dimension of `VT_MAX_OUTPUT_NUM` values.

| INDEX | DESCRIPTION                                                                                                                            | UNITS | NOTE     |
|-------|----------------------------------------------------------------------------------------------------------------------------------------|-------|----------|
| 0     | longitudinal force at contact patch (ISO-W reference system)                                                                           | N     | REQUIRED |
| 1     | lateral force at contact patch (ISO-W reference system)                                                                                | N     | REQUIRED |
| 2     | vertical force at contact patch (ISO-W reference system)                                                                               | N     | REQUIRED |
| 3     | X (overturning) torque at contact patch (ISO-W reference system)                                                                       | Nm    | REQUIRED |
| 4     | Y(rolling) torque at contact patch (ISO-W reference system)                                                                            | Nm    | REQUIRED |
| 5     | Z (aligning) torque contact patch (ISO-W reference system)                                                                             | Nm    | REQUIRED |
| 6     | lateral slip angle                                                                                                                     | rad   | REQUIRED |
| 7     | longitudinal slip [-1..1]                                                                                                              | -     | REQUIRED |
| 8     | inclination angle                                                                                                                      | rad   | REQUIRED |
| 9     | turn slip                                                                                                                              | 1/m   | OPTIONAL |
| 10    | wheel carrier longitudinal velocity at the wheel center to the global reference system expressed in the wheel carrier reference system | m/s   | OPTIONAL |
| 11    | wheel carrier lateral velocity at the wheel center to the global reference system expressed in the wheel carrier reference system      | m/s   | OPTIONAL |
| 12    | wheel carrier vertical velocity at the wheel center to the global reference system expressed in the wheel carrier reference system     | m/s   | OPTIONAL |
| 13    | T(1,1) element of the wheel carrier to global reference system transformation matrix                                                   | -     | OPTIONAL |
| 14    | T(1,2) element of the wheel carrier to global reference system transformation matrix                                                   | -     | OPTIONAL |
| 15    | T(1,3) element of the wheel carrier to global reference system transformation matrix                                                   | -     | OPTIONAL |
| 16    | T(2,1) element of the wheel carrier to global reference system transformation matrix                                                   | -     | OPTIONAL |
| 17    | T(2,2) element of the wheel carrier to global reference system transformation matrix                                                   | -     | OPTIONAL |

## Appendix A: VI-Tire

|    |                                                                                                                          |       |          |
|----|--------------------------------------------------------------------------------------------------------------------------|-------|----------|
| 18 | T(2,3) element of the wheel carrier to global reference system transformation matrix                                     | -     | OPTIONAL |
| 19 | T(3,1) element of the wheel carrier to global reference system transformation matrix                                     | -     | OPTIONAL |
| 20 | T(3,2) element of the wheel carrier to global reference system transformation matrix                                     | -     | OPTIONAL |
| 21 | T(3,3) element of the wheel carrier to global reference system transformation matrix                                     | -     | OPTIONAL |
| 22 | wheel carrier X coordinate at the wheel center w.r.t the global reference system                                         | m     | OPTIONAL |
| 23 | wheel carrier Y coordinate at the wheel center w.r.t the global reference system                                         | m     | OPTIONAL |
| 24 | wheel carrier Z coordinate at the wheel center w.r.t the global reference system                                         | m     | OPTIONAL |
| 25 | longitudinal force at contact patch (SAE-W reference system)                                                             | N     | OPTIONAL |
| 26 | lateral force at contact patch (SAE-W reference system)                                                                  | N     | OPTIONAL |
| 27 | vertical force at contact patch (SAE-W reference system)                                                                 | N     | OPTIONAL |
| 28 | X (overturning) torque at contact patch (SAE-W reference system)                                                         | N     | OPTIONAL |
| 29 | Y(rolling) torque at contact patch (SAE-W reference system)                                                              | N     | OPTIONAL |
| 30 | Z (aligning) torque contact patch (SAE-W reference system)                                                               | N     | OPTIONAL |
| 31 | longitudinal force applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system | N     | OPTIONAL |
| 32 | lateral force applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system      | N     | OPTIONAL |
| 33 | vertical force applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system     | N     | OPTIONAL |
| 34 | overturning torque applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system | Nm    | OPTIONAL |
| 35 | rolling torque applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system     | Nm    | OPTIONAL |
| 36 | aligning torque applied by the tire to the wheel at the wheel center, expressed in the wheel carrier reference system    | Nm    | OPTIONAL |
| 37 | longitudinal force at contact patch (ISO-W reference system)                                                             | N     | REQUIRED |
| 38 | lateral force at contact patch (ISO-W reference system)                                                                  | N     | REQUIRED |
| 39 | vertical force at contact patch (ISO-W reference system)                                                                 | N     | REQUIRED |
| 40 | X (overturning) torque at contact patch (ISO-W reference system)                                                         | Nm    | REQUIRED |
| 41 | Y(rolling) torque at contact patch (ISO-W reference system)                                                              | Nm    | REQUIRED |
| 42 | Z (aligning) torque contact patch (ISO-W reference system)                                                               | Nm    | REQUIRED |
| 43 | effective tire penetration (radial deformation)                                                                          | m     | REQUIRED |
| 44 | vertical velocity at contact patch                                                                                       | m/s   | REQUIRED |
| 45 | longitudinal velocity at contact patch                                                                                   | m/s   | REQUIRED |
| 46 | lateral velocity at contact patch                                                                                        | m/s   | REQUIRED |
| 47 | tire longitudinal speed                                                                                                  | m/s   | REQUIRED |
| 48 | tire effective radius                                                                                                    | m     | REQUIRED |
| 49 | rotational velocity of the wheel w.r.t the wheel carrier                                                                 | rad/s | REQUIRED |
| 50 | defined lateral slip (without relaxation effects)                                                                        | rad   | REQUIRED |
| 51 | defined longitudinal slip (without relaxation effects)                                                                   | -     | REQUIRED |
| 52 | state derivative#1 value                                                                                                 | -     | OPTIONAL |
| 53 | state derivative#2 value                                                                                                 | -     | OPTIONAL |
| 54 | actual longitudinal friction                                                                                             | -     | OPTIONAL |
| 55 | actual lateral friction                                                                                                  | -     | OPTIONAL |
| 56 | pneumatic trail                                                                                                          | -     | OPTIONAL |
| 57 | residual torque                                                                                                          | Nm    | OPTIONAL |
| 58 | longitudinal force moment arm                                                                                            | m     | OPTIONAL |
| 59 | longitudinal relaxation length (transient)                                                                               | m     | OPTIONAL |
| 60 | lateral relaxation length (transient)                                                                                    | m     | OPTIONAL |
| 61 | gyroscopic moment (transient)                                                                                            | Nm    | OPTIONAL |
| 62 | Pacejka tire model SVy <sub>k</sub> value                                                                                | -     | OPTIONAL |
| 63 | Pacejka tire model SVx value                                                                                             | -     | OPTIONAL |
| 64 | Pacejka tire model SVy value                                                                                             | -     | OPTIONAL |

|     |                                                                                                  |    |          |
|-----|--------------------------------------------------------------------------------------------------|----|----------|
| 65  | road-tire contact patch X coordinate expressed in the global reference system                    | m  | REQUIRED |
| 66  | road-tire contact patch Y coordinate expressed in the global reference system                    | m  | REQUIRED |
| 67  | road-tire contact patch Z coordinate expressed in the global reference system                    | m  | REQUIRED |
| 68  | road surface normal X direction cosine at contact patch expressed in the global reference system | -  | REQUIRED |
| 69  | road surface normal Y direction cosine at contact patch expressed in the global reference system | -  | REQUIRED |
| 70  | road surface normal Z direction cosine at contact patch expressed in the global reference system | -  | REQUIRED |
| 71  | road surface friction auxiliary output#1                                                         | -  | OPTIONAL |
| 72  | road surface friction auxiliary output#2                                                         | -  | OPTIONAL |
| 73  | road surface friction auxiliary output#3                                                         | -  | OPTIONAL |
| 74  | reserved (Pacejka models)                                                                        | -  | RESERVED |
| 75  | reserved (Pacejka models)                                                                        | -  | RESERVED |
| 76  | reserved (Pacejka models)                                                                        | -  | RESERVED |
| 77  | reserved (Pacejka models)                                                                        | -  | RESERVED |
| 78  | reserved (Pacejka models)                                                                        | -  | RESERVED |
| 79  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 80  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 81  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 82  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 83  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 84  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 85  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 86  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 87  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 88  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 89  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 90  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 91  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 92  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 93  | reserved (VI-Aircraft tire models)                                                               | -  | RESERVED |
| 94  | loaded tire radius                                                                               | m  | REQUIRED |
| 95  | road-tire contact patch auxiliary X coordinate                                                   | m  | OPTIONAL |
| 96  | road-tire contact patch auxiliary Y coordinate                                                   | m  | OPTIONAL |
| 97  | road-tire contact patch auxiliary Z coordinate                                                   | m  | OPTIONAL |
| 98  | tire section profile activation flag                                                             | -  | OPTIONAL |
| 99  | reserved                                                                                         | -  | -        |
| ... | ...                                                                                              | -  | -        |
| 148 | reserved                                                                                         | -  | -        |
| 149 | road-tire contact patch orientation parameter#1                                                  | -  | OPTIONAL |
| 150 | road-tire contact patch orientation parameter#2                                                  | -  | OPTIONAL |
| 151 | road-tire contact patch orientation parameter#3                                                  | -  | OPTIONAL |
| 152 | reserved                                                                                         | -  | -        |
| 153 | reserved                                                                                         | -  | -        |
| 154 | reserved                                                                                         | -  | -        |
| 155 | tire temperature parameter#1                                                                     | C° | OPTIONAL |
| 156 | tire temperature parameter#2                                                                     | C° | OPTIONAL |
| 157 | tire temperature parameter#3                                                                     | C° | OPTIONAL |
| 158 | tire temperature parameter#4                                                                     | C° | OPTIONAL |
| 159 | tire pressure                                                                                    | -  | OPTIONAL |
| 160 | user-defined channel#1 (available for user tire model output)                                    | -  | -        |
| ... | ...                                                                                              | -  | -        |
| 179 | user-defined channel#20 (available for user tire model output)                                   | -  | -        |
| 180 | reserved                                                                                         | -  | -        |
| ... | ...                                                                                              | -  | -        |

## Appendix A: VI-Tire

|     |                                                                             |   |          |
|-----|-----------------------------------------------------------------------------|---|----------|
| 183 | off-the-road contact flag (0/1 => contact ON the road/contact OFF the road) | - | OPTIONAL |
| 184 | road material type identifier                                               | - | OPTIONAL |
| ... | ...                                                                         | - | -        |
| 299 | reserved                                                                    | - | -        |

#### 4.12.6 User Tire Data Save And Restore

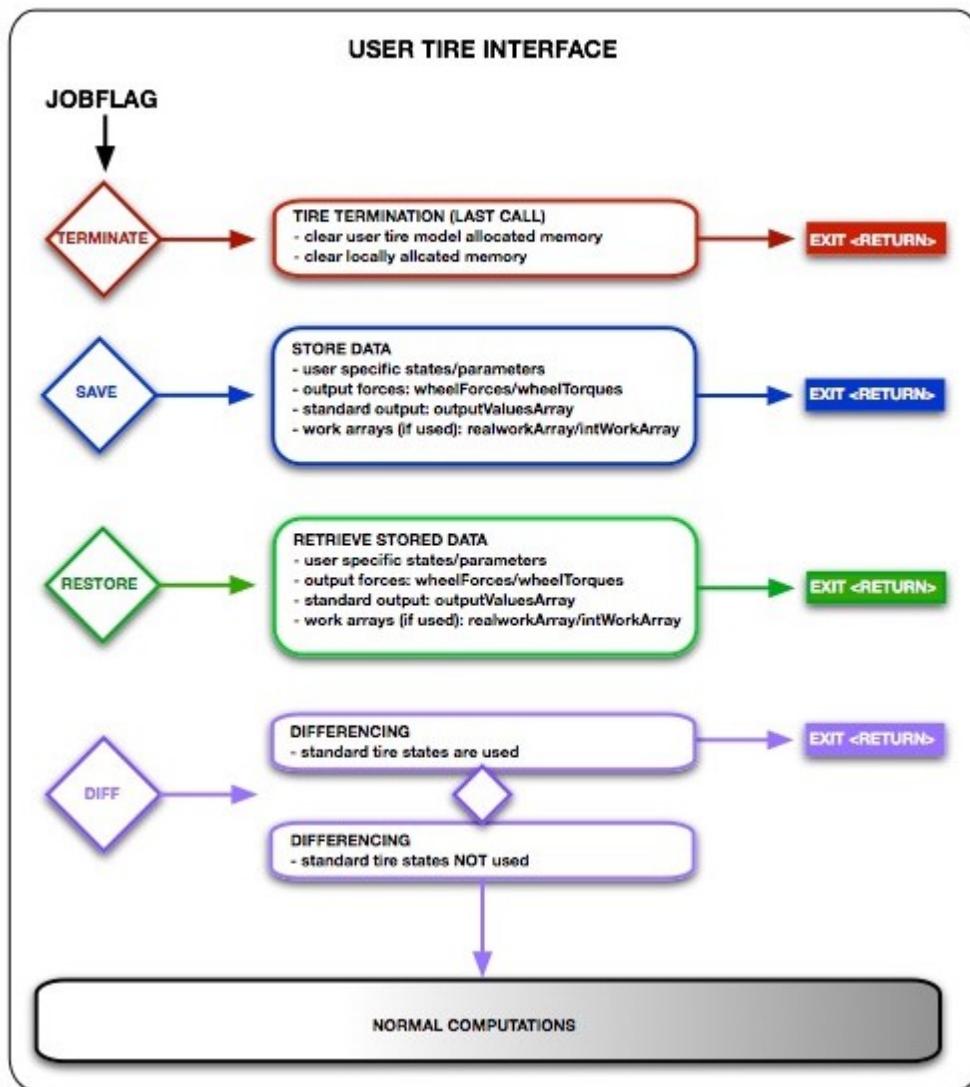
There are some special maneuvers or simulation processes which may issue calls for tire data saving and restoring at certain point in time. While the standard tire I/O is automatically saved/restored by the solver this is not possible for specific states and runtime parameters of a user tire.

The user is responsible of implementing data saving and restoring procedures (if required) following to the special SAVE and RESTORE job flags.

Note that there may be multiple save/restore requests at different point in time, so a key value is required to identify the data. The most commonly used identifier is the simulation time.

The typical user tire job flag management summary follow:

## JOBFLAG MANAGEMENT



### Testing procedure for Save/Restore with VI-CarRealTime

In order to verify a custom tire model is properly supporting the save & restore protocol required by events like MaxPerformance or the execution under the control of VI-DriveSim, the following checking procedure is recommended:

1. run the desired VI-Driver event with the regular version of VI-CarRealTime solver (Course and Stability events excluded). This step can be performed using the interactive mode of the GUI
2. clone the event of point 1 and run it in `files only` mode.
3. run the event created at step 2 using the special solver executable `vicrt_saverestore` stored in the `acarr\win64` subfolder.

```
vicrt_saverestore -f sendfilename
```

During the execution, this alternative application will trigger multiple save and some restore operations.

4. check the consistency of result file produced at step 1 and 3. When this condition is met, the tire model is compliant with the save/restore protocol

## 4.13 Tire/Road Compatibility

The following table describes which tire models are compatible with which road format:

| Tire Model                              | VI-Road Analytic | VI-Road Measured | VI-Road Extrusion | VI-Road Mesh | VI-Road Flat | VI-Road GridMesh | External | Composite | OpenCRG | rgr   |
|-----------------------------------------|------------------|------------------|-------------------|--------------|--------------|------------------|----------|-----------|---------|-------|
| <b>VI-grade Internal Models</b>         | S,R              | S,R              | S,R               | S,M,R        | S,M,R        | S,M,R            | S,M,R    | N/A       | S,M,R   | S,M,R |
| <b>Cosin FTire</b>                      | S                | S                | S                 | S,M          | S,M          | S,M              | S,M      | N/A       | S,M     | S,M,R |
| <b>Siemens MF-TYRE/MF-SWIFT v6.2</b>    | S                | S                | S                 | S            | S            | S                | S        | N/A       | S       | S     |
| <b>Siemens MF-TYRE/MF-SWIFT v2020.1</b> | X                | X                | X                 | S,M          | S,M          | S,M,R            | S,M      | N/A       | S,M,R   | S,M   |
| <b>Fraunhofer CDTire</b>                | X                | X                | X                 | S,M,R        | S,M,R        | S,M,R            | S,M,R    | N/A       | S,M,R   | X     |
| <b>Michelin TameTire 6.1</b>            | X                | X                | X                 | S,M,R        | S,M,R        | X                | S,M,R    | N/A       | X       | S,M,R |
| <b>User Tire (STI)</b>                  | S                | S                | S                 | S,M          | S,M          | S,M              | S,M      | N/A       | S,M     | S,M   |

Key:

S - Single-thread computation capable.

M - Multi-thread computation capable.

R - Real-time capable.

X - Not compatible.

N/A - Not applicable.

NOTE: The composite road format is a wrapper over another referenced road format, so you must look at the compatibility for the underlying road format.

NOTE: The User Tire (STI) real-time capability depends on the complexity of the user tire model.

## 4.14 Bibliography

1. Tyre and Vehicle Dynamics (Second Edition)

Hans B. Pacejka

Copyright 2006 - Published by Elsevier Ltd.

# 5 Appendix B: File Format

## 5.1 AER

- [AER - Standard](#)
- [AER - Advanced](#)

### 5.1.1 AER - Standard Format

The Standard AER file contains aero maps as functions of front and rear ride heights:

- front downforce
- rear downforce
- drag force

The maps resolution and extension is customizable, but it is strongly recommended that the maps are defined for all the ride heights combination allowed by the ride height saturation boundaries. Standard and user defined units could be used to define the maps.

NOTE: the user-defined unit tags are not supported within tables. They are accepted for single value parameters only.

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'aer'
FILE_VERSION = 5
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'Sample Aero Data'
'NOTE: the user-defined unit tags are not supported within tables'
' They are accepted for single value parameters only.'
$-----units
[UNITS]
(BASE)
{length force angle mass time}
'mm' 'N' 'degree' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'm_s' 1 0 0 0 -1 1000.0
'kgpm3' -3 0 0 1 0 1e-9
$-----test_conditions
[TEST_CONDITIONS]
```

## Appendix B: File Format

```

reference_velocity <kmh> = 100.0
air_density <kgpm3> = 1.0 $ [kg/m^3]
reference_density = 1.0

front_ride_height_min = 10.0
front_ride_height_max = 160.0
rear_ride_height_min = 10.0
rear_ride_height_max = 160.0
drag_arm_height_min = -50.0
drag_arm_height_max = 50.0

$-----front_downforce

[FRONT_DOWNFORCE]
(Z_DATA)
{rear_ride_height }
10.0
40.0
80.0
120.0
140.0
160.0

(XY_DATA)
{front_ride_height downforce }
10.0 400.0 400.0 400.0 400.0 400.0 400.0
40.0 400.0 400.0 400.0 400.0 400.0 400.0
80.0 400.0 400.0 400.0 400.0 400.0 400.0
120.0 400.0 400.0 400.0 400.0 400.0 400.0
140.0 400.0 400.0 400.0 400.0 400.0 400.0
160.0 400.0 400.0 400.0 400.0 400.0 400.0

$-----rear_downforce

[REAR_DOWNFORCE]
(Z_DATA)
{rear_ride_height }
10.0
40.0
80.0
120.0
140.0
160.0

(XY_DATA)
{front_ride_height downforce }
10.0 500.0 500.0 500.0 500.0 500.0 500.0
40.0 500.0 500.0 500.0 500.0 500.0 500.0
80.0 500.0 500.0 500.0 500.0 500.0 500.0
120.0 500.0 500.0 500.0 500.0 500.0 500.0
140.0 500.0 500.0 500.0 500.0 500.0 500.0
160.0 500.0 500.0 500.0 500.0 500.0 500.0

$-----drag

[DRAG]
(Z_DATA)
{rear_ride_height }
10.0
40.0
80.0
120.0
140.0
160.0

(XY_DATA)
{front_ride_height drag }
10.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0
40.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0
80.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0
120.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0

```

```
140.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0
160.0 1200.0 1200.0 1200.0 1200.0 1200.0 1200.0
```

## 5.1.2 AER - Advanced Format v1.0

This file format is now deprecated. See the topic [Convert from v1.0 to v2.0](#) to learn how to convert custom aer v1.0 files to the [new file format v2.0](#)

The Advanced AER file contains:

- aero coefficients as functions of front and rear ride heights
- 3-direction aerodynamics force corrections: these corrections are implemented for front and rear downforce as well as drag.

The maps resolution and extension is customizable, but it is strongly recommended that the maps are defined for all the ride heights combination allowed by the ride height saturation boundaries. Standard and user defined units could be used to define the maps.

NOTE: the user-defined unit tags are not supported within tables. They are accepted for single value parameters only.

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'aer'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'Sample Aerodynamic Data'
'NOTE: the user-defined unit tags are not supported within tables'
' They are accepted for single value parameters only.'
'SportsCar aerodynamic forces update for 6DOF:'
' [1] added new sfosub (vsc_admaerosfo) to handle new aero data file'
' [2] added new dependency from yaw, pitch, roll, and steering angles
' by means of 16 new (2D) splines'
$-----UNITS
[UNITS]
(BASE)
{length force angle mass time}
'mm' 'newton' 'degrees' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'KPH' 1 0 0 0 -1 2.777778E+02
'm2' 2 0 0 0 0 1.0E+06
'kpm3' -3 0 0 1 0 1.0E-09
$ 1 KPH = 1e6 mm / 3600 sec = 2.777778E+02 mm/sec
$ 1 m2 = 1e6 mm^2
$ 1 kpm3 = 1e-9 kg/mm^3
$-----GENERAL_PARAMETERS
[GENERAL_PARAMETERS]
AIR_DENSITY <kpm3> = 1.22 $ [kg/m^3]
AERO_FRONTAL_AREA <m2> = 1.5 $ [m^2]
FRONT_RIDE_HEIGHT_MIN = 0.00
FRONT_RIDE_HEIGHT_MAX = 30.00
REAR_RIDE_HEIGHT_MIN = 10.00
REAR_RIDE_HEIGHT_MAX = 70.00
$-----OFFSET_PARAMETERS
[OFFSET_PARAMETERS]
CZ = -0.2 $ [Per REFERENCE_VELOCITY_OF_OFFSET]
BALANCE = 0.0 $ [Per REFERENCE_VELOCITY_OF_OFFSET]
```

## Appendix B: File Format

```

CX_BODY = 0.04 $ [Per REFERENCE_VELOCITY_OF_OFFSET]
CX_FRONT = 0.0 $ [Per REFERENCE_VELOCITY_OF_OFFSET]
CX_REAR = 0.0 $ [Per REFERENCE_VELOCITY_OF_OFFSET]
$-----CZ
[Z]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
50.0
60.0
70.0
(XY_DATA)
{front_ride_height cz_coeff}
0.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
5.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
10.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
15.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
20.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
25.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
30.00 2.500 2.500 2.500 2.500 2.500 2.500 2.500
$-----BALANCE
[BALANCE]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
50.0
60.0
70.0
(XY_DATA)
{front_ride_height balance_coeff}
0.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
5.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
10.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
15.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
20.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
25.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
30.00 45.000 45.000 45.000 45.000 45.000 45.000 45.000
$-----CX_BODY
[CX_BODY]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
50.0
60.0
70.0
(XY_DATA)
{front_ride_height cx_body_coeff}
0.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500
5.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500
10.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500
15.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500
20.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500

```

```

25.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500 0.500
30.00 0.500 0.500 0.500 0.500 0.500 0.500 0.500 0.500
$-----CX_FRONT
[CX_FRONT]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
50.0
60.0
70.0
(XY_DATA)
{front_ride_height cx_front_coeff}
 0.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 5.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 10.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 15.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 20.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 25.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
 30.00 0.250 0.250 0.250 0.250 0.250 0.250 0.250
$-----CX_REAR
[CX_REAR]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
50.0
60.0
70.0
(XY_DATA)
{front_ride_height cx_rear_coeff}
 0.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 5.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 10.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 15.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 20.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 25.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
 30.00 0.300 0.300 0.300 0.300 0.300 0.300 0.300
$-----CB_YAW
[CB_YAW]
{ yaw_angle cbody_effect }
 -5.0 1.00
 -2.0 1.00
 -1.0 1.00
 0.0 1.00
 1.0 1.00
 2.0 1.00
 5.0 1.00
$-----CB_PITCH
[CB_PITCH]
{ pitch_angle cbody_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00

```

## Appendix B: File Format

```
30.0 1.00
$-----CB_ROLL
[CB_ROLL]
{ roll_angle cbody_effect }
-2.0 1.00
-1.0 1.00
0.0 1.00
1.0 1.00
2.0 1.00
$-----CB_STEERING
[CB_STEERING]
{ steering_angle cbody_effect }
-10.0 1.00
-5.0 1.00
-2.0 1.00
0.0 1.00
2.0 1.00
5.0 1.00
10.0 1.00
$-----CF_YAW
[CF_YAW]
{ yaw_angle cfront_effect }
-5.0 1.00
-2.0 1.00
-1.0 1.00
0.0 1.00
1.0 1.00
2.0 1.00
5.0 1.00
$-----CF_PITCH
[CF_PITCH]
{ pitch_angle cfront_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
0.0 1.00
10.0 1.00
20.0 1.00
30.0 1.00
$-----CF_ROLL
[CF_ROLL]
{ roll_angle cfront_effect }
-2.0 1.00
-1.0 1.00
0.0 1.00
1.0 1.00
2.0 1.00
$-----CF_STEERING
[CF_STEERING]
{ steering_angle cfront_effect }
-10.0 1.00
-5.0 1.00
-2.0 1.00
0.0 1.00
2.0 1.00
5.0 1.00
10.0 1.00
$-----CR_YAW
[CR_YAW]
{ yaw_angle crear_effect }
-5.0 1.00
```

```
-2.0 1.00
-1.0 1.00
 0.0 1.00
 1.0 1.00
 2.0 1.00
 5.0 1.00
$-----CR_PITCH
[CR_PITCH]
{ pitch_angle cdrag_effect }
 -30.0 1.00
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CR_ROLL
[CR_ROLL]
{ roll_angle cdrag_effect }
 -2.0 1.00
-1.0 1.00
 0.0 1.00
 1.0 1.00
 2.0 1.00
$-----CR_STEERING
[CR_STEERING]
{ steering_angle cdrag_effect }
 -10.0 1.00
 -5.0 1.00
 -2.0 1.00
 0.0 1.00
 2.0 1.00
 5.0 1.00
 10.0 1.00
$-----CD_YAW
[CD_YAW]
{ yaw_angle cddrag_effect }
 -5.0 1.00
-2.0 1.00
-1.0 1.00
 0.0 1.00
 1.0 1.00
 2.0 1.00
 5.0 1.00
$-----CD_PITCH
[CD_PITCH]
{ pitch_angle cddrag_effect }
 -30.0 1.00
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CD_ROLL
[CD_ROLL]
{ roll_angle cddrag_effect }
 -2.0 1.00
-1.0 1.00
 0.0 1.00
 1.0 1.00
```

## Appendix B: File Format

```

2.0 1.00
$-----CD_STEERING
[CD_STEERING]
{ steering_angle cddrag_effect }
-10.0 1.00
-5.0 1.00
-2.0 1.00
0.0 1.00
2.0 1.00
5.0 1.00
10.0 1.00

```

### 5.1.3 AER - Advanced Format v2.0

The aerodynamic forces include:

- drag force
- front and rear downforce
- front and rear side force
- rolling torque
- pitch torque

[Downforces parameters](#)

[Drag force parameters](#)

[Side forces parameters](#)

[Roll torque parameters](#)

See the [Aerodynamic Forces](#) chapter to learn how the parameters set in the .aer property file are used to calculate the aerodynamic forces.

NOTE: the user-defined unit tags are not supported within tables. They are accepted for single value parameters only.

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'aer'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'VI-grade advanced aerodynamics sample data file'
'NOTE: the user-defined unit tags are not supported within tables'
' They are accepted for single value parameters only.'
$-----UNITS
[UNITS]
(BASE)
{length force angle mass time }
'mm' 'newton' 'degree' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'mps' 1 0 0 0 -1 1e3
'kgpm3' -3 0 0 1 0 1e-9
'kmh' 1 0 0 0 -1 277.78
$-----MODEL
[MODEL]
PROPERTY_FILE_FORMAT = 'ADVANCED'
$-----GENERAL_PARAMETERS
[GENERAL_PARAMETERS]
AIR_DENSITY <kgpm3> = 1.22 $[kg/m^3]
AERO_FRONTAL_AREA = 1.0e6
FRONT_RIDE_HEIGHT_MIN = 0.0
FRONT_RIDE_HEIGHT_MAX = 120.0

```

```

REAR_RIDE_HEIGHT_MIN = 0.0
REAR_RIDE_HEIGHT_MAX = 140.0
DRAG_ARM_HEIGHT_MIN = -1.0e3
DRAG_ARM_HEIGHT_MAX = 1.0e3
$-----OFFSET_PARAMETERS
[OFFSET_PARAMETERS]
VELOCITY_THRESHOLD <kmh> = 0.0 $[km/h]
REFERENCE_VELOCITY_OF_OFFSET <kmh> = 0.0 $[km/h]
CZ = 0.0 $[per REFERENCE_VELOCITY_OF_OFFSET]
BALANCE_Z = 0.0 $[per REFERENCE_VELOCITY_OF_OFFSET]
CX_BODY = 0.0 $[per REFERENCE_VELOCITY_OF_OFFSET]
CX_FRONT = 0.0 $[per REFERENCE_VELOCITY_OF_OFFSET]
CX_REAR = 0.0 $[per REFERENCE_VELOCITY_OF_OFFSET]
$
$ downforce coefficients and modifiers
$-----CZ
[CZ]
(Z_DATA)
{rear_ride_height}
0.0
20.0
40.0
60.0
80.0
100.0
120.0
140.0
(XY_DATA)
{ front_ride_height cz_coeff }
0.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
20.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
40.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
60.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
80.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
100.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
120.0 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639 1.901639
$-----BALANCE_Z
[BALANCE_Z]
(Z_DATA)
{rear_ride_height}
0.0
20.0
40.0
60.0
80.0
100.0
120.0
140.0
(XY_DATA)
{ front_ride_height balance_coeff }
0.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
20.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
40.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
60.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
80.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
100.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
120.0 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793 41.3793
$-----CZB_YAW
[CZB_YAW]
{ yaw_angle cbody_effect }
-5.0 1.0

```

## Appendix B: File Format

```
-2.5 1.0
 0.0 1.0
 2.5 1.0
 5.0 1.0
$-----CZB_PITCH
[CZB_PITCH]
{ pitch_angle cbody_effect }
 -30.0 1.0
 -20.0 1.0
 -10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CZB_ROLL
[CZB_ROLL]
{ roll_angle cbody_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CZB_STEERING
[CZB_STEERING]
{ steering_angle cbody_effect }
 -50.0 1.0
 -20.0 1.0
 0.0 1.0
 20.0 1.0
 50.0 1.0
$-----CZF_YAW
[CZF_YAW]
{ yaw_angle cfront_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 1.0
 2.5 1.0
 5.0 1.0
$-----CZF_PITCH
[CZF_PITCH]
{ pitch_angle cfront_effect }
 -30.0 1.0
 -20.0 1.0
 -10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CZF_ROLL
[CZF_ROLL]
{ roll_angle cfront_effect }
 -20.0 1.0
 -10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
$-----CZF_STEERING
[CZF_STEERING]
{ steering_angle cfront_effect }
 -10.0 1.0
 -5.0 1.0
 0.0 1.0
 5.0 1.0
```

```

10.0 1.0
$-----CZR_YAW
[CZR_YAW]
{ yaw_angle crear_effect }
-5.0 1.0
-2.5 1.0
0.0 1.0
2.5 1.0
5.0 1.0
$-----CZR_PITCH
[CZR_PITCH]
{ pitch_angle crear_effect }
-30.0 1.0
-20.0 1.0
-10.0 1.0
0.0 1.0
10.0 1.0
20.0 1.0
30.0 1.0
$-----CZR_ROLL
[CZR_ROLL]
{ roll_angle crear_effect }
-20.0 1.00
-10.0 1.00
0.0 1.00
10.0 1.00
20.0 1.00
$-----CZR_STEERING
[CZR_STEERING]
{ steering_angle crear_effect }
-50.0 1.00
-20.0 1.00
0.0 1.00
20.0 1.00
50.0 1.00
$
$ dragforce coefficients and modifiers
$-----CX_BODY
[CX_BODY]
(Z_DATA)
{rear_ride_height}
0.0
20.0
40.0
60.0
80.0
100.0
120.0
140.0
(XY_DATA)
{ front_ride_height cx_body_coeff }
0.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
20.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
40.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
60.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
80.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
100.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
120.0 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475 1.311475
$-----CX_FRONT
[CX_FRONT]
(Z_DATA)

```

## Appendix B: File Format

```
{rear_ride_height}
0.0
20.0
40.0
60.0
80.0
100.0
120.0
140.0
(XY_DATA)
{ front_ride_height cx_front_coeff }
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
20.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
40.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
60.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
80.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
100.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
120.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
$-----CX_REAR
[CX_REAR]
(Z_DATA)
{ rear_ride_height }
0.0
20.0
40.0
60.0
80.0
100.0
120.0
140.0
(XY_DATA)
{ front_ride_height cx_front_coeff }
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
20.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
40.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
60.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
80.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
100.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
120.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
$-----CX_YAW
[CX_YAW]
{ yaw_angle cdrag_effect }
-5.0 1.00
-2.5 1.00
0.0 1.00
2.5 1.00
5.0 1.00
$-----CX_PITCH
[CX_PITCH]
{ pitch_angle cdrag_effect }
-30.0 1.0
-20.0 1.0
-10.0 1.0
0.0 1.0
10.0 1.0
20.0 1.0
30.0 1.0
$-----CX_ROLL
[CX_ROLL]
{ roll_angle cdrag_effect }
-20.0 1.00
```

```
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CX_STEERING
[CX_STEERING]
{ steering_angle cdrag_effect }
 -50.0 1.0
 -20.0 1.0
 0.0 1.0
 20.0 1.0
 50.0 1.0
$
$ sideforce coefficients and modifiers
$-----CY
[CY]
(Z_DATA)
{rear_ride_height}
0.0
40.0
80.0
140.0
(XY_DATA)
{ front_ride_height cy_coeff }
0.0 0.2 0.2 0.2 0.2
40.0 0.2 0.2 0.2 0.2
80.0 0.2 0.2 0.2 0.2
120.0 0.2 0.2 0.2 0.2
$-----BALANCE_Y
[BALANCE_Y]
(Z_DATA)
{rear_ride_height}
0.0
40.0
80.0
140.0
(XY_DATA)
{ front_ride_height balance_coeff }
0.0 30.0 30.0 30.0 30.0
40.0 30.0 30.0 30.0 30.0
80.0 30.0 30.0 30.0 30.0
120.0 30.0 30.0 30.0 30.0
$-----CYB_YAW
[CYB_YAW]
{ yaw_angle cbody_effect }
 -5.0 1.00
 -2.5 1.00
 0.0 0.00
 2.5 -1.00
 5.0 -1.00
$-----CYB_PITCH
[CYB_PITCH]
{ pitch_angle cbody_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CYB_ROLL
```

## Appendix B: File Format

```
[CYB_ROLL]
{ roll_angle cbody_effect }
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CYB_STEERING
[CYB_STEERING]
{ steering_angle cbody_effect }
 -50.0 1.00
 -20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----CYF_YAW
[CYF_YAW]
{ yaw_angle cfront_effect }
 -5.0 1.00
 -2.5 1.00
 0.0 1.00
 2.5 1.00
 5.0 1.00
$-----CYF_PITCH
[CYF_PITCH]
{ pitch_angle cfront_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CYF_ROLL
[CYF_ROLL]
{ roll_angle cfront_effect }
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CYF_STEERING
[CYF_STEERING]
{ steering_angle cfront_effect }
 -50.0 1.00
 -20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----CYR_YAW
[CYR_YAW]
{ yaw_angle crear_effect }
 -5.0 1.00
 -2.5 1.00
 0.0 1.00
 2.5 1.00
 5.0 1.00
$-----CYR_PITCH
[CYR_PITCH]
{ pitch_angle crear_effect }
 -30.0 1.00
```

```

-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CYR_ROLL
[CYR_ROLL]
{ roll_angle cbody_effect }
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CYR_STEERING
[CYR_STEERING]
{ steering_angle cbody_effect }
-50.0 1.00
-20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----CMX
[CMX]
(Z_DATA)
{rear_ride_height}
0.0
40.0
80.0
140.0
(XY_DATA)
{ front_ride_height cmx_coeff }
0.0 0.0 0.0 0.0 0.0
40.0 0.0 0.0 0.0 0.0
80.0 0.0 0.0 0.0 0.0
120.0 0.0 0.0 0.0 0.0
$-----CMXB_YAW
[CMXB_YAW]
{ yaw_angle cbody_effect }
-5.0 1.00
-2.5 1.00
 0.0 0.00
 2.5 -1.00
 5.0 -1.00
$-----CMXB_PITCH
[CMXB_PITCH]
{ pitch_angle cbody_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CMXB_ROLL
[CMXB_ROLL]
{ roll_angle cbody_effect }
-20.0 1.00
-10.0 1.00

```

## Appendix B: File Format

```
0.0 1.00
10.0 1.00
20.0 1.00
$-----CMXB_STEERING
[CMXB_STEERING]
{ steering_angle cbody_effect }
-50.0 1.00
-20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----CMXF_YAW
[CMXF_YAW]
{ yaw_angle cfront_effect }
-5.0 1.00
-2.5 1.00
 0.0 1.00
 2.5 1.00
 5.0 1.00
$-----CMXF_PITCH
[CMXF_PITCH]
{ pitch_angle cfront_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CMXF_ROLL
[CMXF_ROLL]
{ roll_angle cfront_effect }
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CMXF_STEERING
[CMXF_STEERING]
{ steering_angle cfront_effect }
-50.0 1.00
-20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----CMXR_YAW
[CMXR_YAW]
{ yaw_angle crear_effect }
-5.0 1.00
-2.5 1.00
 0.0 1.00
 2.5 1.00
 5.0 1.00
$-----CMXR_PITCH
[CMXR_PITCH]
{ pitch_angle crear_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
```

```

20.0 1.00
30.0 1.00
$-----CMXR_ROLL
[CMXR_ROLL]
{ roll_angle crear_effect }
-20.0 1.00
-10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
$-----CMXR_STEERING
[CMXR_STEERING]
{ steering_angle crear_effect }
-50.0 1.00
-20.0 1.00
 0.0 1.00
 20.0 1.00
 50.0 1.00
$-----WIND_EFFECT
[WIND_EFFECT]
$ Direction w.r.t. X axis (angle units)
DIRECTION = 180.0
$ Wind velocity
SPEED = 0

```

## 5.1.4 Converting from v1.0 to v2.0

The following is the new aer format file v2.0.

The new parts will be written in **blue**, the changed ones in **red**.

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'aer'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'VI-grade advanced aerodynamics sample data file'
'NOTE: the user-defined unit tags are not supported within tables'
' They are accepted for single value parameters only.'
$-----UNITS
[UNITS]
(BASE)
{length force angle mass time }
'mm' 'newton' 'degrees' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'KPH' 1 0 0 0 -1 2.777778E+02
'm2' 2 0 0 0 0 1.0E+06
'kpm3' -3 0 0 1 0 1.0E-09
$ 1 KPH = 1e6 mm / 3600 sec = 2.777778E+02 mm/sec
$ 1 m2 = 1e6 mm^2
$ 1 kpm3 = 1e-9 kg/mm^3
$-----MODEL
[MODEL]
 PROPERTY_FILE_FORMAT = 'ADVANCED'
$-----GENERAL_PARAMETERS
[GENERAL_PARAMETERS]
AIR_DENSITY <kpm3> = 1.22 $[kg/m^3]
AERO_FRONTAL_AREA <m2> = 1.0 $[m^2]
FRONT_RIDE_HEIGHT_MIN = 0.0
FRONT_RIDE_HEIGHT_MAX = 40.0

```

## Appendix B: File Format

```

REAR_RIDE_HEIGHT_MIN = 0.0
REAR_RIDE_HEIGHT_MAX = 40.0
DRAG_ARM_HEIGHT_MIN = 0.0
DRAG_ARM_HEIGHT_MAX = 1000.0
$-----OFFSET_PARAMETERS
[OFFSET_PARAMETERS]
CZ = 0.0 ${per REFERENCE_VELOCITY_OF_OFFSET}
BALANCE_Z = 0.0 ${per REFERENCE_VELOCITY_OF_OFFSET} (was named BALANCE in the old v1.0
file)
CX_BODY = 0.0 ${per REFERENCE_VELOCITY_OF_OFFSET}
CX_FRONT = 0.0 ${per REFERENCE_VELOCITY_OF_OFFSET}
CX_REAR = 0.0 ${per REFERENCE_VELOCITY_OF_OFFSET}
$
$ downforce coefficients and modifiers
$-----CZ
[CZ]
(Z_DATA)
{rear_ride_height}
13.0
16.0
20.0
22.0
25.0
28.0
32.0
35.0
38.0
(XY_DATA)
{ front_ride_height cz_coeff }
13.0 2.65 2.65 2.65 2.67 2.67 2.68 2.70 2.70 2.72
16.0 2.58 2.59 2.60 2.61 2.63 2.64 2.64 2.65 2.65
20.0 2.52 2.53 2.54 2.56 2.57 2.58 2.58 2.61 2.61
22.0 2.50 2.50 2.49 2.50 2.52 2.52 2.54 2.55 2.54
25.0 2.40 2.42 2.42 2.45 2.45 2.46 2.48 2.48 2.49
28.0 2.35 2.40 2.40 2.38 2.40 2.40 2.42 2.42 2.44
32.0 2.29 2.30 2.30 2.33 2.34 2.35 2.35 2.37 2.38
35.0 2.23 2.25 2.26 2.27 2.28 2.29 2.30 2.32 2.31
38.0 2.18 2.20 2.20 2.22 2.23 2.24 2.24 2.25 2.27
$-----BALANCE
[BALANCE_Z] (was named BALANCE in the old v1.0 file)
(Z_DATA)
{rear_ride_height}
13.0
16.0
20.0
22.0
25.0
28.0
32.0
35.0
38.0
(XY_DATA)
{ front_ride_height balance_coeff }
13.0 35.6 37.0 37.2 38.0 38.5 34.0 39.5 40.0 40.5
16.0 34.8 36.0 36.1 37.0 37.5 38.0 38.8 39.0 40.0
20.0 34.0 35.0 35.0 36.0 36.8 37.0 38.0 38.5 39.0
22.0 33.1 34.0 35.0 35.0 36.0 36.5 37.0 37.5 38.0
25.0 32.3 33.0 34.0 34.0 35.0 35.5 36.0 37.0 37.0
28.0 31.4 32.0 33.0 33.0 34.0 34.7 35.0 35.8 36.5
32.0 30.2 31.0 32.0 32.0 33.0 33.8 34.0 34.8 35.5
35.0 29.3 30.0 31.0 31.0 32.0 32.8 33.3 34.0 34.5
38.0 28.1 29.0 30.0 30.0 31.0 31.5 32.2 33.0 33.5
$
$-----CZB_YAW
[CZB_YAW] (supersedes CB_YAW in the old v1.0 file)

```

```

{ yaw_angle cbody_effect }
 -5.0 0.99
 -2.5 1.0
 0.0 1.1
 2.5 1.0
 5.0 0.99
$-----CZB_PITCH

[CZB_PITCH] (supersedes CB_PITCH in the old v1.0 file)
{ pitch_angle cbody_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CZB_ROLL

[CZB_ROLL] (supersedes CB_ROLL in the old v1.0 file)
{ roll_angle cbody_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CZB_STEERING

[CZB_STEERING] (supersedes CB_STEERING in the old v1.0 file)
{ steering_angle cbody_effect }
 -10.0 1.00
 -5.0 1.00
 0.0 1.00
 5.0 1.00
 10.0 1.00
$-----CZF_YAW

[CZF_YAW] (supersedes CF_YAW in the old v1.0 file)
{ yaw_angle cfront_effect }
 -5.0 1.20
 -2.5 1.05
 0.0 1.00
 2.5 1.05
 5.0 1.20
$-----CZF_PITCH

[CZF_PITCH] (supersedes CF_PITCH in the old v1.0 file)
{ pitch_angle cfront_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CZF_ROLL

[CZF_ROLL] (supersedes CF_ROLL in the old v1.0 file)
{ roll_angle cfront_effect }
 -2.0 1.01
 0.0 1.0
 2.0 1.01
$-----CZF_STEERING

[CZF_STEERING] (supersedes CF_STEERING in the old v1.0 file)
{ steering_angle cfront_effect }
 -10.0 1.00
 -5.0 1.00
 0.0 1.00
 5.0 1.00
 10.0 1.00
$-----CZF_STEERING

```

## Appendix B: File Format

```

$-----CZR_YAW
[CZR_YAW] (supersedes CR_YAW in the old v1.0 file)
{ yaw_angle creak_effect }
-5.0 1.05
-2.5 1.03
0.0 1.00
2.5 1.03
5.0 1.05
$-----CZR_PITCH
[CZR_PITCH] (supersedes CR_PITCH in the old v1.0 file)
{ pitch_angle creak_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
0.0 1.00
10.0 1.00
20.0 1.00
30.0 1.00
$-----CZR_ROLL
[CZR_ROLL] (supersedes CR_ROLL in the old v1.0 file)
{ roll_angle creak_effect }
-2.0 1.01
0.0 1
2.0 1.01
$-----CZR_STEERING
[CZR_STEERING] (supersedes CR_STEERING in the old v1.0 file)
{ steering_angle creak_effect }
-10.0 1.00
-5.0 1.00
0.0 1.00
5.0 1.00
10.0 1.00
$ dragforce coefficients and modifiers
$-----CX_BODY
[CX_BODY]
(Z_DATA)
{rear_ride_height}
13.0
16.0
20.0
22.0
25.0
28.0
32.0
35.0
38.0
(XY_DATA)
{ front_ride_height cx_body_coeff }
13.0 1.02 1.02 1.02 1.02 1.03 1.03 1.03 1.03 1.04
16.0 1.02 1.02 1.02 1.02 1.02 1.02 1.03 1.03 1.03
20.0 1.01 1.02 1.02 1.02 1.02 1.02 1.02 1.03 1.03
22.0 1.02 1.01 1.01 1.01 1.02 1.02 1.02 1.02 1.03
25.0 1.01 1.00 1.01 1.01 1.01 1.02 1.02 1.02 1.02
28.0 1.00 1.00 1.01 1.01 1.01 1.01 1.01 1.02 1.02
32.0 1.00 1.00 1.00 1.00 1.01 1.01 1.01 1.01 1.02
35.0 0.99 1.00 1.00 1.00 1.00 1.00 1.01 1.01 1.01
38.0 0.98 0.99 0.99 1.00 1.00 1.00 1.00 1.01 1.01
$-----CX_FRONT
[CX_FRONT]
(Z_DATA)
{rear_ride_height}
13.0
16.0
20.0

```

```

22.0
25.0
28.0
32.0
35.0
38.0
(XY_DATA)
{ front_ride_height cx_front_coeff }
13.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
16.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
20.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
22.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
25.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
28.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
32.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
35.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
38.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
$-----CX_REAR

[CX_REAR]
(Z_DATA)
{ rear_ride_height }
13.0
16.0
20.0
22.0
25.0
28.0
32.0
35.0
38.0
(XY_DATA)
{ front_ride_height cx_front_coeff }
13.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
16.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
20.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
22.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
25.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
28.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
32.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
35.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
38.0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
$-----CX_YAW

[CX_YAW] (supersedes CD_YAW in the old v1.0 file)
{ yaw_angle cddrag_effect }
-5.0 0.98
-2.5 1.0
0.0 1.01
2.5 1.0
5.0 0.98
$-----CX_PITCH

[CX_PITCH] (supersedes CD_PITCH in the old v1.0 file)
{ pitch_angle cddrag_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
0.0 1.00
10.0 1.00
20.0 1.00
30.0 1.00
$-----CX_ROLL

[CX_ROLL] (supersedes CD_ROLL in the old v1.0 file)
{ roll_angle cddrag_effect }
-2.0 0.98
0.0 1

```

## Appendix B: File Format

```
2.0 1.02
$-----CX_STEERING
[CX_STEERING] (supersedes CD_STEERING in the old v1.0 file)
{ steering_angle cdrag_effect }
-10.0 1.00
-5.0 1.00
0.0 1.00
5.0 1.00
10.0 1.00
$
$ sideforce coefficients and modifiers
$-----CY
[CY]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
(XY_DATA)
{ front_ride_height cy_coeff }
10.0 0.2 0.2 0.2 0.2
20.0 0.2 0.2 0.2 0.2
30.0 0.2 0.2 0.2 0.2
40.0 0.2 0.2 0.2 0.2
$-----BALANCE_Y
[BALANCE_Y]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
(XY_DATA)
{ front_ride_height balance_coeff }
10.0 30.0 30.0 30.0 30.0
20.0 30.0 30.0 30.0 30.0
30.0 30.0 30.0 30.0 30.0
40.0 30.0 30.0 30.0 30.0
$-----CYB_YAW
[CYB_YAW]
{ yaw_angle cbody_effect }
-5.0 -1.0
-2.5 -1.0
0.0 0.0
2.5 1.0
5.0 1.0
$-----CYB_PITCH
[CYB_PITCH]
{ pitch_angle cbody_effect }
-30.0 1.00
-20.0 1.00
-10.0 1.00
0.0 1.00
10.0 1.00
20.0 1.00
30.0 1.00
$-----CYB_ROLL
```

```
[CYB_ROLL]
{ roll_angle cbody_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CYB_STEERING
[CYB_STEERING]
{ steering_angle cbody_effect }
 -10.0 1.0
 -5.0 1.0
 0.0 1.0
 5.0 1.0
 10.0 1.0
$-----CYF_YAW
[CYF_YAW]
{ yaw_angle cfront_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 1.0
 2.5 1.0
 5.0 1.0
$-----CYF_PITCH
[CYF_PITCH]
{ pitch_angle cfront_effect }
 -30.0 1.0
 -20.0 1.0
 -10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CYF_ROLL
[CYF_ROLL]
{ roll_angle cfront_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CYF_STEERING
[CYF_STEERING]
{ steering_angle cfront_effect }
 -10.0 1.0
 -5.0 1.0
 0.0 1.0
 5.0 1.0
 10.0 1.0
$-----CYR_YAW
[CYR_YAW]
{ yaw_angle crear_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 1.0
 2.5 1.0
 5.0 1.0
$-----CYR_PITCH
[CYR_PITCH]
{ pitch_angle crear_effect }
 -30.0 1.0
```

## Appendix B: File Format

```
-20.0 1.0
-10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CYR_ROLL
[CYR_ROLL]
{ roll_angle cbody_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CYR_STEERING
[CYR_STEERING]
{ steering_angle cbody_effect }
 -10.0 1.00
 -5.0 1.00
 0.0 1.00
 5.0 1.00
 10.0 1.00
$
$ rollmoment coefficients and modifiers
$-----CMX
[CMX]
(Z_DATA)
{rear_ride_height}
10.0
20.0
30.0
40.0
(XY_DATA)
{ front_ride_height cmx_coeff }
10.0 0.0 0.0 0.0 0.0
20.0 0.0 0.0 0.0 0.0
30.0 0.0 0.0 0.0 0.0
40.0 0.0 0.0 0.0 0.0
$-----CMXB_YAW
[CMXB_YAW]
{ yaw_angle cbody_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 0.0
 2.5 -1.0
 5.0 -1.0
$-----CMXB_PITCH
[CMXB_PITCH]
{ pitch_angle cbody_effect }
 -30.0 1.00
 -20.0 1.00
 -10.0 1.00
 0.0 1.00
 10.0 1.00
 20.0 1.00
 30.0 1.00
$-----CMXB_ROLL
[CMXB_ROLL]
{ roll_angle cbody_effect }
```

```
-2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CMXB_STEERING
[CMXB_STEERING]
{ steering_angle cbody_effect }
 -10.0 1.0
 -5.0 1.0
 0.0 1.0
 5.0 1.0
 10.0 1.0
$-----CMXF_YAW
[CMXF_YAW]
{ yaw_angle cfront_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 0.0
 2.5 -1.0
 5.0 -1.0
$-----CMXF_PITCH
[CMXF_PITCH]
{ pitch_angle cfront_effect }
 -30.0 1.0
 -20.0 1.0
 -10.0 1.0
 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CMXF_ROLL
[CMXF_ROLL]
{ roll_angle cfront_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CMXF_STEERING
[CMXF_STEERING]
{ steering_angle cfront_effect }
 -10.0 1.0
 -5.0 1.0
 0.0 1.0
 5.0 1.0
 10.0 1.0
$-----CMXR_YAW
[CMXR_YAW]
{ yaw_angle crear_effect }
 -5.0 1.0
 -2.5 1.0
 0.0 0.0
 2.5 -1.0
 5.0 -1.0
$-----CMXR_PITCH
[CMXR_PITCH]
{ pitch_angle crear_effect }
 -30.0 1.0
 -20.0 1.0
 -10.0 1.0
```

## Appendix B: File Format

```

 0.0 1.0
 10.0 1.0
 20.0 1.0
 30.0 1.0
$-----CMXR_ROLL
[CMXR_ROLL]
{ roll_angle crear_effect }
 -2.0 1.0
 0.0 1.0
 2.0 1.0
$-----CMXR_STEERING
[CMXR_STEERING]
{ steering_angle crear_effect }
 -10.0 1.00
 -5.0 1.00
 0.0 1.00
 5.0 1.00
 10.0 1.00

```

## 5.2 CFG

The CFG file defines the list of registered databases in the session.

It is automatically created by VI-CarRealTime when starting a new session, and it is updated when managing databases from the GUI.

The CFG file is used by the solver to "resolve" the databases, that is to replace the *alias* with the full path with which the database has been registered in the session.

```

!-----!
! ***** VI-CarRealTime Database Configuration File *****
!-----!
! Database name Path of Database
!-----!
DATABASE carrealtime_shared C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/CARREA~1.CDB
DATABASE visafety_shared C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/VISAFE~1.CDB
DATABASE CityCar C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/CityCar.cdb
DATABASE CompactCar C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/COMPAC~1.CDB
DATABASE Crossover C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/CROSSO~1.CDB
DATABASE Pickup C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/Pickup.cdb
DATABASE RaceCar C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/RaceCar.cdb
DATABASE SedanCar C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/SedanCar.cdb
DATABASE SportCar C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/SportCar.cdb
DATABASE SUV C:/PROGRA~1/VI-grade/VI-CAR~3/acarrt/examples/SHARED~1/SUV.cdb

```

The file format is composed by 3 columns representing respectively:

- DATABASE string;
- database *alias* (it is a common rule to set the database alias to have the same name as the database itself)
- full path of the database

## 5.3 DRD

A drd file is text file storing the set of data required to describe a path like space coordinates. The file should start with an header line (marked by the # character) containing the columns description: please note that the columns order is fixed and does not depends on the header content.

| #            | x-coord      | y-coord       | width        | lat.inc      | speed        | z-coord |
|--------------|--------------|---------------|--------------|--------------|--------------|---------|
| +8.8844E-009 | +0.0000E+000 | +1.50000E+001 | -7.7037E-034 | +0.0000E+000 | -4.2370E-033 |         |
| -5.0000E+000 | +0.0000E+000 | +1.50000E+001 | -3.6646E-019 | +0.0000E+000 | -4.2370E-033 |         |
| -1.0000E+001 | +0.0000E+000 | +1.50000E+001 | -4.5808E-019 | +0.0000E+000 | -4.2370E-033 |         |
| -1.5000E+001 | -4.4408E-016 | +1.50000E+001 | +4.0444E-033 | +0.0000E+000 | -4.2370E-033 |         |
| -2.0000E+001 | -2.2204E-015 | +1.50000E+001 | +1.0077E-018 | +0.0000E+000 | -4.2370E-033 |         |
| -2.5000E+001 | -2.6645E-015 | +1.50000E+001 | +1.4658E-018 | +0.0000E+000 | -4.2370E-033 |         |
| -3.0000E+001 | +0.0000E+000 | +1.50000E+001 | -1.7718E-032 | +0.0000E+000 | -4.2370E-033 |         |
| -3.5000E+001 | +7.1054E-015 | +1.50000E+001 | -3.6646E-018 | +0.0000E+000 | -4.2370E-033 |         |
| -4.0000E+001 | +1.0214E-014 | +1.50000E+001 | -5.4053E-018 | +0.0000E+000 | -4.2370E-033 |         |
| -4.5000E+001 | +4.4408E-016 | +1.50000E+001 | +5.7007E-032 | +0.0000E+000 | -4.2370E-033 |         |
| -5.0000E+001 | -2.4424E-014 | +1.50000E+001 | +1.3650E-017 | +0.0000E+000 | -4.2370E-033 |         |
| -5.5000E+001 | -3.6415E-014 | +1.50000E+001 | +2.0155E-017 | +0.0000E+000 | -4.2370E-033 |         |
| -6.0000E+001 | +4.4408E-016 | +1.50000E+001 | -2.4651E-031 | +0.0000E+000 | -4.2370E-033 |         |
| -6.5000E+001 | +9.3258E-014 | +1.50000E+001 | -5.0938E-017 | +0.0000E+000 | -4.2370E-033 |         |
| -7.0000E+001 | +1.3722E-013 | +1.50000E+001 | -7.5216E-017 | +0.0000E+000 | -4.2370E-033 |         |
| -7.5000E+001 | +4.4408E-016 | +1.50000E+001 | +8.6281E-031 | +0.0000E+000 | -4.2370E-033 |         |
| -8.0000E+001 | -3.4594E-013 | +1.50000E+001 | +1.9010E-016 | +0.0000E+000 | -4.2370E-033 |         |
| -8.5000E+001 | -5.1070E-013 | +1.50000E+001 | +2.8071E-016 | +0.0000E+000 | -4.2370E-033 |         |

### x-coord

Path point X coordinate

### y-coord

Path point Y coordinate

### width

Path variable width for the current point. the width information is used by the corner cutting functionality included in VI-Road.

### lat.inc

The lateral inclination data is a description of the path banking based on the trigonometric tangent of the banking.

### speed

The speed column is not strictly connected to the path definition, but a slot for storing the speed has been implemented in order to give a easy way to associate a reference speed to the path being described.

### z-coord

Path point Z coordinate

The given data should always be expressed in SI units (meter, seconds, radians) and no restrictions on the points resolution is required (so points spacing could change along the path).

## 5.4 PMF

The PMF file format specifies scaling factors for aerodynamics and electric motor performance in function of path S.

It is used with MaxPerformance, File driven and StaticLapTime events.

For Aerodynamics the modifier is applied as a scaling factor applied to the drag and downforce calculated by the solver.

The Motors modifiers works as follows:

- Greater than 0: modifies maximum torque available for the motor as a scaling factor.
- Equal to 0: motor gives 0 torque (open clutch).
- Less than 0: modifies maximum torque as a scaling factor and turns motor to regeneration mode.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'DAT'
FILE_VERSION = 1
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'meter'
ANGLE = 'rad'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----AERODYNAMIC_SCALE_MAP
[AERODYNAMIC_SCALE_MAP]
{ path_s air_drag_scale air_front_down_scale air_rear_down_scale}
0 1 1 1
50 1 1 1
100 1 1 1
150 1 1 1
200 1 1 1
250 1 1 1
600 1 1 1
300 1 1 1
350 1 1 1
400 1 1 1
450 1 1 1
500 1 1 1
550 1 1 1
600 1 1 1
$-----MOTOR_SCALE_MAP
$ > 0 : closed clutch
$ = 0 : open clutch
$ < 0 : closed clutch with regenerative brake
[MOTOR_SCALE_MAP]
{ path_s engine_scale main_motor_scale central_motor_scale front_motor_scale
rear_motor_scale front_left_motor_scale front_right_motor_scale rear_left_motor_scale
rear_right_motor_scale}
0 1 1 1 1 1
 1 1 1 1 1
50 1 -1 1 1 1
 1 1 1 1 1
100 1 -0.5 1 1 1
 1 1 1 1 1
```

|     |   |    |    |   |   |   |
|-----|---|----|----|---|---|---|
| 150 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  | 1 |   |   |
| 200 | 1 | 1  | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 250 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 300 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 350 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 400 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 450 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 1  |   |   |   |
| 500 | 1 | -1 | 1  | 1 | 1 | 1 |
|     | 1 |    | 0  | 0 |   |   |
| 550 | 1 | 1  | 1  | 1 | 1 | 1 |
|     | 1 |    | -1 |   |   |   |
| 600 | 1 | 1  | 1  | 1 | 1 | 1 |
|     | 1 |    | -1 |   |   |   |

## 5.5 PWR

The PWR file contains a 3D map defining the engine torque characteristic as a function of engine speed and throttle. The map resolution for both throttle and the engine speed is customizable.

Different flavors of property files are available:

- [single map](#)
- [gear dependent maps](#)
- [electric map](#)

### Single Map

In the example below a linear throttle dependency is defined (only 2 throttle values).

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'pwr'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
(BASE)
{length force angle mass time}
'mm' 'newton' 'degrees' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'rpm' 0 0 1 0 -1 6.0
'torque' 1 1 0 0 0 1.0
$-----ENGINE
[ENGINE]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
```

## Appendix B: File Format

```

0 0 0
500 -20000 120000
1000 -42000 270000
1500 -44000 400000
2000 -46000 490000
2500 -48000 526000
3000 -50000 620000
3500 -50000 716000
4000 -50000 808000
4500 -50000 910000
5000 -50000 950000
5500 -50000 970000
6000 -50000 936000
6250 -50000 924000
6500 -52000 910000
6750 -56000 854000
7000 -60000 740000
7500 -64000 690000
8000 -68000 650000

```

**Gear Dependent Maps**

The engine torque map can also be gear-dependent, such meaning that for each gear it's possible to specify a different engine map.

Here an example:

```

$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'pwr'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
(BASE)
{length force angle mass time}
'mm' 'newton' 'degrees' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'rpm' 0 0 1 0 -1 6.0
'torque' 1 1 0 0 0 1.0

$----- ENGINE_MAP_LIST
[ENGINE_MAP_LIST]
GEAR_DEPENDENCY='TRUE'
GENERIC_MAP='ENGINE'
REVERSE_GEAR='ENGINE_MAP_REVERSE_GEAR'
GEAR_1='ENGINE_MAP_GEAR_1'
GEAR_2='ENGINE_MAP_GEAR_2'
GEAR_3='ENGINE_MAP_GEAR_3'
GEAR_4='ENGINE_MAP_GEAR_4'

$-----ENGINE
[ENGINE]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
 0 0 0

```

```
500 -69262 80000
1000 -69263 235000
1500 -69266 400000
2000 -69272 400000
2500 -69284 400000
3000 -69307 400000
3500 -69353 400000
4000 -69447 400000
4500 -69625 400000
5000 -69750 400000
5500 -69875 400000
6000 -70250 400000
6500 -73000 400000
7000 -72000 400000
7500 -74000 400000
$-----ENGINE_MAP_REVERSE_GEAR
[ENGINE_MAP_REVERSE_GEAR]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
 0 0 0
 500 -69266 180000
1000 -69263 235000
1500 -69266 650000
2000 -69272 650000
2500 -69284 650000
3000 -69307 650000
3500 -69353 650000
4000 -69447 650000
4500 -69625 650000
5000 -69750 650000
5500 -69875 650000
6000 -70250 650000
6500 -73000 650000
7000 -72000 650000
7500 -74000 500000

$-----ENGINE_MAP_GEAR_1
[ENGINE_MAP_GEAR_1]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
 0 0 0
 500 -69266 180000
1000 -69263 650000
1500 -69266 650000
2000 -69272 650000
2500 -69284 650000
3000 -69307 650000
3500 -69353 650000
4000 -69447 650000
4500 -69625 650000
5000 -69750 650000
5500 -69875 650000
6000 -70250 650000
```

## Appendix B: File Format

```
6500 -73000 650000
7000 -72000 650000
7500 -74000 500000
```

```
$-----ENGINE_MAP_GEAR_2
[ENGINE_MAP_GEAR_2]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
0 0 0
500 -69262 80000
1000 -69263 550000
1500 -69266 550000
2000 -69272 550000
2500 -69284 550000
3000 -69307 550000
3500 -69353 550000
4000 -69447 550000
4500 -69625 550000
5000 -69750 550000
5500 -69875 550000
6000 -70250 550000
6500 -73000 550000
7000 -72000 550000
7500 -74000 550000
```

```
$-----ENGINE_MAP_GEAR_3
[ENGINE_MAP_GEAR_3]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
0 0 0
500 -69262 80000
1000 -69263 450000
1500 -69266 450000
2000 -69272 450000
2500 -69284 450000
3000 -69307 450000
3500 -69353 450000
4000 -69447 450000
4500 -69625 450000
5000 -69750 450000
5500 -69875 450000
6000 -70250 450000
6500 -73000 450000
7000 -72000 450000
7500 -74000 450000
```

```
$-----ENGINE_MAP_GEAR_4
[ENGINE_MAP_GEAR_4]
(Z_DATA)
{throttle}
```

```

0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
 0 0 0
 500 -69262 80000
1000 -69263 440000
1500 -69266 440000
2000 -69272 440000
2500 -69284 440000
3000 -69307 440000
3500 -69353 440000
4000 -69447 440000
4500 -69625 440000
5000 -69750 440000
5500 -69875 440000
6000 -70250 440000
6500 -73000 440000
7000 -72000 440000
7500 -74000 500000

```

**[ENGINE\_MAP\_LIST]** block defines the following parameters:

- **GEAR\_DEPENDENCY='TRUE'**

It must be set to TRUE in order to activate the multiple maps mode.

- **GENERIC\_MAP='ENGINE'**

It associates the map defined in the [ENGINE] block (which must exist in the property file) to the generic map. Such map will be used for all the gears which are not explicitly defined into ENGINE\_MAP\_LIST block.

- **REVERSE\_GEAR='ENGINE\_MAP\_REVERSE\_GEAR'**

It forces the solver to use the torque map defined in [ENGINE\_MAP\_REVERSE\_GEAR] block (which must exist in the property file), whenever the driver selects the reverse gear.

- **GEAR\_1='ENGINE\_MAP\_GEAR\_1'**

It forces the solver to use the torque map defined in [ENGINE\_MAP\_GEAR\_1] block (which must exist in the property file), whenever the driver selects the first gear. Same concept applies for all the other gears.

## Electric Maps

For electric motors, it is possible to select a PWR property file which defines the following motor characteristic maps:

- Motor Torque Map  
[ENGINE] block of pwr file.
- Braking Torque Map  
[ENGINE\_BRAKING] block of pwr file.
- Coasting Torque Map  
[COASTING] block of pwr file.

```

$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'pwr'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
$-----UNITS

```

## Appendix B: File Format

```
[UNITS]
(BASE)
{length force angle mass time}
'mm' 'newton' 'degrees' 'kg' 'sec'
(USER)
{unit_type length force angle mass time conversion}
'rpm' 0 0 1 0 -1 6.0
'torque' 1 1 0 0 0 1.0
$-----ENGINE

[ENGINE]
(Z_DATA)
{throttle}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
0 0 0
500 0 80000
1000 0 235000
1500 0 400000
2000 0 600000
2500 0 625000
3000 0 650000
3500 0 700000
4000 0 750000
4500 0 740000
5000 0 710000
5500 0 680000
6000 0 650000
6500 0 570000
7000 0 520000
7500 0 500000
$----- ENGINE_BRAKING

[ENGINE_BRAKING]
(Z_DATA)
{brake}
0.0
1.00
(XY_DATA)
{engine_speed <rpm> torque@throttle <torque>}
0 0 0
500 0 80000
1000 0 235000
1500 0 400000
2000 0 600000
2500 0 625000
3000 0 650000
3500 0 700000
4000 0 750000
4500 0 740000
5000 0 710000
5500 0 680000
6000 0 650000
6500 0 570000
7000 0 520000
7500 0 500000
$----- COASTING

[COASTING]
{engine_speed <rpm> torque <torque>}
0 0
500 -6926
```

```

1000 -6926
1500 -6926
2000 -6927
2500 -6928
3000 -6930
3500 -6935
4000 -6944
4500 -6962
5000 -6975
5500 -6987
6000 -7025
6500 -7300
7000 -7200
7500 -7400

```

**Note:** if [ENGINE] block is missing from property file, an error is thrown and the simulation is stopped. If either [ENGINE\_BRAKING] or [COASTING] blocks are missing, a warning is raised and null maps are used for such blocks.

## 5.6 REP

The Full-Vehicle Report File (REP file) collects the information about the model in different conditions:

- **Assembly/Design Condition**

Vehicle information at design time condition.

- **Before Setup**

Vehicle information after the first static equilibrium has been achieved.

- **After Setup**

Vehicle condition after the adjustments phase (such block is written only if vehicle contains adjustments).

- **Before Running**

Vehicle condition just before entering dynamic analysis.

**Note:** Measures defined in report file are expressed in Global Reference Frame system.

```
=====
= VI/CarRealTime =
= VEHICLE SETUP STATUS REPORT =
=====
```

### Before Running

```

SIMULATION TIME = 0.000000

MODEL UNITS :
LENGTH = METER
MASS = KILOGRAM
TIME = SECOND
FORCE = NEWTON
ANGLE = RADIAN

<> VEHICLE GLOBAL CONDITIONS :
>> VEHICLE CG ABS X = -1.679645
>> VEHICLE CG ABS Y = -0.000000
>> VEHICLE CG HEIGHT = 0.370820
>> VEHICLE INERTIA IXX = 180.268539
>> VEHICLE INERTIA IYY = 1231.926838
>> VEHICLE INERTIA IZZ = 1286.423882

```

## Appendix B: File Format

```
>> VEHICLE INERTIA IXY = 0.000000
>> VEHICLE INERTIA IXZ = 1.410954
>> VEHICLE INERTIA IYZ = 0.000000
>> VEHICLE MASS = 677.925941
>> DRIVER MASS = 74.842741
>> FUEL MASS = 46.000000
>> VEHICLE WEIGHT = 677.925941
>> CHASSIS CG GLOBAL X = -1.717525
>> CHASSIS CG GLOBAL Y = -0.000000
>> CHASSIS CG GLOBAL Z = 0.403595
>> CHASSIS CG LOCAL X = -1.722707
>> CHASSIS CG LOCAL Y = -0.000000
>> CHASSIS CG LOCAL Z = 0.400981
```

<> FRONT LEFT SUSPENSION SETUP CONDITIONS :

```
>> WEIGHT = 148.473296
>> WEIGHT RATIO% = 21.901093
>> TIRE RADIUS = 0.324303
>> TOE ANGLE = 0.002205
>> CAMBER ANGLE = -0.048279
>> PIN HEIGHT = 0.003363
>> RIDE HEIGHT = 0.023421
```

<> FRONT RIGHT SUSPENSION SETUP CONDITIONS :

```
>> WEIGHT = 148.473296
>> WEIGHT RATIO% = 21.901093
>> TIRE RADIUS = 0.324303
>> TOE ANGLE = 0.002205
>> CAMBER ANGLE = -0.048279
>> PIN HEIGHT = 0.003363
>> RIDE HEIGHT = 0.023421
```

<> REAR LEFT SUSPENSION SETUP CONDITIONS :

```
>> WEIGHT = 190.489912
>> WEIGHT RATIO% = 28.098907
>> TIRE RADIUS = 0.353974
>> TOE ANGLE = 0.001731
>> CAMBER ANGLE = -0.061224
>> PIN HEIGHT = 0.014572
>> RIDE HEIGHT = 0.062467
```

<> REAR RIGHT SUSPENSION SETUP CONDITIONS :

```
>> WEIGHT = 190.489912
>> WEIGHT RATIO% = 28.098907
>> TIRE RADIUS = 0.353974
>> TOE ANGLE = 0.001731
>> CAMBER ANGLE = -0.061224
>> PIN HEIGHT = 0.014572
>> RIDE HEIGHT = 0.062467
```

Please note that some fields of the SETUP CONDITIONS blocks could not be filled (so will show 0).

## 5.7 SPC

The SPC file contains the information needed to configure and run a seven postrig simulation. In addition to the standard header and units blocks, the file contains a configuration block and when required by the selected analysis mode also a data block holding experimental data.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'spc'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'mm'
ANGLE = 'degrees'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'

$-----SEVENPOST_SETUP
[SEVENPOST_SETUP]
OPERATING_MODE = 'TRACK_SPRING'
```

Five operating modes are currently supported:

- OFF  
All inputs to the vehicle are set to 0 so no pad displacement will occur and no down forces will be applied.
- PASSTHROUGH  
The controller in this case acts as a bypass.
- OPEN  
The mode works in combination with a set of tabular data function of time: the information stored in the [DATA] table are connected to the sevenpostrig actuator with no preprocessing.
- TRACK\_HUBACC  
The mode works in combination with a set of tabular data function of time: it is designed to actuate the pad displacement for tracking hub acceleration.
- TRACK\_SPRING  
The mode works in combination with a set of tabular data function of time: it is designed to actuate the pad displacement for tracking spring length.

```
ACTUATORS_MODE = 'DISPLACEMENT'
```

When the *OPEN* and *PASSTHROUGH* modes are selected the actuators could set to one of the following options:

- DISPLACEMENT
- VELOCITY
- ACCELERATION

**Note:** when *TRACK\_HUBACC* or *TRACK\_SPRING* operating mode is selected, the actuators are automatically set to *DISPLACEMENT* mode.

```
AERODYNAMICS = 'DATA'
```

The main role of the downforce actuators is to apply to the vehicle the aerodynamic forces.

## Appendix B: File Format

Three options are available for including the aerodynamic effect:

- OFF  
it switches off the aerodynamic forces.
- DATA  
it retrieves the aerodynamic forces from the DATA table.
- MODEL  
it activates the built in aerodynamic from the car model.

**Note:** when using PASSTHROUGH operating mode, the DATA option is not supported.

```
ACCEL_COMPENSATION = 'ON'
```

In addition to the aerodynamic forces, the downforce actuators could be used to introduce the load transfer effects produced by longitudinal and lateral acceleration. In order to have this feature working the DATA table should contain the ax and ay columns (9th and 10th columns). It is recommended to supply vehicle center of mass acceleration.

```
ROLL_COMPENSATION = 'ON'
PITCH_COMPENSATION = 'ON'
```

Selecting TRACK\_HUBACC and TRACK\_SPRING modes, two additional controller module could be enabled in order to compensate roll and pitch drifts produced by noise in the experimental data. It is strongly recommended to activate this options for flat road reconstruction.

```
SPEED_MODE = 'MODEL'
```

When the aerodynamic mode is set to *MODEL* it is possible to select two different sources for the vehicle speed: when the SPEED\_MODE option is set to model a constant speed is assigned to the aeroforces input (defined during the model generation process). Selecting the option FILE the speed data are inherited from the 10th column of the DATA table.

```
STEER_MODE = 'MODEL'
```

when the selected steering actuation mode is motion, the steering input could be supplied using the 11th column of the DATA table setting the STEER\_MODE key to FILE. Using STEER\_MODE = 'MODEL' keep the steering locked to the constant input specified in the testrig setup panel.

```
$-----DATA
[DATA]
{time FL FR RL RR F_down RL_down RR_down lonacc latacc lonvel steer
}
0.00E+0 4.10E+3 7.25E+3 3.36E+3 7.43E+3 3.47E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
1.00E-2 2.05E+2 -1.75E+2 1.07E+2 -2.14E+2 3.47E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
2.00E-2 1.90E+1 -7.58E+1 1.17E+2 1.76E+2 3.47E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
3.00E-2 1.63E+2 -2.89E+2 -2.91E+2 4.88E+2 3.47E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
4.00E-2 3.71E+2 -4.70E+2 -5.51E+2 5.40E+2 3.46E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
5.00E-2 6.06E+2 -7.32E+2 -3.52E+2 2.69E+2 3.46E+3 1.73E+3 1.73E+3 0.0 0.0 0.0 0.0
```

The [DATA] table holds the experimental data input for the sevenpost controller. The first column is the time and it should be followed by four columns defining the targets for the pad actuation, the meaning of the data stored in columns 2 to 5 depends on the selected OPERATING\_MODE.

Using OPEN mode, the FL, FR, RL, RR columns store the pad DISPLACEMENT, VELOCITY or ACCELERATION according to the selected ACTUATORS\_MODE; when TRACK\_HUBACC is enabled, the FL, FR, RL, RR

represent hub absolute vertical acceleration. Finally using TRACK\_SPRING columns 2 to 5 should contain the spring length.

In addition to the basic five columns input, three extra columns could be specified for aerodynamic downforces and two extra columns for body accelerations. Please note that body acceleration (when provided) should be stored in columns 9th and 10th.

The sign convention adopted is the following:

1. pad related data are positive upward
2. downforce related data are positive downward
3. longitudinal acceleration is positive during acceleration (so negative on braking)
4. lateral acceleration is positive during right turn

#### Notes:

- The hub acceleration tracking mode use a complex signal conditioning process to isolate signal noise. The preprocessed signal is shorter than the original one by approximately four seconds.
- Older spc files are not compatible with the current format (v 2.0) and needs to be updated manually changing the following elements:
  1. HEADER block is required
  2. SEVENPOST\_SETUP block supersedes older blocks ACTUATION\_SETUP and AERO\_SETUP
  3. the DATA table could contain 2 extra columns storing vehicle center of mass longitudinal and lateral acceleration.
  4. the DATA table could contain 2 extra columns (in addition to point 3) for longitudinal velocity and steering input

The [DATA] table can also be formatted in a different way in order to actuate the 7Post Rig with forces and moments at wheel hubs.

In order to use such SPC file format, the following constraints must be respected:

- OPERATING\_MODE = 'OPEN' in [SEVENPOST\_SETUP] block;
- ACTUATORS\_MODE = 'DISPLACEMENT' in [SEVENPOST\_SETUP] block;
- Wheel Actuator Mode must be set to *hub* in 7Post event panel;
- Body Fix Setup Mode must be set to *Fix Body at Design* or to *Fix Body at Setup* in 7Post event panel;

The following input channels are available:

- time  
time column.
- FL, FR, RL RR  
target vertical displacement of the 4 pads.
- FL\_FX\_BRK, FR\_FX\_BRK, RL\_FX\_BRK, RR\_FX\_BRK  
longitudinal braking force applied in the [ISO-W](#) reference frame at W point location.
- FL\_FX\_DRV, FR\_FX\_DRV, RL\_FX\_DRV, RR\_FX\_DRV  
longitudinal driving force applied in the [ISO-C](#) reference frame at C point location.
- FL\_FY, FR\_FY, RL\_FY, RR\_FY  
lateral force applied in the [ISO-W](#) reference frame at W point location.
- FL\_MX, FR\_MX, RL\_MX, RR\_MX  
overturning moment applied in the [ISO-W](#) reference frame at W point location.
- FL\_MY, FR\_MY, RL\_MY, RR\_MY

## Appendix B: File Format

rolling resistance moment applied in the [ISO-W](#) reference frame at W point location.

- FL\_MZ, FR\_MZ, RL\_MZ, RR\_MZ  
aligning moment applied in the [ISO-W](#) reference frame at W point location.

**Note:** the order of the column is not fixed and it's not mandatory to define all the channels in the DATA block; only the channels defined in the DATA block will be actually used by the 7Post Rig.

An example of SPC file defining external input forces is shipped with carrealtime\_shared database ([mdids://carrealtime\\_shared/loadcases.tbl/hub\\_force\\_inputs.spc](mdids://carrealtime_shared/loadcases.tbl/hub_force_inputs.spc)).

## 5.7.1 Example 1

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'spc'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'mm'
ANGLE = 'degrees'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----SEVENPOST_SETUP
[SEVENPOST_SETUP]
OPERATING_MODE = 'OPEN'
AERODYNAMICS = 'OFF'
ACTUATORS_MODE = 'DISPLACEMENT'
ACCEL_COMPENSATION = 'OFF'
$-----DATA
[DATA]
{time FL FR RL RR }
0 0 13.6394614 0 25.24412954
0.01 1.883715586 11.65857999 5.877852523 26.65750336
0.02 3.759997007 8.945147127 9.510565163 27.8342633
0.03 5.621439438 5.669657768 9.510565163 28.76396434
0.04 7.460696615 2.03792296 5.877852523 29.4383544
0.05 9.270509831 -1.721862042 1.22515E-15 29.85144754
0.06 11.04373658 -5.37345612 -5.877852523 29.99957712
0.07 12.77337875 -8.687416187 -9.510565163 29.88142832
0.08 14.45261022 -11.45551395 -9.510565163 29.49804986
0.09 16.07480385 -13.50381963 -5.877852523 28.85284461
0.1 17.63355757 -14.70363067 -2.4503E-15 27.95153948
0.11 19.12271969 -14.97955851 5.877852523 26.8021345
0.12 20.53641318 -14.31426561 9.510565163 25.41483188
0.13 21.86905882 -12.74955475 9.510565163 23.80194541
0.14 23.11539728 -10.38374247 5.877852523 21.97779119
0.15 24.27050983 -7.365481469 3.67545E-15 19.95856054
0.16 25.32983777 -3.884420178 -5.877852523 17.76217629
0.17 26.2892004 -0.159286481 -9.510565163 15.40813371
0.18 27.14481157 3.575855772 -9.510565163 12.91732744
0.19 27.89329458 7.086313855 -5.877852523 10.31186606
0.2 28.53169549 10.15151278 -4.90059E-15 7.614875826
0.21 29.05749483 12.57885482 5.877852523 4.850295415
0.22 29.46861752 14.21582115 9.510565163 2.042663439
0.23 29.76344104 14.95955515 9.510565163 -0.783099363
0.24 29.94080185 14.76332529 5.877852523 -3.601911319
0.25 30 13.6394614 6.12574E-15 -6.388752455
```

|      |             |              |              |              |
|------|-------------|--------------|--------------|--------------|
| 0.26 | 29.94080185 | 11.65857999  | -5.877852523 | -9.118886572 |
| 0.27 | 29.76344104 | 8.945147127  | -9.510565163 | -11.76808081 |
| 0.28 | 29.46861752 | 5.669657768  | -9.510565163 | -14.31282072 |
| 0.29 | 29.05749483 | 2.03792296   | -5.877852523 | -16.73051903 |
| 0.3  | 28.53169549 | -1.721862042 | -7.35089E-15 | -18.99971606 |

## 5.7.2 Example 2

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'spc'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'mm'
ANGLE = 'degrees'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----SEVENPOST_SETUP
[SEVENPOST_SETUP]
OPERATING_MODE = 'TRACK_HUBACC'
AERODYNAMICS = 'DATA'
ACCEL_COMPENSATION = 'OFF'
$-----DATA
[DATA]
{time FL FR RL RR F_down RL_down RR_down}
+0.00E+0 +4.54E+3 +8.35E+3 +3.76E+3 +8.41E+3 +4.25E+3 +2.12E+3 +2.12E+3
+1.00E-2 +2.07E+2 -2.45E+2 +1.50E+2 -2.27E+2 +4.25E+3 +2.12E+3 +2.12E+3
+2.00E-2 +8.02E+0 -1.16E+2 +1.65E+2 +1.41E+2 +4.25E+3 +2.12E+3 +2.12E+3
+3.00E-2 -6.24E+1 -8.83E+1 -1.27E+2 +4.13E+2 +4.25E+3 +2.12E+3 +2.12E+3
+4.00E-2 +7.18E+1 -3.06E+2 -3.22E+2 +4.65E+2 +4.25E+3 +2.12E+3 +2.12E+3
+5.00E-2 +2.99E+2 -4.50E+2 -2.28E+2 +3.51E+2 +4.25E+3 +2.12E+3 +2.12E+3
+6.00E-2 +7.01E+2 -8.54E+2 +1.01E+2 +1.90E+1 +4.25E+3 +2.12E+3 +2.12E+3
+7.00E-2 +2.36E+3 -1.55E+2 -5.75E+1 -9.87E+2 +4.24E+3 +2.12E+3 +2.12E+3
+8.00E-2 +2.39E+3 -3.06E+2 -3.69E+2 -1.69E+3 +4.23E+3 +2.11E+3 +2.11E+3
+9.00E-2 +1.96E+3 -4.30E+2 -2.93E+2 -1.47E+3 +4.20E+3 +2.10E+3 +2.10E+3
+1.00E-1 +1.18E+3 -4.74E+2 -9.61E+1 -8.57E+2 +4.16E+3 +2.08E+3 +2.08E+3
```

## 5.7.3 Example 3

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'spc'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'mm'
ANGLE = 'degrees'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----SEVENPOST_SETUP
[SEVENPOST_SETUP]
OPERATING_MODE = 'TRACK_SPRING'
```

## Appendix B: File Format

```

AERODYNAMICS = 'DATA'
ACCEL_COMPENSATION = 'ON'
ROLL_COMPENSATION = 'ON'
PITCH_COMPENSATION = 'ON'
$-----DATA

[DATA]

{time FL FR RL RR F_down RL_down RR_down lonacc latacc}
0.00E+0 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.24E+2 2.02E+2 2.02E+2 -3.00E+2 +2.36E+0
1.00E-2 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.24E+2 2.02E+2 2.02E+2 -8.05E+2 -9.09E-1
2.00E-2 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.24E+2 2.02E+2 2.02E+2 -1.08E+3 -1.61E+1
3.00E-2 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.24E+2 2.02E+2 2.02E+2 -1.31E+3 -3.98E+1
4.00E-2 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.23E+2 2.02E+2 2.02E+2 -1.52E+3 -5.15E+1
5.00E-2 2.17E+2 2.17E+2 2.82E+2 2.82E+2 3.23E+2 2.02E+2 2.02E+2 -1.71E+3 -5.28E+1
6.00E-2 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.23E+2 2.02E+2 2.02E+2 -1.85E+3 -4.50E+1
7.00E-2 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.23E+2 2.02E+2 2.02E+2 -1.91E+3 +1.68E+2
8.00E-2 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.23E+2 2.01E+2 2.01E+2 -1.55E+3 +2.15E+2
9.00E-2 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -1.22E+3 +4.66E+2
1.00E-1 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -8.71E+2 +2.81E+2
1.10E-1 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -5.10E+2 +4.28E+2
1.20E-1 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -3.43E+2 +4.68E+2
1.30E-1 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -6.59E+2 +5.14E+2
1.40E-1 2.16E+2 2.16E+2 2.82E+2 2.82E+2 3.22E+2 2.01E+2 2.01E+2 -1.21E+3 +5.57E+2
1.50E-1 2.16E+2 2.16E+2 2.82E+2 2.83E+2 3.22E+2 2.01E+2 2.01E+2 -1.39E+3 +5.96E+2
1.60E-1 2.16E+2 2.16E+2 2.82E+2 2.83E+2 3.22E+2 2.01E+2 2.01E+2 -1.43E+3 +6.28E+2
1.70E-1 2.16E+2 2.16E+2 2.82E+2 2.83E+2 3.21E+2 2.01E+2 2.01E+2 -1.43E+3 +6.53E+2
1.80E-1 2.16E+2 2.16E+2 2.82E+2 2.83E+2 3.21E+2 2.01E+2 2.01E+2 -1.41E+3 +6.71E+2
1.90E-1 2.15E+2 2.16E+2 2.82E+2 2.83E+2 3.21E+2 2.00E+2 2.00E+2 -1.40E+3 +6.81E+2
2.00E-1 2.15E+2 2.16E+2 2.82E+2 2.83E+2 3.21E+2 2.00E+2 2.00E+2 -1.40E+3 +6.83E+2
2.10E-1 2.15E+2 2.16E+2 2.82E+2 2.83E+2 3.21E+2 2.00E+2 2.00E+2 -1.39E+3 +6.77E+2
2.20E-1 2.15E+2 2.16E+2 2.82E+2 2.83E+2 3.20E+2 2.00E+2 2.00E+2 -1.39E+3 +6.92E+2

```

## 5.7.4 Example 5

This SPC file example defines a DATA block which provides the following input to the 7Post Rig:

- pads vertical displacement target (FL, FR, RL, RR)
- steering wheel angle (STEER\_ANGLE)
- lateral force applied at the wheel hubs (FL\_FY, RL\_FY, FR\_FY, RR\_FY)
- longitudinal braking force applied at front right wheel hub (FR\_RX\_BRK)
- longitudinal driving force applied at front right wheel hub (FR\_RX\_DRV)
- overturning moment applied at front right wheel hub (FR\_MX)
- rolling resistance moment applied at front right wheel hub (FR\_MY)
- aligning moment applied at front right wheel hub (FR\_MZ)

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'spc'
FILE_VERSION = 2.0
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]

```

```

LENGTH = 'mm'
ANGLE = 'degrees'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----SEVENPOST_SETUP
[SEVENPOST_SETUP]
OPERATING_MODE = 'OPEN'
AERODYNAMICS = 'OFF'
ACTUATORS_MODE = 'DISPLACEMENT'
ACCEL_COMPENSATION = 'OFF'
STEER_MODE = 'FILE'
$-----DATA
[DATA]
{time FL FR RL RR FL_FY RL_FY FR_FY
RR_FY STEER_ANGLE FR_FX_BRK FR_FX_DRV FR_MX FR_MY FR_MZ}
0.00 0.0 0.0 0.00 25.24 -0.00 0.0 0.0 -
0.00 0.0 0.0 0.00 -0.00 -0.00 -0.00 -0.00
0.01 0.0 0.0 5.87 26.65 -125.33 0.0 0.0 -
31.41 11.30 157.05 -31.41 -31.41 -31.41 -31.41
0.02 0.0 0.0 9.51 27.83 -248.68 0.0 0.0 -
62.79 22.55 313.95 -62.79 -62.79 -62.79 -62.79
0.03 0.0 0.0 9.51 28.76 -368.12 0.0 0.0 -
94.10 33.72 470.54 -94.10 -94.10 -94.10 -94.10
0.04 0.0 0.0 5.87 29.43 -481.75 0.0 0.0 -
125.33 44.76 626.66 -125.33 -125.33 -125.33 -125.33
0.05 0.0 0.0 0.00 29.85 -587.78 0.0 0.0 -
156.43 55.62 782.17 -156.43 -156.43 -156.43 -156.43

```

## 5.8 VTT

The VTT file contains the information needed to configure and run a VI-TireTestrigRealtime simulation. In addition to the standard header and units blocks, the file contains a model block and when required by the selected analysis mode also a data block holding simulation input data.

### 5.8.1 VIGRADE\_HEADER Block

The header block contains the general information about the tiem orbit file like the version and the format. No user settings are currently supported in this block

```

[VIGRADE_HEADER]
FILE_TYPE = 'vtt'
FILE_VERSION = 1
FILE_FORMAT = 'ASCII'

```

### 5.8.2 UNITS\_Block

The information included in each VTT file are expressed in the units system declared in the units block.

```

[UNITS]
LENGTH = 'meter'
ANGLE = 'degree'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'

```



## 5.8.4 SYSTEM Block

The system block collects the solver parameters used for the event.

## [SYSTEM]

**INTEGRATION STEP = 0.001**

It is the time step at which equation of motion are solved.

STATIC MAX TIME = 100

**\$ [OPTIONAL]**

It is the time limit to achieve static equilibrium (as default it is set at 100 sec).

**BUFFER OUTPUT = 'TRUE'**

**\$ [OPTIONAL]**

If true the analysis results are written at the end of simulation. The default is TRUE.

TIRE INTEGRATOR ACTIVE = 'FALSE'

**\$ [OPTIONAL]**

If true, solver uses a separated euler solver for tire computation. This solver will use TIRE INTEGRATION STEP value as integrational time step. The default is FALSE.

TIRE INTEGRATION STEP = 0.001

It represent the integrator frequency of eternal solver for tire computation.

## 5.8.5 ANALYSIS Block

All information on the simulation to be performed are included in this block. Some of the parameters are common to all simulation types, while others are specific to the type of events to be performed.

Detailed description of the parameters is given in the following sections:

[Common Analysis](#)  
[Longitudinal Slip Variation Analysis](#)  
[Lateral Slip Variation Analysis](#)  
[Combined Analysis](#)  
[Cornering Stiffness Analysis](#)  
[Free Input Analysis](#)  
[Six Dof Analysis](#)  
[Z Locked Analysis](#)

### ANALYSIS Block Common

The following parameters are independent from the chosen event.

#### [ANALYSIS]

**OUTPUT\_PREFIX = 'tt\_event\_1'**

Output prefix for the results file.

**INTIAL\_TIME = 0 \$ [OPTIONAL]**

Initial time at which the simulation starts (the default value is set to 0).

**END\_TIME = 2**

Simualtion end time.

**OUTPUT\_STEP = 0.01**

It is the time step used to write output result file.

**VELOCITY = 27.7777778**

Set the longitudinal translational velocity of the tire (vehicle speed).

**SIDE = 'LEFT'**

Tire side. Keys accepted are : LEFT (default), RIGHT, SYMMETRIC and MIRRORED.

**XML\_OUTPUT = 1**

This key enables result file in xml format (default value is 1).

**CSV\_OUTPUT = 0**

This key enables result file in comma separated file format (default value is 0).

**TAB\_OUTPUT = 0**

This key enables result file in tab separated file format (default value is 0).

**MAT\_OUTPUT = 0**

This key enables result file in matlab file format (default value is 0).

**TYPE = 'LONGITUDINAL'**

Label to identify the event you will run. Tire testing realtime supports the following events:

**LONGITUDINAL**

Analysis will be performed at constant longitudinal velocity, wheel angular velocity will be tuned in order to cover whole longitudinal slip interval.

**LATERAL**

Analysis will be performed at constant longitudinal velocity. Wheel spin rotation is free. A sine signal of user defined amplitude is used to lead toe angle.

**COMBINED**

Analysis will be performed at constant longitudinal velocity, wheel spin rotation will be tuned in order to match longitudinal slip desired. A sine signal of user defined amplitude is used to lead toe angle.

**CORNERING\_STIFFNESS**

Analysis will be performed at constant longitudinal velocity, wheel spin rotation will be tuned in order to cover whole longitudinal slip interval. Normal load will follow a user written function.

**FREE\_INPUT**

Analysis will be performed at constant longitudinal velocity, wheel spin rotation will be, as user request, free or prescribed. User will be able to impose toe and preload variation laws.

**SIX\_DOF**

Analysis will be performed using a free wheel, just imposing hub initial conditions.

**Z\_LOCKED**

Analysis will be performed using a model having prescribed hub position and free wheel spin rotation.

## ANALYSIS Block Longitudinal Slip

For Longitudinal Slip event no more parameters are required.

## ANALYSIS Block Lateral Slip

For Lateral Slip Variation event the user must specify the toe amplitude:

```
TOE_AMPLITUDE = 10
```

It defines the range to explore for lateral slip.

## ANALYSIS Block Combined

For Combined event the user must specify the toe amplitude and the desired value for the longitudinal slip.

```
TOE_AMPLITUDE = 1
```

It defines the range to explore for lateral slip.

```
LONG_SLIP = 5
```

Longitudinal (Constant) slip imposed to the wheel during the lateral slip variation.

## ANALYSIS Block CornStiff

For Cornering Stiffness event the user must specify the toe constant value and the desired vertical load function.

```
TOE_VALUE = 1
```

Contant value for the tire toe.

```
(PRELOAD)
```

```
PRELOAD_FUNCTION_TYPE = ''
```

Set how the vertical load on the tire should be modified during the simulation.

Detailed description about the predefined functions is given in the [Function Block](#) section

## ANALYSIS Block Free Input

For Free Input event the user must specify the desired functions for toe lateral slip (toe angle), vertical preload and for the wheel omega (if the user wants to impose wheel rotation using a motion):

```
PREScribed_ROTATION = 'TRUE' $ Allowed values: TRUE, FALSE
```

Flag to activate/deactivate the wheel rotational motion. If it is false it means free wheel.

```
(TOE)
```

```
TOE_FUNCTION_TYPE = ''
```

Define toe as a function of time.

```
(PRELOAD)
```

```
PRELOAD_FUNCTION_TYPE = ''
```

Define the vertical load as a function of time.

```
(WHEEL_OMEGA)
```

```
WHEEL_OMEGA_FUNCTION_TYPE = ''
```

Define wheel rotation as a function of time (only in case of true value for PREScribed\_ROTATION key).

Detailed description about the predefined functions is given in the [Function Block](#) section

## ANALYSIS Block Six Dof

For Six Dof event the user must specify, in a block named [SIX\_DOF] the desired initial conditions and inertial values for wheel part

```
$-----SIX_DOF
[SIX_DOF]

IXX = 10
Main inertia along x wheel direction.

IYY = 1
Wheel spin inertia.

IZZ = 10
Main inertia along z wheel direction.

MASS = 380
Wheel mass

z = 0.275
Hub z position.

vx = 5
Initial translational velocity along x direction.

vy = 1
Initial translational velocity along y direction.

vz = 0
Initial translational velocity along z direction.

wx = 0
Initial angular velocity around x direction.

wy = 0
Initial angular velocity around y direction.

wz = 0
Initial angular velocity around z direction.
```

## ANALYSIS Block Z Locked

In Z Locked analysis, hub carrier will be maintained at prescribed height for all dynamic simulation. User must specify, in a block named [Z\_LOCKED] the desired initial conditions and inertial values for wheel part

```
$-----[Z_LOCKED]
[IYY = 1
 Wheel spin inertia.

MASS = 380
 Wheel mass

Z = 0.275
 Hub z position. If missing, a static analysis will be performed in order to lock hub at equilibrium height.

VX = 5
 Initial translational velocity along x direction.

VY = 1
 Initial translational velocity along y direction.

VZ = 0
 Initial translational velocity along z direction.

WX = 0
 Initial angular velocity around x direction.

WY = 0
 Initial angular velocity around y direction.

WZ = 0
 Initial angular velocity around z direction.
```

## 5.8.6 FUNCTION

Cornering Stiffness and Free Input event need the use of time changing law as input.  
The implemented function are the following:

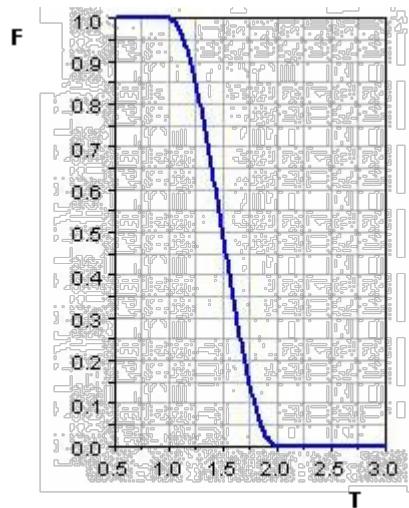
[STEP](#)  
[SINE](#)  
[RAMP](#)  
[USER](#)

## STEP

Step Example:

```
PRELOAD_FUNCTION_TYPE = 'STEP'
F0 = 1
F1 = 0
T0 = 1
T1 = 2
```

The STEP function generates a continuous connecting function between the given pair of points ( $T_0, F_0$  and  $T_1, F_1$ ).

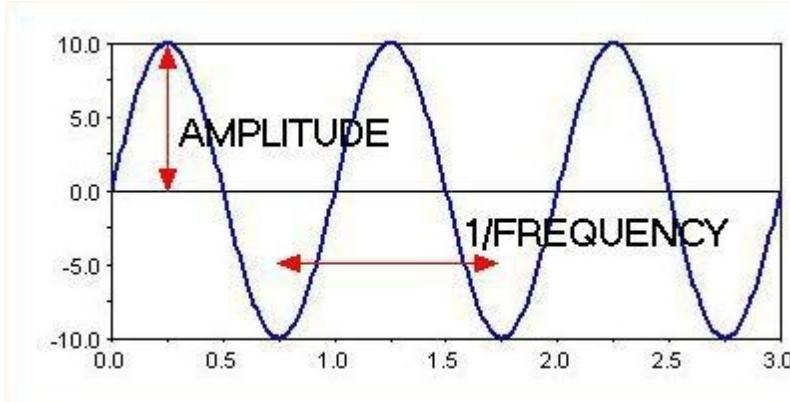


## SINE

Sine example:

```
TOE_FUNCTION_TYPE = 'SINE'
AMPL = 10
FREQ = 1
PHASE = 0
```

The SINE function is described using the three basic parameters: amplitude, frequency and phase. The phase units depends on the units defined for angles while frequency is measured in 1/time (leading to hertz when time unit is second)

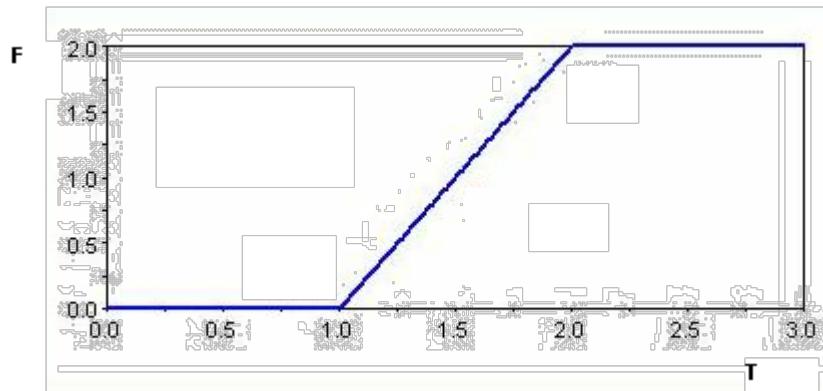


## RAMP

Step Example:

```
PRELOAD_FUNCTION_TYPE = 'RAMP'
F0 = 1
F1 = 0
T0 = 1
T1 = 2
```

The RAMP function generates a linear connecting function between the given pair of points (T0,F0 and T1,F1).



## USER

User Function example.

It allows to specify a user defined function of time using a spline.

```
PRELOAD_FUNCTION_TYPE = 'USER'
SPLINE_NAME = 'FZMAP'
```

Name of the block which contains the user spline data.

The format for the user spline block is the following:

**[FZMAP]**

Block Name

**X\_DATA = 'TIME'**

Independent map data type.

**Y\_DATA = 'FORCE'**

Dependent map data type. The Y\_DATA keys support these options:

FORCE

ANGLE

ANGULAR\_VELOCITY

```
(VALUES)
{x y}
0 1000
0.1 1000
0.2 1000
0.3 1000
0.4 1000
0.5 1000
0.6 1000
0.7 1000
0.8 1000
0.9 1000
1.0 1000
1.1 1000
1.2 1000
1.3 1000
1.4 1000
1.5 1000
1.6 1000
1.7 1000
1.8 1000
1.9 1000
2.0 1000
2.1 1000
```

## 5.8.7 VTT Examples

## Longitudinal Slip

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION =1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
PRELOAD_CONSTANT = 3500
WHEEL_INERTIA = 0.15 $ [OPTIONAL]
TIRE_PROPERTY_FILE = 'pac2002_235_60R16.tir'
ROAD_DATA_FILE = '' $ [OPTIONAL]
CAMBER_ANGLE = 0
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.001
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]
$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'tt_long'
INITIAL_TIME = 0 $ [OPTIONAL]
OUTPUT_STEP = 0.01
END_TIME = 2
VELOCITY = 27.7777778
$ Parameters specific for the selected analysis type
TYPE = 'LONGITUDINAL'
```

## Lateral Slip

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION =1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
PRELOAD_CONSTANT = 3500
WHEEL_INERTIA = 0.15 $ [OPTIONAL]
TIRE_PROPERTY_FILE = 'pac2002_235_60R16.tir'
ROAD_DATA_FILE = '' $ [OPTIONAL]
CAMBER_ANGLE = 0
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.0005
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]
$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'tt_lat_slip'
INTIAL_TIME = 0 $ [OPTIONAL]
OUTPUT_STEP = 0.01
END_TIME = 2
VELOCITY = 27.7777778
$ Parameters specific for the selected analysis type
TYPE = 'LATERAL'
TOE_AMPLITUDE = 10
```

## Combined

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION =1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
WHEEL_INERTIA = 0.15 $ [OPTIONAL]
PRELOAD_CONSTANT = 3500
TIRE_PROPERTY_FILE = 'pac2002_235_60R16.tir'
ROAD_DATA_FILE = '' $ [OPTIONAL]
CAMBER_ANGLE = 0
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.001
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]
$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'tt_combined'
INTIAL_TIME = 0 $ [OPTIONAL]
OUTPUT_STEP = 0.01
END_TIME = 2
VELOCITY = 27.7777778
$ Parameters specific for the selected analysis type
TYPE = 'COMBINED'
TOE_AMPLITUDE = 10
LONG_SLIP = 5
```

## Cornering Stiffness

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION = 1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
PRELOAD_CONSTANT = 3500
WHEEL_INERTIA = 0.15 $ [OPTIONAL]
TIRE_PROPERTY_FILE = 'pac2002_235_60R16.tir'
ROAD_DATA_FILE = '' $ [OPTIONAL]
CAMBER_ANGLE = 0
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.0005
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]
$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'tt_corn_stiff'
INTIAL_TIME = 0 $ [OPTIONAL]
OUTPUT_STEP = 0.01
END_TIME = 2
VELOCITY = 27.7777778
$ Parameters specific for the selected analysis type
TYPE = 'CORNERRING_STIFFNESS'
TOE_VALUE = 1
(PRELOAD)
PRELOAD_FUNCTION_TYPE = 'STEP'
T_0 = 0.1
T_1 = 1.9
F_0 = -2000
F_1 = 2000
```

**Free Input**

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION = 1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
PRELOAD_CONSTANT = 3500
WHEEL_INERTIA = 0.2 $ [OPTIONAL]
TIRE_PROPERTY_FILE = 'pac2002_235_60R16.tir'
ROAD_DATA_FILE = '' $ [OPTIONAL]
CAMBER_ANGLE = 0
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.0001
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]
$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'tt_free_input_2'
INTIAL_TIME = 0 $ [OPTIONAL]
OUTPUT_STEP = 0.01
END_TIME = 2
VELOCITY = 27.7777778
$ Parameters specific for the selected analysis type
TYPE = 'FREE_INPUT'
PRESCRIBED_ROTATION = 'TRUE' $ Allowed values: TRUE, FALSE
(Toe)
TOE_FUNCTION_TYPE = 'SINE'
AMPL = 10
FREQ = 1
PHASE = 0
(PRELOAD)
PRELOAD_FUNCTION_TYPE = 'USER'
SPLINE_NAME='FZMAP'
(WHEEL_OMEGA)
WHEEL_OMEGA_FUNCTION_TYPE = 'USER'
SPLINE_NAME = 'OMEGAMAP'
$-----FZMAP
[FZMAP]
X_DATA = 'TIME'
Y_DATA = 'FORCE'
(VALUES)
{x y}
```

```
0 1000
0.1 1000
0.2 1000
0.3 1000
0.4 1000
0.5 1000
0.6 1000
0.7 1000
0.8 1000
0.9 1000
1.0 1000
1.1 1000
1.2 1000
1.3 1000
1.4 1000
1.5 1000
1.6 1000
1.7 1000
1.8 1000
1.9 1000
2.0 1000
2.1 1000
$-----[OMEGAMAP]
[X_DATA = 'TIME'
Y_DATA = 'ANGULAR_VELOCITY'
(VALUES)
{x y}
0 5000
0.1 5000
0.2 5000
0.3 5000
0.4 5000
0.5 5000
0.6 5000
0.7 5000
0.8 5000
0.9 5000
1.0 5000
1.1 5000
1.2 5000
1.3 5000
1.4 5000
1.5 5000
1.6 5000
1.7 5000
1.8 5000
1.9 5000
2.0 5000
```

## Six Dof

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION =1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
TIRE_PROPERTY_FILE = 'mdids://carrealtime_shared/tires.tbl/p96f.tir'
ROAD_DATA_FILE = 'mdids://carrealtime_shared/roads.tbl/flat.rdf'
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.0001
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]

$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'p96f_free'
OUTPUT_STEP = 0.001
END_TIME = 1.0
$ Parameters specific for the selected analysis type
TYPE = 'SIX_DOF'
XML_OUTPUT=1
CSV_OUTPUT=1
TAB_OUTPUT=1
PI_OUTPUT=1
MAT_OUTPUT=1

$-----SIX_DOF
[SIX_DOF]
IXX = 10
IYY = 1
IZZ = 10
MASS = 380
Z = 0.275
VX = 5
VY = 1
VZ = 0
WX = 0
WY = 0
WZ = 0
```

**Z Locked**

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE ='vtt'
FILE_VERSION =1.0
FILE_FORMAT ='ASCII'
$-----UNITS
[UNITS]
LENGTH ='meter'
FORCE ='newton'
ANGLE ='degrees'
MASS ='kg'
TIME ='second'
$-----MODEL
$ This block contains model parameters
[MODEL]
TIRE_PROPERTY_FILE = 'mdids://carrealtime_shared/tires.tbl/p96f.tir'
ROAD_DATA_FILE = 'mdids://carrealtime_shared/roads.tbl/flat.rdf'
$-----SYSTEM
$ This block contains system settings
[SYSTEM]
INTEGRATION_STEP = 0.0001
STATIC_MAX_TIME = 100 $ [OPTIONAL]
BUFFER_OUTPUT = 'TRUE' $ [OPTIONAL]

$-----ANALYSIS
$ This block contains simulation settings and analysis type parameters
[ANALYSIS]
$ Parameters common to all analysis types
OUTPUT_PREFIX = 'p96f_free'
OUTPUT_STEP = 0.001
END_TIME = 1.0
$ Parameters specific for the selected analysis type
TYPE = 'SIX_DOF'
XML_OUTPUT=1
CSV_OUTPUT=1
TAB_OUTPUT=1
PI_OUTPUT=1
MAT_OUTPUT=1

$-----SIX_DOF
[SIX_DOF]
IXX = 10
IYY = 1
IZZ = 10
MASS = 380
Z = 0.275
VX = 5
VY = 1
VZ = 0
WX = 0
WY = 0
WZ = 0

```

## 5.9 RDF

VI-Road model is a road geometry model that is compatible with the standard tire provided with Adams Car. It can be used to describe analytical, measured, extruded or meshed roads for handling maneuvers. It provides basic path data for driver algorithms. The description in this document covers all the features of the vi-road modeling techniques.

The road model is normally defined via input data ASCII file, based on teim orbit format. The file contains a set of common and specific data blocks. The specific data blocks may vary according to the road geometry description method and global options.

Some model variants support also binary file format for improved I/O performance or data protection.

### 5.9.1 Common Blocks

All common blocks are located in the first part of the file.

The mandatory blocks are identified with the following names:

- [MDI\\_HEADER](#)
- [UNITS](#)
- [MODEL](#)
- [GLOBAL](#)
- [WIDTH](#)
- [FRICTION](#)
- [ROAD\\_DATA](#)

#### MDI\_HEADER Block

Supports general file information, like file type version and format. Usually all comments on the road model are stored here.

Attributes for **MDI\_HEADER** block:

##### FILE\_TYPE

file type identification string, must be always 'rdf' for VI-Road roads.

##### FILE\_VERSION

file version number, should be 7.0 or higher.

##### FILE\_FORMAT

basic format, only 'ASCII' supported.

##### (COMMENTS)

the comments attribute is followed by the {comment\_string} in the next line, then by any number of comments rows enclosed between ' ' delimiters.

Example of **MDI\_HEADER** block:

```
[MDI_HEADER]
FILE_TYPE = 'rdf'
FILE_VERSION = 12.0
FILE_FORMAT = 'ASCII'
```

## UNITS Block

Stores data units information.

Attributes for **UNITS** block:

### LENGTH

length units

### ANGLE

angle units

### FORCE

force units

### MASS

mass units

### TIME

time units

See ADAMS manuals for supported units keys.

Example of **UNITS** block:

```
[UNITS]
LENGTH = 'meter'
ANGLE = 'radian'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
```

## MODEL Block

Contains road model specific information.

Attributes for **MODEL** block:

### METHOD

road module model definition, must be set as 'USER'

### FUNCTION\_NAME

define the road subroutine name. It should be `vitoools::vi_road`.

### TYPE

road model type. Can be 'ANALYTIC', 'EXTRUSION', 'FLAT', 'MEASURED', 'MESH', 'COMPOSITE', 'EXTERNAL'.

The [ANALYTIC](#) model addresses analytical description of the road plant path, vertical height, and bank angle by means of geometrical segments (lines, clothoids, arcs, step functions).

The [MEASURED](#) road model is based on X, Y, Z coordinates of its centerline, plus bank angle values at each point.

## Appendix B: File Format

The [EXTRUSION](#) is defined by a centerline and a section (in this release the number of points per section should be constant).

The [MESH](#) road is a polygon based road, defined by nodes coordinates and elements.

The [COMPOSITE](#) road is used to modify certain values of a referenced rdf.

The [OPEN CRG](#) road is a wrapper to the actual OpenCRG road data file.

The [EXTERNAL](#) model is designed to interface VI-CarRealTime solver to packages featuring their own road environment. This kind of RDF is not meant to be loaded in VI-Road graphical interface.

### MODE

any of the road models may have different sub-modeling options, addressed by this key. Standard ('STANDARD') and smooth ('SMOOTH') modes are currently supported. The second option causes road track plant path transformation by means of smooth spline algorithm. The smooth mode can be used only with open measured road models.

### FORMAT

Road data format, only 'STANDARD' (standard) is supported.

### OPERATING\_MODE

This key is present in EXTERNAL roads. Its default value is 1. It means that an auxiliary plane is computed runtime for each wheel. The plane equation is carried out by using contact point coordinates and its direction cosines. When OPERATING\_MODE value is set to 0 the input contact point locations are directly used without any additional computation.

### LEGACY\_MODE

This key should be used to switch back to older VI-ROAD single-thread (STD)/single-point-contact (SPC) computations results. It is required to match single and multi-thread output for those tire/road models combinations supporting multi-thread (MTD)/multiple-points-contact (MPC). The new VI-ROAD interface for MTD/MPC take into account additional parameters (like the basic tire geometry) to improve results reliability and stability, but the output may differ from the older (STI/TYDEX based) interface. The differences may become noticeable specially if the tire-road contact is located outside the road boundaries (off-road contact) or if the contact itself is intermittent (flying tire).

Supported values are:

0 = no legacy (output aligned with latest road computations interface). This is the default value.

1 = output aligned with CRT v17 road interface

The key is taken into account only by road models supporting MTD/MPC interface (i.e. FLAT, MESH, EXTERNAL, COMPOSITE).

All the other road models rely on the older interface in any case.

Example of MODEL block:

```
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'vitoools::vi_road'
TYPE = 'ANALYTIC'
MODE = 'STANDARD'
FORMAT = 'STANDARD'
```

## GLOBAL Block

Contains all road model global values and options.

Attributes for **GLOBAL** block:

### OUTBOUND\_SAFE

when set to `TRUE`, in the case of the vehicle going off the road, the road is widened by the solver to maintain the contact patch on the road (if the road is Analytic or Measured), or the z-value used is the last z calculated when the vehicle was still on the road, and this z-value is used until the vehicle goes back on the road (for Mesh, Surfmesh and Extrusion roads).

If the vehicle is going off the road because the road is finished, if this key is set to `TRUE`, then for all the roads it's used the last valid z-value, otherwise the z-value becomes immediately equal to 0.

### EVENT\_CHECK

when it's set to `TRUE`, warning messages regarding the road (for example "going off the road" ...) will be displayed.

### DISTANCE\_CALC

in `ANALYTIC` roads, modify the `DISTANCE_CALC` key to '`DYNAMIC_STEP`' to have the solver will use a non standard point-to-road centerline distance calculation method: this method uses a variable discretization step, depending on the particular road characteristics.

With `MESH` roads setting the `DISTANCE_CALC` key to '`AUTO`' will activate a geometry dependent point-to-road surface algorithm able to better detect and discriminate overlapped elements areas.

### SAFE\_CALC

when `TRUE` performs the computation for analytic and measured roads looking for the closest centerline point among all the available ones. This functionality is designed for debugging purposes only.

### FRICITION

specifies if the friction will be defined in '`STANDARD`' mode ([FRICITION](#) block) or using a '`TABLE`' ([FRICITION\\_TABLE](#) block). In addition the option `GLOBAL_TABLE` is supported. In this case no left/right friction should be specified (table has only s and friction columns)

### IRREGULARITIES

specifies weather or not irregularities have to be added to the basic (flat) road surface. Analytic (PSD) and measured (STANDARD) irregularities are currently supported ('`ANALYTIC_PSD`' and '`MEASURED_STD`' key respectively).

### LEFT\_EDGE

defines left road edge geometry type (if any). Only kerb geometry is currently supported ('`KERB`' key).

### RIGHT\_EDGE

defines right road edge geometry type (if any). Only kerb geometry is currently supported ('`KERB`' key).

### MESH\_CHECK

if set to `TRUE` the mesh is checked to validate it. This key is present only in `EXTRUSION`, `SURFMESH` and `MESH` road models.

## Appendix B: File Format

**MESH\_LOOKUP\_CELL\_SIZE**

the average of the road elements first lookup grid for SURFMESH and MESH road models. It depends on the mesh elements average size, and may affect computational speed and overall memory allocation. The default size of 1 (one) meter works fine for the most of the mesh road geometries, but in some cases (e.g. elements average size bigger than cell size) increasing it may improve performances.

**VG****ORIGIN\_X****ORIGIN\_Y****ORIGIN\_Z**

define the road starting point with respect to global reference.

**ORIGIN\_A3**

define the road z-rotation

- 0 degrees => x positive,
- 180 degrees => x negative

**DISCRETIZATION\_STEP**

the default value is 0.5 meters. Use it to reduce the step if the simulation needs it.

**SMOOTHING\_TIME**

generate an initial smoothing in computed road height (Z) in order to avoid failures with static solution. The parameter define the duration of the smoothing function. This feature applies to analytic, measured as well as meshed roads.

**CP\_COMPUTATION\_MODE**

This flag is new in version 16 and is required to restore the v15 contact patch computation method. In v16 a more precise algorithm has been introduced to better support complex road shapes. When the CP\_COMPUTATION\_MODE=0 the road file will behave as in v15.

**MATERIALS\_MODE**

defines the materials definition mode for MESH road model. To activate the [MATERIALS](#) data table for the MESH [ELEMENTS](#) the value must be set to 1 (default value is 0).

Example of **GLOBAL** block :

```
[GLOBAL]
OUTBOUND_SAFE = 'TRUE'
EVENT_CHECK = 'TRUE'
DISTANCE_CALC = 'STANDARD'
FRICTION = 'STANDARD'
IRREGULARITIES = 'FLAT'
LEFT_EDGE = 'KERB'
RIGHT_EDGE = 'FLAT'
ORIGIN_X = 0
ORIGIN_Y = 0
ORIGIN_Z = 0
ORIGIN_A3 = 3.141592653589793
```

## WIDTH Block

Defines road left and right width values.

Attributes for WIDTH block:

### **LEFT\_SIDE\_WIDTH**

road width from the left of centreline.

### **RIGHT\_SIDE\_WIDTH**

road width from the right of centreline.

Example of **WIDTH** block:

```
[WIDTH]
LEFT_SIDE_WIDTH = 5.0
RIGHT_SIDE_WIDTH = 5.0
```

## WIDTH\_TABLE Block

The Width Table lets defining different road width values along the road.

**Note:** if both **WIDTH** and **WIDTH\_TABLE** blocks are present, **WIDTH\_TABLE** values override the ones defined in **WIDTH** Block.

Example of **WIDTH\_TABLE** block:

```
[WIDTH_TABLE]
{ track_s width }
0 10
10 10
11 9
20 9
30 10
40 10
```

## FRICTION Block

Stores road left and right side friction values.

Attributes for FRICTION block:

### **LEFT\_SIDE\_MU**

road left side friction.

### **RIGHT\_SIDE\_MU**

road right side friction.

Example of **FRICTION** block:

```
[FRICTION]
LEFT_SIDE_MU = 1.0
RIGHT_SIDE_MU = 1.0
```

Different friction values can be set as function of track S.

See Optional Block [FRICTION\\_TABLE](#) for further informations.



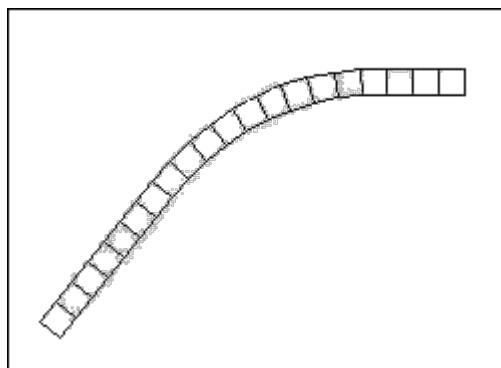
## 5.9.2 Optional Blocks

Description of optional block follows. Those data blocks may exist or not, depending on the road model or global options.

- [PLANT\\_PATH](#)
- [VERTICAL PATH](#)
- [BANK ANGLE PATH](#)
- [MEASURED CENTERLINE](#)
- [IRREGULARITIES](#)
  - [LEFT\\_PSD\\_TABLE](#)
  - [RIGHT\\_PSD\\_TABLE](#)
  - [PSD\\_COHERENCY\\_TABLE](#)
  - [PSD\\_PHASE\\_TABLE](#)
- [MEASURED IRREGULARITIES](#)
- [LEFT\\_KERB\\_GEOMETRY](#)
- [RIGHT\\_KERB\\_GEOMETRY](#)
- [FRICTION TABLE](#)
- [CENTERLINE\\_PROPERTIES](#)
- [SECTION\\_PROPERTIES](#)
- [SECTION](#)
- [NODES](#)
- [ELEMENT](#)
- [TRACK\\_PATH](#)

### PLANT\_PATH Block

Contains the analytical description of road plant path (XY plane). The block consists in a table reporting path curvature values as function of track abscissa (S). Each couple of rows defines an analytical segment of the road. Zero curvature segments are interpreted as lines, and constant curvature segments as arcs. When curvature changes from one segment to another clothoid geometry is used, while zero curvature segments are interpreted as lines, and constant curvature segments as arcs. From the above description should be clear that any change in direction is achieved by mean of clothoid segments.



Example of **PLANT\_PATH** block:

```
[PLANT_PATH]
{ s curvature }
 0.0 0.0
 20.0 0.0
 50.0 0.01
 100.0 0.01
 150.0 0.0
 200.0 0.0
```

## Appendix B: File Format

In the above example (supposing the length units are meters) the road path starts with a straight segment of 20m, then clothoid (spiral) geometry allows smooth curvature change from 0.0 to 0.01 ( $R=100.0\text{m}$ ) within thirty meters (20-50). From 50.0 to 100.0 meters the road is represented by a constant radius arc of 100m. Then the road path is smoothed again to reach zero curvature. The final fifty (150-200) meters are defined as a straight line.

**VERTICAL\_PATH Block**

Supports analytical description of road vertical path. The block consists in a table reporting path vertical height values as function of track abscissa (S). Each couple of rows defines an analytical segment of the road Z coordinate. Different heights are connected by mean of STEP5 function segments

Example of **VERTICAL\_PATH** block :

```
[VERTICAL_PATH]
{ s height }
 0.0 0.0
 200.0 0.0
```

This block is taken into account only for analytical road models.

**BANK\_ANGLE\_PATH Block**

Stores description of the road bank angle path. As the previous analytical path blocks this one consists in a table reporting road bank angle values as function of track abscissa (S). Each couple of rows defines an analytical segment of the bank angle. Different angular values are connected by mean of STEP5 function segments

Example of **BANK\_ANGLE\_PATH** block :

```
[BANK_ANGLE_PATH]
{ s angle }
 0.0 0.0
 200.0 0.0
```

This block is taken into account only for analytical road models.

**MEASURED\_CENTERLINE Block**

Defines measured data road track. It consists of four columns table reporting X, Y, Z, and bank angle at each road centerline point. In case of closed roads the last measured point must not be superimposed to the first one. In other words closing the path must be left to ARM algorithm.

Example of **MEASURED\_CENTERLINE** block:

```
[MEASURED_CENTERLINE]
{ x_coord y_coord z_coord bank_angle }
 +20.000000 +0.000000 +0.000000 +0.000000
 +15.000000 +0.000000 +0.000000 +0.000000
 +10.000000 +0.000000 +0.000000 +0.000000
 +5.000000 +0.000000 +0.000000 +0.000000
 +0.000000 +0.000000 +0.000000 +0.000000
 -5.000000 +0.000000 +0.000000 +0.000000
 -10.000000 +1.000000 +0.100000 +0.000000
 -15.000000 +2.000000 +0.200000 +0.000000
 -20.000000 +3.000000 +0.300000 +0.000000
 -25.000000 +4.000000 +0.400000 +0.000000
```

## IRREGULARITIES Block

This block defines road irregularities global data. These parameters are used only if IRREGULARITIES option in the GLOBAL block is active (set to 'ANALYTIC' or 'MEASURED').

Attributes for IRREGULARITIES block:

### **MODEL**

defines irregularities type. Supported options are PSD for pseudo-random analytic algorithm or MEASURED\_STD for discrete measured irregularities .

**Note:** the pseudo random numbers generated are different in linux platform, but the FFT obtained has the same trend in frequencies.

### **STARTING\_S**

defines where the irregularities start, considering road abscissa S. Default value is 0.0 (from the beginning of the road). Note that irregularities are active also during statics analysis so the vehicle may start dynamics simulation in unexpected position if STARTING\_S= 0. Use [SMOOTHING\\_TIME](#) to start analysis on a flat road surface (e.g. SMOOTHING\_TIME = 1.0).

### **ENDING\_S**

defines where the irregularities end, considering road abscissa S. Default value is full road length. Note that irregularities are active also during statics analysis so the vehicle may start dynamics simulation in unexpected position on closed roads if ENDING\_S is greater or equal to the actual road length (due to the fact the rear wheel are placed on irregularities). Use [SMOOTHING\\_TIME](#) to start analysis on a flat road surface (e.g. SMOOTHING\_TIME = 1.0)

### **SCALE**

scaling factor for all irregularities. It multiplies the final DZ value in time domain.

### **RNG\_ALGORITHM**

defines the pseudo-random number generator (RNG) algorithm, affecting the analytic irregularities final results. Acceptable values are 0 and 1. The default value is 1, meaning the improved RNG algorithm is used.

### **FFT\_POINTS**

points for PSD resolution.

### **PHASE**

left to right PSD signal phase.

### **PHASE\_TABLE**

defines weather or not a special table for PSD phase is provided.

**Note:** such table is not available for meshed road.

### **COHERENCY**

left to right PSD signal coherency. Currently its value is used just to activate (1.0) or deactivate (0.0) the actual coherency.

### **OMEGA\_1**

PSD shape min omega value (W1).

## Appendix B: File Format

**OMEGA\_2**

PSD shape max omega value (W2).

**LEFT\_PSD\_O1**

left side PSD value at min omega PSD(W1). Unit is [ $L^3$ ]

**LEFT\_PSD\_O2**

left side PSD value at min omega PSD(W2). Unit is [ $L^3$ ]

**LEFT\_PSD\_TABLE**

defines weather or not a special table for left side PSD values is provided. ('YES' implies the existence of [LEFT\_PSD\_TABLE] additional block).

**Note:** such table is not available for meshed road.

**RIGHT\_PSD\_O1**

right side PSD value at min omega PSD(W1). Unit is [ $L^3$ ]

**RIGHT\_PSD\_O2**

right side PSD value at min omega PSD(W2). Unit is [ $L^3$ ]

**RIGHT\_PSD\_TABLE**

defines weather or not a special table for right side PSD values is provided. ('YES' implies the existence of [RIGHT\_PSD\_TABLE] additional block).

**Note:** such table is not available for meshed road.

**Note:** for vehicle with left and right tire, the left PSD values are applied to the left front and rear tire (the values will be different because the s for front and rear tire is different), and the right values are applied to the right front and rear tire; for vehicle with only one front and one rear tire ( motorcycle applications), only the left PSD are applied to the front and rear tire ( right PSD is not used in this case )

Example of **IRREGULARITIES** block:

```
[IRREGULARITIES]
MODEL = 'ANALYTIC_PSD'
STARTING_S = 100
ENDING_S = 1000
SCALE = 1
FFT_POINTS = 128
PHASE = 5
PHASE_TABLE = 'YES'
COHERENCY = 1.0
OMEGA_1 = 0.1
OMEGA_2 = 32
LEFT_PSD_O1 = 0.0003
LEFT_PSD_O2 = 3e-009
LEFT_PSD_TABLE = 'YES'
RIGHT_PSD_O1 = 0.0003
RIGHT_PSD_O2 = 3e-009
RIGHT_PSD_TABLE = 'YES'
```

## Theory

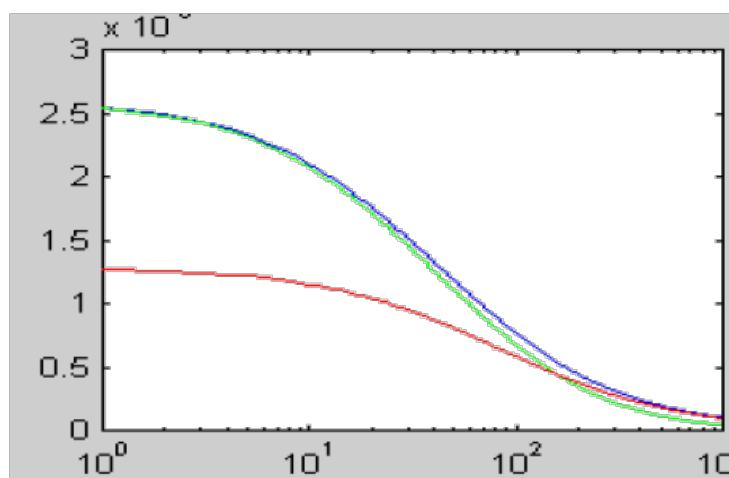
### Road disturbances

The PSD equation that describes the frequency domain **Power Spectral Density** of the disturbances in vertical direction is of the type:

$$\text{PSD}(i) = (A * \Omega_c^2) / ((\text{Freq}(i)^2 + \Omega_r^2) * (\text{Freq}(i)^2 + \Omega_c^2));$$

where:

|            |                                            |
|------------|--------------------------------------------|
| A          | PSD formula Amplitude coefficient          |
| $\Omega_c$ | PSD formula coefficient for high frequency |
| $\Omega_r$ | PSD formula coefficient for low frequency  |
| Freq(i)    | PSD independent frequency array            |
| PSD(i)     | PSD output values array (returned)         |



The typical shape of PSD function is a semilog plot as is shown above. It's possible to change the curve shape by modifying A,  $\Omega_r$  and  $\Omega_c$ .

The inverse PSD equation to get the DZ(t) from the above PSD equation is of the type:

$$DZ(t) = \text{SUM}((2 * \text{SQRT}(DF * PSD(Fi)) * \text{COS}(S(t) * Wi' + Phi'))$$

where:

|      |                                                                                                     |
|------|-----------------------------------------------------------------------------------------------------|
| DF   | frequency delta, defined as $(\Omega_c - \Omega_r) / (\text{FFTpoints} - 1)$ , in [1/length]        |
| Fi   | frequency values array, defined as $\Omega_r + i * DF$ [ $i=0, \text{FFTPOINTS}-1$ ], in [1/length] |
| S(t) | track arc length, relative track_S[0:(S_end - S_0)], in length unit                                 |
| X    | amplitude random values array                                                                       |
| Y    | phase random values array, random[FFTPOINTS], adimensional                                          |

and:

$$\begin{aligned} Wi' &= 2\pi * (Fi + X * DF) &<< X=\text{rnd value } [-Fmin, Fmax] >> \\ Phi' &= 2\pi * Y &<< Y=\text{rnd value } [Phmin, Phmax] >> \end{aligned}$$

The computed vertical disturbance function of time ( $DZ(t)$ , [mm]) has to be added to the 3D road ideal profile height ( $Z(t)$ ), which is given by the projection of the tire contact point onto the 3D (or 2D) road surface during the dynamic simulation. The inverted PSD (or  $DZ(t)$ , see above) road disturbance will be applied independently to each of the four wheels. The front and rear wheel disturbances will be delayed by a time (or distance) lag corresponding to the wheel base.

## Appendix B: File Format

1. Different disturbances profiles could be defined to act under beneath each of the four wheels, and
2. left and right tires could be given a different random seed if same disturbance profile is used, in order to avoid deterministic (same amplitude and phase) excitation.

Front/Rear are delayed by the vehicle base, and each wheel gets a different disturbance due to the RANDOM functions applied to the frequencies and phases while inverting the FFT function.

The MEASURED irregularities simply computes the given (measured) road Z disturbances as function of road track centerline coordinate (S). The values are linearly interpolated with S and can be defined separately for the left and right tires.

### Sample

Extract from a data file with simple PSD trapezoidal shaped irregularities:

```
[IRREGULARITIES]
MODEL = 'PSD'
FFT_POINTS = 512
SCALE = 1.0
OMEGA_1 = 1.0
OMEGA_2 = 32.0
LEFT_PSD_O1 = 1.0E-4
LEFT_PSD_O2 = 1.0E-8
LEFT_PSD_TABLE = 'NO'
RIGHT_PSD_O1 = 1.0E-4
RIGHT_PSD_O2 = 1.0E-8
RIGHT_PSD_TABLE = 'NO'
COHERENCY = 1.0
PHASE = 20.0
PHASE_TABLE = 'NO'
```

- at OMEGA\_1 = 1 rad/m excitation has a PSD 1.E-4 (i.e. 0.01m\*\*2)
- at OMEGA\_2 = 32 rad/m excitation has a PSD 1.E-8 (i.e. 0.0001m\*\*2)

**Note:** in this case there are no OMEGA tables: LEFT\_PSD\_TABLE = 'NO' and RIGHT\_PSD\_TABLE = 'NO'

To convert those rad/m frequencies into Hz multiply them by the speed of the car in m/s and divide them by  $2\pi$ .

| Speed  | Disturbance amplitude | Frequency                      |
|--------|-----------------------|--------------------------------|
| 50 m/s | 10 mm                 | $50 * 1 / 2\pi$ = about 8Hz    |
| 50 m/s | 0.1 mm                | $50 * 32 / 2\pi$ = about 255Hz |

PSD tables for phase, if present, would be used to apply different disturbances to L/R wheels.

### LEFT\_PSD\_TABLE Block

Defines left side irregularities PSD values as function of omega. Omega max and min are still those specified in the [IRREGULARITIES] data block. In case the omega data provided by in the table do not cover all the range specified by OMEGA\_1 and OMEGA\_2, the extreme value of the phase data is used for extrapolation.

```
[LEFT_PSD_TABLE]
{ omega psd }
1.2271846e-001 3.4588644e-004
2.4543693e-001 8.6764694e-005
3.6815539e-001 4.1614292e-005
```

```
.....
.....
3.1170490e+001 4.4855219e-009
3.1293208e+001 4.4850076e-009
3.1415927e+001 2.2424181e-009
```

**Note:** such table is not available for meshed road.

### RIGHT\_PSD\_TABLE Block

Defines right side irregularities PSD values as function of omega. Omega max and min are still those specified in the [IRREGULARITIES] data block. In case the omega data provided by in the table do not cover all the range specified by OMEGA\_1 and OMEGA\_2, the extreme value of the phase data is used for extrapolation.

```
[RIGHT_PSD_TABLE]
{ omega psd }
1.2271846e-001 3.5787581e-004
2.4543693e-001 8.4376497e-005
3.6815539e-001 3.9796489e-005
.....
.....
3.1170490e+001 4.5170710e-009
3.1293208e+001 4.5165552e-009
3.1415927e+001 2.2581824e-009
```

**Note:** such table is not available for meshed road.

### PSD\_COHERENCY\_TABLE Block

This data section is deprecated since v15.0 and no more used.

Defines left-to-right side irregularities PSD values coherency (as function of omega). Omega max and min are still those specified in the [IRREGULARITIES] data block. In case the omega data provided by in the table do not cover all the range specified by OMEGA\_1 and OMEGA\_2, the extreme value of the phase data is used for extrapolation.

```
[PSD_COHERENCY_TABLE]
{ omega coherency }
0.1 0.99
1.0 0.99
2.0 0.98
3.0 0.97
4.0 0.96
5.0 0.95
6.0 0.90
7.0 0.88
8.0 0.88
9.0 0.86
10.0 0.77
15.0 0.62
20.0 0.58
25.0 0.35
30.0 0.35
32.0 0.70
```

## PSD\_PHASE\_TABLE Block

Defines left-to-right side irregularities PSD values phase (as function of omega). Omega max and min are still those specified in the [IRREGULARITIES] data block. In case the omega data provided by in the table do not cover all the range specified by OMEGA\_1 and OMEGA\_2, the extreme value of the phase data is used for extrapolation.

```
[PSD_PHASE_TABLE]
{ omega phase }
 0.1 0.0
 1.0 0.0
 2.0 1.0
 3.0 1.5
 4.0 2.0
 5.0 2.5
 6.0 2.0
 7.0 0.0
 8.0 2.0
 9.0 3.0
10.0 4.0
15.0 -2.0
20.0 0.0
25.0 0.0
30.0 0.0
32.0 -2.0
```

**Note:** such table is not available for meshed road.

## MEASURED\_IRREGULARITIES

Defines left-to-right side irregularities MEASURED values.

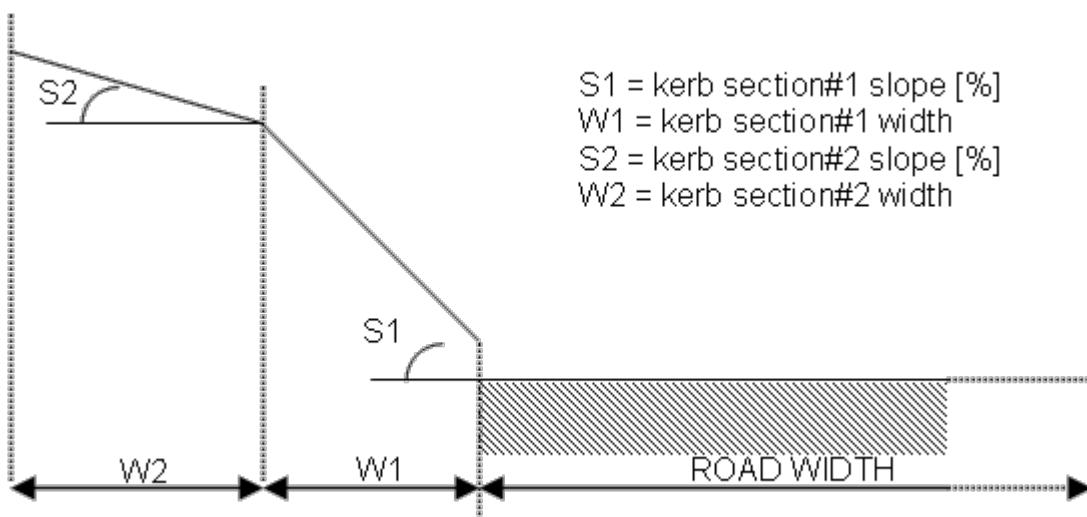
```
[MEASURED_IRREGULARITIES]
{ s left_z right_z }
 0.0 0.0 0.0
100.0 0.0 0.0
100.1 0.01 0.0
100.2 0.02 0.0
100.3 0.03 0.0
100.4 0.04 0.0
100.5 0.05 0.0
100.6 0.04 0.0
100.7 0.03 0.0
100.8 0.02 0.0
100.9 0.01 0.0
101.0 0.0 0.0
1000.0 0.0 0.0
```

The above example represents a small obstacle of one meter long (5cm height)and placed on the left side of the road, starting at 100m from the road origin.

**Note:** it's important to use a MEASURED\_CENTERLINE block defined with a constant discretization.

## LEFT\_KERB\_GEOMETRY Block

Describes left kerb edge geometry. Slope, width and friction values for two kerb section as fun of track S are reported. This block must be defined if LEFT\_EDGE attribute in the GLOBAL block is set equal to 'KERB'.



Example of **LEFT\_KERB\_GEOMETRY** block:

```
[LEFT_KERB_GEOMETRY]
{ track_s slope_1[%] width_1 friction1 slope_2[%] width_2 friction2 }
 0.0 20.0 1.0 1.0 0.0 1.0 1.0
 100.0 20.0 1.0 1.0 0.0 1.0 1.0
```

### RIGHT\_KERB\_GEOMETRY Block

Describes right kerb edge geometry. Data are the same as LEFT\_KERB\_GEOMETRY block.

Example of **RIGHT\_KERB\_GEOMETRY** block:

```
[RIGHT_KERB_GEOMETRY]
{ track_s slope_1[%] width_1 friction1 slope_2[%] width_2 friction2 }
 0.0 20.0 1.0 1.0 0.0 1.0 1.0
 100.0 20.0 1.0 1.0 0.0 1.0 1.0
```

### FRICTION\_TABLE Block

Permits to set different frictions as function of track S.

Different left and right friction values, or symmetric ones, can be specified.

The friction values are applied to the road (not to the tire), and left and right values are applied to the left and right part of the road (defined by the centerline), without interpolation.

This feature is not applicable to MESH/SURFMESH and EXTRUSION roads:

- for MESH and SURFMESH roads the friction is specified for each element in the [ELEMENT](#) block.
- for EXTRUSION roads the friction is specified for each node in the section, see the [SECTION\\_PROPERTIES](#) block.

Example of **FRICTION\_TABLE** block with **symmetric** left/right friction values ([FRICTION](#) = 'GLOBAL\_TABLE' ):

```
[FRICTION_TABLE]
{ track_s friction
 0.0 1.0
 10.0 1.0
 100.0 0.1
 200.0 0.3
 500.0 0.3
 600.0 1.0
```

## Appendix B: File Format

```
1000.0 1.0
```

Example of **FRICITION\_TABLE** block with **different left/right** friction values (**FRICITION** = 'TABLE' ):

```
[FRICITION_TABLE]
{ track_s frictionL frictionR }
 0.0 1.0 1.0
 10.0 1.0 1.0
100.0 0.1 0.1
200.0 0.3 1.0
500.0 0.3 1.0
600.0 1.0 1.0
1000.0 1.0 1.0
```

**CENTERLINE\_PROPERTIES Block**

Used in EXTRUSION road types.

```
[CENTERLINE_PROPERTIES]
SEGMENTATION = 'NONE'
```

Discretization of the polyline interpolation:

- **NONE**: no interpolation
- **AUTO**: the interpolation step is automatically calculated
- **DIMENSION**: the step of the interpolation will be the value set in **SEGMENT**
- **STEP**: total number of interpolation steps (the interpolation step will be equal to the curve length divided by the number of steps)

**Note:** INTERPOLATION + SEGMENTATION combination are still experimental in the actual release

```
INTERPOLATION = 'BASIC'
```

Interpolation of the polyline:

- **BASIC**: no interpolation
- **LINEAR**: linear interpolation
- **CUBIC\_SPLINE**: interpolation using a cubic spline
- **QUINTIC\_SPLINE**: interpolation using a quintic spline

**Note:** INTERPOLATION + SEGMENTATION combination are still experimental in the actual release

```
SEGMENT = 0
```

Specifies the interpolation step (to be used in combination with [SEGMENTATION=DIMENSION](#))

**(PROPERTIES)**

```
{ s ori a1 a2 a3 scale1 scale2 friction section }
 0 0 0 0 0 1 1 1 1
 50 0 0 0 0 1 1 1 2
150 0 0 0 0 1 1 1 1
200 0 0 0 0 1 1 1 3
```

**s**

The centerline basic point for the section

**ori**

The orientation mode: 0 = Euler

**a1 a2 a3**

Angles defining the orientation of the section

ori=0 and a1="twist value" can be used to twist the section with respect to the centerline axis if [ABSOLUTE\\_ORIENTATION=FALSE](#) in the SECTION\_PROPERTIES block

a1,a2 and a3 can be used in combination with ABSOLUTE\_ORIENTATION=TRUE in the SECTION\_PROPERTIES block to set an absolute orientation of the section

**Note:** using ABSOLUTE\_ORIENTATION=TRUE could lead to discontinuities: to be used carefully.

**scale1**

Scaling the section values in the u direction

**scale2**

Scaling the section values in the v direction

**friction**

Friction values applied to all the section. It multiplies the [friction](#) value set in the SECTION\_PROPERTIES block.

**section**

Select the number of the section in the section LIST in the SECTION\_PROPERTIES block

## SECTION\_PROPERTIES Block

Used in EXTRUSION road types

The sub-block LIST contains the list of the sections, while the SECTION\_x sub-block

The friction parameter is experimental in the actual release.

```
[SECTION_PROPERTIES]
SEGMENTATION = 'NONE'
```

Discretization of the polyline interpolation:

- NONE: no interpolation
- AUTO: the interpolation step is automatically calculated
- DIMENSION: the step of the interpolation will be the value set in [SEGMENT](#)
- STEP: total number of interpolation steps (the interpolation step will be equal to the curve length divided by the number of steps)

**Note:** INTERPOLATION + SEGMENTATION combination are still experimental in the actual release

```
INTERPOLATION = 'BASIC'
```

Interpolation of the polyline:

- BASIC: no interpolation
- LINEAR: linear interpolation
- CUBIC\_SPLINE: interpolation using a cubic spline
- QUINTIC\_SPLINE: interpolation using a quintic spline

**Note:** INTERPOLATION + SEGMENTATION combination are still experimental in the actual release

```
SEGMENT = 0
```

Specifies the interpolation step (to be used in combination with [SEGMENTATION=DIMENSION](#))

**ABSOLUTE\_ORIENTATION=FALSE**

If set to False (default), the sections are always normal to the centerline, for each arclength.

To twist the section with respect to the centerline axis, be sure to have [ori=0](#), and set the [angle a1](#) to the desired twist value

If set to True, the orientation at the arclength used as a basic point for the section, will be maintained for all the other arclengths belonging to that section. The section will be always centered on the centerline, for each arclength, but it could be not normal to it.

**Note:** using `ABSOLUTE_ORIENTATION=TRUE` could lead to discontinuities: to be used carefully.

**POINTS = 6**

Sets the number of points used to define all the sections

**VARIABLE\_SECTION = 'TRUE'**

True each time in the `LIST` sub block there are multiple lines (even if the same section is repeated)

**LIST Sub-block**

List of the available sections. The order in which they are listed is important because it is used in the `CENTERLINE_PROPERTIES` block

```
(LIST)
{ name }
'SECTION_1'
'SECTION_2'
```

**SECTION Block**

Used in EXTRUSION road types

The `SECTION` block contains the specific section geometry data

```
$-----SECTION_1
[SECTION_1]
{ u v friction }
-5 0 1
-2 0 1
2 0 1
5 0 1
$-----SECTION_2
[SECTION_2]
{ u v friction }
-5 0.1 1
-2 0 1
2 0 1
5 0.1 1
$-----SECTION_3
[SECTION_3]
{ u v friction }
-5 0.3 1
-2 0 1
2 0 1
5 0.3 1
```

**u**

In the plane of the section u is the horizontal abscissa

v

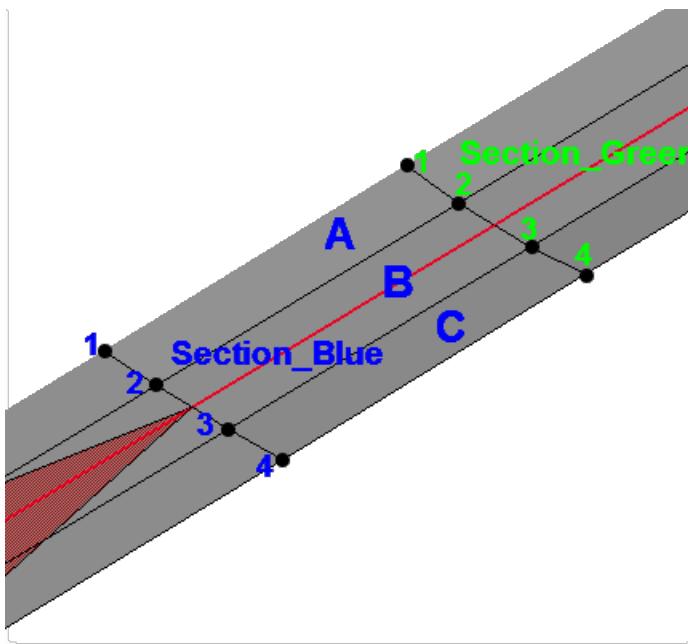
In the plane of the section u is the vertical abscissa

#### **friction**

Local friction for the indicated point. It will be multiplied by the SECTION\_PROPERTIES block [friction](#) value.  
In the **A** region the **friction** value will be:

(Section\_Blue\_friction \* (**blue\_1\_friction** + **blue\_2\_friction**) + Section\_Green\_friction \* (**green\_1\_friction** + **green\_2\_friction**) ) / 4

**Note:** this feature is still experimental



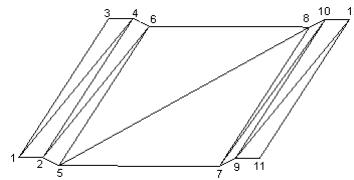
## NODES Block

Used in MESH and SURFMESH road types

In the first column are listed the id node, that will be used in the [ELEMENT](#) block

```
[NODES]
{ id x_coord y_coord z_coord }
 1 0 3 0.2
 2 0 2.4 0.2
 3 400 3 0.2
 4 400 2.4 0.2
 5 0 2 0
 6 400 2 0
 7 0 -2 0
 8 400 -2 0
 9 0 -2.4 0.2
10 400 -2.4 0.2
11 0 -3 0.2
12 400 -3 0.2
```

## Appendix B: File Format



**Note:** for 32 bit system platforms, the number of nodes is limited to 2 million. For 64 bit system platforms, the number of nodes is limited to 10 million.

## ELEMENTS Block

Used in MESH/SURFMESH road types.

In the following ELEMENTS block example the first triangle is defined by nodes 1, 2 and 4, the second by 4, 3 and 1, ... defined in the [NODES](#) block. See [meshed figure](#) in [NODES](#) block topic.

```
[ELEMENTS]
{n1 n2 n3 mu }
1 2 4 1
4 3 1 1
2 5 6 0.9
6 4 2 0.9
5 7 8 0.7
8 6 5 0.7
7 9 10 0.5
10 8 7 1
9 11 12 1
12 10 9 1
```

The MESH road model supports also the grouping of the friction under the more general [MATERIALS](#) table. If the MATERIALS\_MODE is set to the value of 1 (one) the friction parameter (last in the table row) is interpreted as material ID (referencing a specific [MATERIALS](#) table) instead than friction.

The above example in case of a MESH with MATERIALS would be rewritten as:

```
[ELEMENTS]
{n1 n2 n3 mat }
1 2 4 1
4 3 1 1
2 5 6 4
6 4 2 4
5 7 8 2
8 6 5 2
7 9 10 3
10 8 7 1
9 11 12 1
12 10 9 1
```

The associated [MATERIALS](#) table must be available in the file:

```
[MATERIALS]
{id mu }
1 1
2 0.7
3 0.5
4 0.9
```

**Note:** for 32 bit system platforms, the number of elements is limited to 2 million. For 64 bit system platforms, the number of elements is limited to 10 million.

## MATERIALS Block

The MATERIALS data block is used in MESH road types to allow definition of friction values for elements groups. It is activated only if the MATERIALS\_MODE attribute is set to a value of 1 (default is 0). The example below shows a table of four materials with different friction values. The materials ID should be addressed by [ELEMENTS](#) table properties.

```
[MATERIALS]
{id mu }
1 1
2 0.7
3 0.5
4 0.9
```

**Note:** for 32 bit system platforms, the number of materials is limited to 2 million. For 64 bit system platforms, the number of materials is limited to 10 million.

## TRACK\_PATH Block

This block is used together with MESH road model, to associate a path (e.g. centerline) to the mesh geometry. The path is needed to activate certain optional features, like PSD irregularities, over this road model. The path can be defined in two ways, explicit (data values directly written in a table) and by DRD file (file pathname required).

Attributes for **TRACK\_PATH** block:

### TYPE

the path data format type, may be `NUMERIC_DATA` (explicit table) or `DRD_FILE` (path is described by DRD file data).

### REFERENCE

the path reference system type: `RELATIVE` (i.e. relative to the road reference system) or `ABSOLUTE` (the path coordinates are referenced to the global reference system).

The basic difference between the two reference systems definitions is that for `RELATIVE` the path follows the road geometry along its transformations (if any), while no transformation at all is applied in case of `ABSOLUTE` specification. Note that usually the DRD files has data expressed in the global reference system, thus `REFERENCE = 'ABSOLUTE'` has to be used.

NUMERIC\_DATA path geometry sample:

```
[TRACK_PATH]
TYPE = 'NUMERIC_DATA'
REFERENCE = 'RELATIVE'
(TABLE)
{ x_coord y_coord z_coord }
 0.0 0.0 0.0
 10.0 0.0 0.0
 20.0 0.0 0.0
 30.0 0.0 0.0
 40.0 0.0 0.0
 50.0 0.0 0.0
 60.0 0.0 0.0
 70.0 0.0 0.0
 80.0 0.0 0.0
 90.0 0.0 0.0
```

---

*Appendix B: File Format*

```
100.0 0.0 0.0
```

DRD\_FILE path geometry sample:

```
[TRACK_PATH]
TYPE = 'DRD_FILE'
REFERENCE = 'ABSOLUTE'
FILE_NAME = 'mesh_centerline.drd'
```

## ROAD\_DATA block

The ROAD\_DATA block is used only by [composite](#) roads.

Attribute for ROAD\_DATA block:

### FILE\_NAME

It defines the road data file that will be referenced by the composite road.

```
[ROAD_DATA]
FILE_NAME = './straight_mesh_psd_irreg.rbf'
```

## OPEN\_CRG block

The OPEN\_CRG data block is used only by [OPEN\\_CRG](#) roads.

Attribute for OPEN\_CRG block:

### FILE

It defines the actual OpenCRG road data file that will be referenced by the RDF wrapper.

### MPC\_ACTIVE

activates or deactivates the multiple-point-contact (MPC) mode for OpenCRG roads. The MPC mode computes three points on the OpenCRG road in order to extract information about the surface normal vector (not provided by the OpenCRG API). The MPC is activated by default and may be de-activated (to improve computational efficiency) setting MPC\_ACTIVE= 'FALSE'. Note that the normal vector computed without MPC is always directed along the Z axis of the road reference frame: {0, 0, 1}.

### APPLY\_MODIFIERS

applies or not the CRG modifiers, if available. The default is TRUE, i.e. the CRG modifiers are applied when the road is loaded.

```
[OPEN_CRG]
FILE = './handmade_straight_extended.crg'
MPC_ACTIVE = 'TRUE'
APPLY_MODIFIERS = 'TRUE'
```

## 5.9.3 RDF Examples

The following road data file formats are available:

- [Analytic Sample](#)
- [Measured Sample](#)

- [Extrusion Sample](#)
- [Mesh Sample](#)
- [Flat Sample](#)
- [Composite Sample](#)
- [External Sample](#)

## Analytic Sample

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'vitools::vi_road'
TYPE = 'ANALYTIC'
MODE = 'STANDARD'
FORMAT = 'STANDARD'
$-----GLOBAL
[GLOBAL]
OUTBOUND_SAFE = 'TRUE'
EVENT_CHECK = 'TRUE'
DISTANCE_CALC = 'STANDARD'
FRICTION = 'STANDARD'
IRREGULARITIES = 'FLAT'
LEFT_EDGE = 'FLAT'
RIGHT_EDGE = 'FLAT'
ORIGIN_X = 0
ORIGIN_Y = 0
ORIGIN_Z = 0
ORIGIN_A3 = 3.141592653589793
$-----WIDTH
[WIDTH]
LEFT_SIDE_WIDTH = 5
RIGHT_SIDE_WIDTH = 5
$-----WIDTH_TABLE
[WIDTH_TABLE]
{ track_s width }
0 10
800 10
$-----FRICTION
[FRICTION]
LEFT_SIDE_MU = 1
RIGHT_SIDE_MU = 1
$-----PLANT_PATH
[PLANT_PATH]
{ s curvature }
0 0
```

## Appendix B: File Format

```

160 0
180 0.01
240 0.01
260 0
420 0
440 -0.02
570 -0.02
590 0
650 0
660 0.033
730 0.033
800 0
$-----VERTICAL_PATH
[VERTICAL_PATH]
{ s height }
0 0
800 0
$-----BANK_ANGLE_PATH
[BANK_ANGLE_PATH]
{ s angle }
0 0
800 0

```

**Measured Sample**

```

$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE = 'rdf'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'meter'
ANGLE = 'radian'
FORCE = 'newton'
MASS = 'kilogram'
TIME = 'second'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'vitoools::vi_road'
TYPE = 'MEASURED'
MODE = 'STANDARD'
FORMAT = 'STANDARD'
$-----GLOBAL
[GLOBAL]
OUTBOUND_SAFE = 'TRUE'
EVENT_CHECK = 'TRUE'
DISTANCE_CALC = 'STANDARD'
FRICTION = 'STANDARD'
IRREGULARITIES = 'FLAT'
LEFT_EDGE = 'KERB'
RIGHT_EDGE = 'KERB'
ORIGIN_X = 0
ORIGIN_Y = 0
ORIGIN_Z = 0
ORIGIN_A3 = 0
$-----WIDTH
[WIDTH]
LEFT_SIDE_WIDTH = 5
RIGHT_SIDE_WIDTH = 5

```

```

$-----FRICTION
[FRICTION]
LEFT_SIDE_MU = 1
RIGHT_SIDE_MU = 1
$-----MEASURED_CENTERLINE
[MEASURED_CENTERLINE]
{ x y z bank_angle }
0.000000000E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
-1.000052388E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
-2.000104776E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
-3.000157164E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
-4.000209552E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
-5.000261940E+00 0.000000000E+00 0.000000000E+00 0.000000E+00...
7.000366716E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
6.000314328E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
5.000261940E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
4.000209552E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
3.000157164E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
2.000104776E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
1.000052388E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
0.000000000E+00 0.000000000E+00 0.000000000E+00 0.000000E+00
$-----LEFT_KERB_GEOMETRY
[LEFT_KERB_GEOMETRY]
{ track_s slope_1[%] width_1 mu_1 slope_2[%] width_2 mu_2 }
0 0 0 1 0 0 1
285 0 0 1 0 0 1
295 0 1.5 1 0 0 1
335 0 1.5 1 0 0 1
345 0 0 1 0 0 1
1700 0 0 1 0 0 1
$-----RIGHT_KERB_GEOMETRY
[RIGHT_KERB_GEOMETRY]
{ track_s slope_1[%] width_1 mu_1 slope_2[%] width_2 mu_2 }
0 0 0 1 0 0 1
350 0 0 1 0 0 1
360 0 1.5 1 0 0 1
395 0 1.5 1 0 0 1
405 0 0 1 0 0 1
1700 0 0 1 0 0 1

```

## Extrusion Sample

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'vitoools::vi_road'
TYPE = 'EXTRUSION'
$-----GLOBAL
[GLOBAL]
OUTBOUND_SAFE = 'TRUE'

```

## Appendix B: File Format

```
EVENT_CHECK = 'TRUE'
DISTANCE_CALC = 'STANDARD'
FRICTION = 'STANDARD'
ORIGIN_X = 2
ORIGIN_Y = 0
ORIGIN_Z = 0
ORIGIN_A3 = 3.141592653589793
$-----MEASURED_CENTERLINE
[MEASURED_CENTERLINE]
{ x_coord y_coord z_coord bank_angle }
0 0 0 0
5 0 0 0
10 0 0 0
15 0 0 0
20 0 0 0
25 0 0 0
30 0 0 0
35 0 0 0
40 0 0 0
45 0 0 0
50 0 0 0
55 0 0 0
60 0 0 0
65 0 0 0
70 0 0 0
75 0 0 0
80 0 0 0
85 0 0 0
90 0 0 0
95 0 0 0
100 0 0 0
105 0 0 0
110 0 0 0
115 0 0 0
120 0 0 0
125 0 0 0
130 0 0 0
135 0 0 0
140 0 0 0
145 0 0 0
150 0 0 0
155 0 0 0
160 0 0 0
165 0 0 0
170 0 0 0
175 0 0 0
180 0 0 0
185 0 0 0
190 0 0 0
195 0 0 0
200 0 0 0
205 0 0 0
210 0 0 0
215 0 0 0
220 0 0 0
225 0 0 0
230 0 0 0
235 0 0 0
240 0 0 0
245 0 0 0
250 0 0 0
```

```
255 0 0 0
260 0 0 0
265 0 0 0
270 0 0 0
275 0 0 0
280 0 0 0
285 0 0 0
290 0 0 0
295 0 0 0
300 0 0 0
305 0 0 0
310 0 0 0
315 0 0 0
320 0 0 0
325 0 0 0
330 0 0 0
335 0 0 0
340 0 0 0
345 0 0 0
350 0 0 0
355 0 0 0
360 0 0 0
365 0 0 0
370 0 0 0
375 0 0 0
380 0 0 0
385 0 0 0
390 0 0 0
395 0 0 0
400 0 0 0
$-----CENTERLINE_PROPERTIES
[centerline_properties]
SEGMENTATION = 'NONE'
INTERPOLATION = 'BASIC'
SEGMENT = 0
(Properties)
{ s ori a1 a2 a3 scale1 scale2 friction section }
0 0 0 0 1 1 1 1
50 0 0 0 1 1 1 2
$-----SECTION_PROPERTIES
[section_properties]
SEGMENTATION = 'NONE'
INTERPOLATION = 'BASIC'
SEGMENT = 0
ABSOLUTE_ORIENTATION = 'FALSE'
POINTS = 6
VARIABLE_SECTION = 'TRUE'
(List)
{ name }
'SECTION_1'
'SECTION_2'
$-----SECTION_1
[section_1]
{ u v friction }
-3 0.2 1
-2.4 0.2 1
-2 0 1
2 0 1
2.4 0.2 1
3 0.2 1
$-----SECTION_2
```

## Appendix B: File Format

```
[SECTION_2]
{ u v friction }
-3 0.2 1
-2.4 0.1 1
-2 0 1
2 0 1
2.4 0.1 1
3 0.2 1
```

**Mesh Sample**

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'MILLIMETER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'VITOOLS::VI_ROADSINGLE'
TYPE = 'MESH'
$-----GLOBAL
[GLOBAL]
OUTBOUND_SAFE = 'TRUE'
EVENT_CHECK = 'TRUE'
DISTANCE_CALC = 'STANDARD'
FRICTION = 'STANDARD'
MESH_CHECK = 'FALSE'
MESH_LOOKUP_CELL_SIZE= 1000
MESH_LOOKUP_ZTOL = 100
ORIGIN_X = 0
ORIGIN_Y = 0
ORIGIN_Z = 0
ORIGIN_A3 = 3.141592653589793
$-----NODES
[NODES]
{ id x_coord y_coord z_coord }
1 -10000 -10000 -1000
2 -10000 10000 -1000
3 100000 -10000 5000
4 100000 10000 5000
$-----ELEMENTS
[ELEMENTS]
{ n1 n2 n3 mu }
1 3 2 1
3 4 2 0.5
```

**Flat Sample**

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
```

```

FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'vitools::vi_road'
TYPE = 'FLAT'
MODE = 'STANDARD'
FORMAT = 'STANDARD'
$-----GLOBAL
[GLOBAL]
ORIGIN_X = 0
ORIGIN_Y = 0
ORIGIN_Z = 0
$-----FLAT_GEOMETRY
[FLAT_GEOMETRY]
X_DIMENSION = 160.0
Y_DIMENSION = 80.0
$-----FRICTION
[FRICTION]
GLOBAL = 1.0

```

## External Sample

The external road is used to interface VI-CarRealTime solver with another package which defines its own road. In order to run a simulation with an external road, an rdf as the one defined below must be passed in input to the event to be submitted.

An external road must provide the following inputs (in VI-CarRealTime global reference system):

- Contact Patch X location
- Contact Patch Y location
- Contact Patch Z location
- Road Normal X direction cosine
- Road Normal Y direction cosine
- Road Normal Z direction cosine
- Road Friction

Here an example of an external road data file:

```

$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'VITOOLS::VI_ROADSINGLE'

```

## Appendix B: File Format

```
TYPE = 'EXTERNAL'
```

## Composite Sample

The composite road is a new road format. It is used to overwrite a predefined set of parameters of a referenced road.

Currently only [MESH](#) roads are supported as referenced roads.

By using the composite road, the user can interact with the referenced meshed road (in general a binary road data file whose data can't be manually edited) in order to overwrite certain parameters and certain data blocks.

If the same parameter is defined both in the referenced road and in the composite road, the solver will use the composite road value.

Here an example of a composite road data file:

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'VITOOLS::VI_ROADSINGLE'
TYPE = 'COMPOSITE'
$-----GLOBAL
[GLOBAL]
MESH_LOOKUP_CELL_SIZE = 2
MATERIALS_MODE = 1
$-----IRREGULARITIES
[IRREGULARITIES]
MODEL = 'ANALYTIC_PSD'
CALC_MODE = 0
STARTING_S = 10
ENDING_S = 90
SCALE = 0.5
FFT_POINTS = 256
PHASE = 0.003
COHERENCY = 0.75
OMEGA_1 = 0.01
OMEGA_2 = 15.2
LEFT_PSD_O1 = 0.0008
LEFT_PSD_O2 = 1.182e-006
RIGHT_PSD_O1 = 0.0008
RIGHT_PSD_O2 = 1.182e-006
$-----MATERIALS
[MATERIALS]
{ id mu }
 1 0.8
 2 0.5
$-----ROAD_DATA
[ROAD_DATA]
FILE_NAME = './straight_mesh_psd_irreg.rbf'
```

With version 17.1, the composite road supports the replacement of the following blocks:

- **[GLOBAL]**

Global block can be defined in the composite road but only the following parameters are allowed:

- [MESH\\_LOOKUP\\_CELL\\_SIZE](#)
- [MATERIALS\\_MODE](#)
- [IRREGULARITIES](#)

**[MATERIALS]**

- **[IRREGULARITIES]**

If the IRREGULARITIES block is defined in the composite road, the whole set of parameters children of such block will be used by the solver for the analysis. It is not possible to overwrite only certain parameters of the IRREGULARITIES block, the block will be entirely replaced.

Such block is effectively used only if the referenced meshed road data file has the [TRACK\\_PATH](#) defined.

## OpenCRG Sample

The OPEN\_CRG road is simply a wrapper to the OpenCRG road model, used to provide some additional features to the basic model.

Here an example of a OPEN\_CRG road data file:

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'RDF'
FILE_VERSION = 12
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----MODEL
[MODEL]
METHOD = 'USER'
FUNCTION_NAME = 'VITOOLS::VI_ROADSINGLE'
TYPE = 'OPEN_CRG'
$-----GLOBAL
[GLOBAL]
SMOOTHING_TIME = 1.0
$-----OPEN_CRG
[OPEN_CRG]
FILE = './handmade_straight_extended.crg'
MPC_ACTIVE = 'TRUE'
```

## 5.9.4 Discontinued ARM Road Model

The older ARM (Advanced Road Model) is still supported inside the VI-Road module, but its usage is strongly discouraged (deprecated).

An ARM road data file can be identified by the presence of the **FUNCTION\_NAME = 'vitoools::viarm2adm'** under the **[MODEL]** data block.

If you absolutely need to get back to work an ARM road model data file, the following changes has to be done to the file itself:

## Appendix B: File Format

1. change the **FUNCTION\_NAME** from 'FUNCTION\_NAME = 'vitoools::viarm2adm' to **FUNCTION\_NAME = 'vitoools::vi\_road'**
2. remove the **DIRECTION** attribute key from the [GLOBAL] data block
3. add the attribute **ORIGIN\_A3 = 180** (or 3.141589265.. according to the file angular units)

## 5.9.5 Deprecated SURFMESH Road Model

The SURFMESH road model is deprecated, even if still supported (for the time being) inside the VI-Road module. Its usage is strongly discouraged, since the most advanced [MESH](#) road model took its place.

To transform an existing SURFMESH road in a [MESH](#) road it is generally sufficient:

- within the **[MODEL]** data block, change the following key:  
**TYPE = 'SURFMESH'** to **TYPE = 'MESH'** .
- within the **[NODES]** and **[ELEMENTS]** data blocks, delete the following key (if present):  
**NUMBER\_OF\_NODES = ...**

To include the MESH road additional functionalities (e.g. [TRACK\\_PATH](#) and [IRREGULARITIES](#)) to the original SURFMESH data please look at the MESH documentation.

## 5.10 SDF

**Important Note:** SDF File Format is obsolete! It will be removed in next major release.

The SDF file is a fully comprehensive teim orbit file containing :

- the event description
- the controller options
- the vehicle parameters

The structure of each SDF file is based on the following needed blocks:

- HEADER
- UNITS
- MODEL
- CONTROLLER
- VEHICLE
- MANEUVER

A set of optional blocks may be needed according to the event definition. They are used to store target maps, paths, etc..

When no description is supplied, it means that currently only one option is supported for that feature (no alternatives available).

The values shown in the SDF fragments are the defaults values for each feature. Along the chapter the DRD and DCD file formats will be referenced. For a description of the DRD/DRD2 formats please refer to the VI-Road Documentation. For the DCD format please refer to standard Adams Car documentation.

### 5.10.1 MODEL Block

The model block defines the driver model to be used and its most general settings. The following keys should be specified:

**TYPE**           = 'VEHICLE\_STANDARD'

For Adams Car environment only **VEHICLE\_STANDARD** is supported while for the VI-MotorCycle environment it should be set to **MOTORCYCLE\_STANDARD**

```
MODE = 'ADAMS'
FORMAT = 'STANDARD'
REF_SYSTEM = 'RELATIVE'
INITIAL_PATH = 'AUTO'
```

The **REF\_SYSTEM** key define if the path merging option is active or not. When a multi maneuver event is requested and the ref system is set to **RELATIVE** the path associated at each maneuver is roto-translated in order to match the current path. Also the S-coordinate specified in each maneuver table is meant to be relative to the maneuver beginning (on each maneuver switch the counter of s-coordinate is set to 0). The **ABSOLUTE** setting instead instruct the controller to use each path/map with no preliminary operation. The operations performed for the first maneuver depend on the key **INITIAL\_PATH**: using **AUTO**, the first path is aligned to the gyro location/orientation while using **ABSOLUTE** no path alignment occurs (this should be used in conjunction with **REF\_SYSTEM='ABSOLUTE'** ).

## 5.10.2 CONTROLLER Block

The controller block includes the controller general settings. The following features are supported:

```
MIN_PREVIEW_DISTANCE = 5.000000
```

is the bottom saturation value for preview distance computation. It should be grater than the vehicle wheel base. It should be used to prevent the preview distance from becoming too small during low speed maneuvers

```
STEER_SIGMA_RESMAX = 5.000000
STEER_SIGMA_RESMIN = -5.000000
STEER_SIGMA_SEARCHIND0 = 15
```

This keywords are ignored in MODE='VEHICLE\_STANDARD'. They are used to define trajectory planning phase in the MOTORCYCLE\_STANDARD mode

```
TIRE_FZ_CONTROL = 'YES'
```

Enable/disable the check on tire vertical forces. When the option is active and the rear tires normal force is 0, the controller reduces the control actions on throttle/brake

```
RPM_CONTROL = 'NO'
```

RPM controller is an advanced feature that affects the gear shifting process. When the option is YES, the controller will acts on throttle during the gear shifting trying to match the correct RPM. These features help reducing the longitudinal acceleration peaks due to gear shifting

```
RAX_SATURATION = 'YES'
```

Enable the saturation of the required AX when the maximum engine torque is already applied and more acceleration is required. This feature is designed to reduce the throttle overshoot.

```
FEED_FORWARD = 'YES'
```

enable/disable the feed forward part of the longitudinal controller. Turning off the feed forward is typically useful for debugging porpoises or for making the controller response really slow.

```
KICKDOWN_CONTROL = 'NO'
```

Instruct the controller to use the gear that makes the engine produce the maximum torque when a full throttle acceleration is required. This option requires that the vehicle block includes the gear ratio keys.

## Appendix B: File Format

**GEAR\_OPENTHROTTLE\_CONTROL = 'NO'**

When this option is active and the controller is configured for open loop throttle mode, if a gearshift process starts, the controller will step the throttle demand to 0 until the gear shifting is completed (open loop signal is overridden).

### 5.10.3 VEHICLE Block

The vehicle block contains the controller internal model parameters:

- General parameters
- Vehicle mode parameters
- Motorcycle mode parameters

#### General parameters:

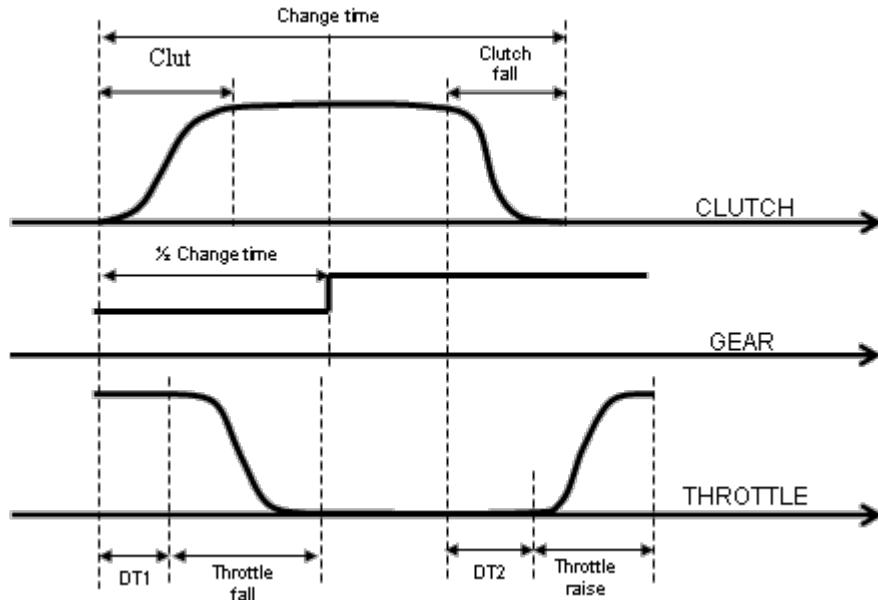
**STEERING\_MODE = 1**  
**WHEELBASE = 2.997200**

Set the wheel base of the one track model.

**GEAR\_SHIFT\_TIME = 0.200000**  
**MIN\_GEAR = 1**  
**GEAR\_SHIFT\_DELAY = 0.020000**  
**GEAR\_SHIFT\_DT1 = 0.000000**  
**GEAR\_SHIFT\_DT2 = 0.000000**  
**THROTTLE\_FALL\_TIME = 0.030000**  
**THROTTLE\_RAISE\_TIME = 0.030000**  
**CLUTCH\_FALL\_TIME = 0.030000**  
**CLUTCH\_RAISE\_TIME = 0.100000**

The gear shifting process is defined using the above keys.

A plot preview of clutch and throttle signals is also available:



**MIN\_GEAR = 1**  
**MAX\_GEAR = 7**

Define the upper and lower bound of the available gears

```
GEAR_1_UPSHIFT_RPM = 8000.000000
GEAR_1_DOWNSHIFT_RPM = 3000.000000
```

For each gear it's possible to specify valid rpm range. The gearshift process occurs when the actual engine rpm exceeds the defined range. When the **KICKDOWN** option is enabled, the gearshift may occur before reaching the range limits. It is possible to specify a unique rpm upshift/downshift value for all gears using the key **GEAR\_UPSHIFT\_RPM** and **GEAR\_DOWNSHIFT\_RPM**.

```
GEAR_1_RATIO = 1.000000
```

The gearshift ratio defines the reduction ratio of each gear defined as at the gearbox level. These keys are required only when the kickdown option is active.

```
F_MAX_TORQUE = 1740.000000
R_MAX_TORQUE = 1040.000000
```

The max torque parameters define the torque delivered at the wheel level when a full brake demand is applied. They affects only the braking demand calculation

```
F_BIAS = 0.600000
```

The brake bias parameter specify how the brake demand is split between the front and rear axle. In general it is equal to .

```
FINAL_DRIVE = 1.000000
```

Specify the final drive ratio. It should include all the available reduction ratios between the gearbox output and the driving wheels. This keyword is required only when the KICKDOWN feature is active.

```
MIN_RPM_LIMIT = 0.000000
MAX_RPM_LIMIT = 50000.000000
```

Define the rpm range valid for the engine model. When the rpm fall below the **MIN\_RPM\_LIMIT** and the lowest gear is used, the clutch is disengaged. The upshift/downshift parameters should be inside the range defined by the **MIN/MAX\_RPM\_LIMIT**.

```
F_TIRE_RADIUS = 0.320000
R_TIRE_RADIUS = 0.320000
```

Average radius of front and rear tires. This information affects the calculation of the brake demand. In general the unloaded tire radius should be supplied.

```
STEERING_ANGLE_MAX = 12.566371
STEERING_ANGLE_MIN = -12.566371
```

The maximum steering angle produced by both the control algorithms is saturated using the limits above (in vehicle mode the saturation is applied only when the actuation mode is rotation).

### Vehicle mode parameters:

```
STEERING_RATIO = 17.142900
```

Define the Steering wheel vs average wheel toe ratio. It can be easily calculated from a steering analysis on the front suspension assembly.

```
STEERING_RACK_RATIO = 0.174500
```

It is the reduction ratio of the pinion/rack coupler (or equivalent for non standard steering models). This parameter is used only when driving by force/displacement

## Appendix B: File Format

```
STEERING_DISP_MAX = 1.000000
STEERING_DISP_MIN = -1.000000
STEERING_FORCE_MAX = 10000.000000
STEERING_FORCE_MIN = -10000.000000
STEERING_TORQUE_MAX = 100.000000
STEERING_TORQUE_MIN = -100.000000
```

The steering demand produced by the controller is always saturated using the limits specified by the saturation parameters above.

**Motorcycle mode parameters:**

```
TIRE_SECTION_RADIUS = 0.000000
```

Define the average tire section radius of the motorcycle rear tire. This parameter affects the controller target roll calculation.

**5.10.4 MANEUVER Block**

The aim of the maneuver block is to collect the set of maneuvers to be performed. It also specify the simulation initial conditions.

```
[MANEUVER]
INITIAL_SPEED = 47.777778

INITIAL_STEER = 0.000000
INITIAL_THROTTLE = 0.000000
INITIAL_BRAKING = 0.000000
INITIAL_GEAR = 5
INITIAL_CLUTCH = 0.000000
```

The initial demand parameters may be omitted. In this case they defaults to 0

```
(MANEUVERS)
{ name abort_time step_size }
'MAN1' 2.0 1.0E-2
'MAN2' 2.0 1.0E-2
'MAN3' 2.0 1.0E-2
```

the abort time expresses the duration of each maneuver. It is suggested to maintain the step size parameter equal to 1E-02 seconds.

For each maneuver listed in the **MANEUVERS** sub block, an additional maneuver block should be supplied. In the case shown above three data block called MAN1, MAN2 and MAN3 should exists inside the SDF file.

**5.10.5 Maneuver sub block**

```
[MAN1]
TASK = 'NORMAL'
SMOOTHING_TIME = 0.100000
```

The smoothing time paramter specify a transition time between the previous states of the controller and the ones defined in the curent manoeuvre. A step function is used to make the transition smooth.

```
TIME_REF_MODE = 'RELATIVE'
```

This keyword works together with the **REF\_SYSTEM** keyword of the **MODEL** block. Setting the **TIME\_REF\_MODE** to **RELATIVE** instruct the controller to consider the time maps of the current

manoeuvres as relative to the beginning of the manoeuvre even if the `REF_SYSTEM_MODE` is set to `ABSOLUTE`.

```
(STEERING)
METHOD = 'MACHINE'
ACTUATOR_TYPE = 'ROTATION'
(THROTTLE)
METHOD = 'MACHINE'
(BRAKING)
METHOD = 'MACHINE'
(GEAR)
METHOD = 'MACHINE'
(CLUTCH)
METHOD = 'MACHINE'
```

For each control channel, a control method should be supplied. The supported method are the same used for in the dcf definition apart from the `CONTROL_FUNCTION` mode that is not directly supported by the controller. Refer to the example SDF provided with the plug in for an overview of the most common configurations. Note that the same method should be specified for Throttle/Brake and Gear/Clutch.

## MACHINE sub block

```
(MACHINE_CONTROL)
SPEED_CONTROL = 'MAINTAIN'
PATH_CONTROL = 'PATH'
```

When some channels are set to machine, the machine control block should be defined. It instruct the controller for the task to fulfill.

## PATH\_CONTROL

The path control keyword supports the following options:

- [STRAIGHT](#)
- [SKIDPAD](#)
- [PATH](#)
- [LATERAL\\_ACC](#)

## STRAIGHT

A straight line path is automatically created and aligned to the forward vehicle direction. No additional input are needed for completing the path specification

## SKIDPAD

The skidpad path is built merging a straight line with a constant radius turn. The following parameters should be supplied inside the `machine_control` block to complete the skidpad definition:

- `RADIUS = 50000.00`
- `TURN_ENTRY_DISTANCE = 11111.11`
- `TURN_DIRECTION = 'LEFT'`
- `AMPLITUDE = 10`
- `STEP_SIZE = 0.1`

## Appendix B: File Format

The **amplitude** parameter define the total yaw angle of the path. The default value is 2p. Using greater values are needed to produce multiple laps. The **step size** key define the resolution of the skidpad path. Reducing the step size helps getting smoother signals from the steering.

### PATH

The path setting is the most general one. This option should be supplied with the **DRIVER\_PATH** specification which define the name of the additional data block that describes the path.

```
STEERING_CONTROL = 'PATH'
DRIVER_PATH = 'PATH_1'
```

Refer to the [PATH block](#) chapter for the description of the supported features.

### LATERAL\_ACC

Instruct the controller for actuating the steering in order to follow the lateral acceleration map defined in the map specified by the keyword:

```
LATERAL_ACC_MAP = 'AY_MAP_1'
```

Refer to the [MAP block](#) for additional details on the maps definition. This option is not supported in MotorCycle mode.

### SPEED\_CONTROL

The speed control keyword defines the task for the longitudinal controller. The supported options are:

- [MAINTAIN](#)
- [MAP](#)
- [ACCMAP](#)
- [ACCMAP2](#)
- [LATACCMAP](#)

### MAINTAIN

The longitudinal controller will try to hold the vehicle speed to the current value (due to initialization or inherited from previous maneuvre)

### MAP

Reference an additional block containing a speed map. Refer to the MAP Block paragraph for information about the maps format. The name of the map containing the longitudinal speed profile is defined with the auxiliary keyword:

```
SPEED_MAP = 'MAP_1'
```

### ACCMAP, ACCMAP2

Reference an additional block containing an acceleration map. Refer to the MAP Block paragraph for information about the maps format. The name of the map containing the longitudinal acceleration profile is defined with the auxiliary keyword:

```
SPEED_MAP = 'MAP_1'
```

Using **ACCMAP2** option the controller will also control the error on the computed target speed (integral of target AX). This means that if the task is a constant longitudinal acceleration and a gearshift occurs, it will happen that desired acceleration will not be delivered for the gearshift duration. The controller will try to compensate the error on target speed, requiring more acceleration for some time after the gearshift is completed. Using the **ACCMAP** option instead the controller will disregard completely the reference speed and will only try to match the target acceleration except during each gearshifting.

### LATACCMAP

Reference an additional block containing an acceleration map. Refer to the MAP Block paragraph for information about the maps format. The name of the map containing the lateral acceleration map is defined with the auxiliary keyword:

```
SPEED_MAP = 'MAP_1'
```

The controller will actuate throttle and brake in order to achieve the specified lateral acceleration map when driving on the assigned path. This option requires that the steering method is set to MACHINE and is not supported in MotorCycle mode.

### END\_CONDITION Sub Block

The end condition sub block contains the list of condition which can cause the manoeuvre termination before reaching the manoeuvre abort time. For the full set of supported keyword please refer to the Adams Car driving machine documentation.

```
(END_CONDITIONS)
{measure test value allowed_error filter_time delay_time group }
'VELOCITY' '<<' 1.38889 0 0 0 '----'
```

End condition grouping is supported.

## 5.10.6 PATH Block

The path block defines the target for the steering controller when a non predefined path is assigned (**STRAIGHT** or **SKIDPAD**)

The keys to provide depends on the TYPE defined:

```
TYPE = 'DCD_FILE'
```

The supported path TYPE are:

- [DCD\\_FILE](#)
- [DRD\\_FILE](#)
- [MEASURED](#)

Some optional keyword may be added to pre-process the path

```
INTERPOLATION = 'CUBIC'
```

Define the interpolation method for the supplied points. When the points resolution is smaller than 0.5 meter, no interpolation is performed, otherwise the cubic interpolation method is applied. The other supported option is NONE.

```
INTERPOLATION_STEP = 0.5
```

Specify the resolution to be used during the interpolation process (no effect when interpolation is NONE)

**Appendix B: File Format****REF\_X, REF\_Y, REF\_Z**

Specify a rigid translation of the path along the global reference coordinate (ground)

**REF\_AZ**

Define a rigid path rotation along the global Z axis.

**DCD\_FILE**

The usage of the **DCD\_FILE** option requires to define the full path to access the desired dcd file using the following keyword:

```
FILE = '<database_name>/driver_roads.tbl/path_1.dcd'
```

The usage of database aliases is supported.

**DRD\_FILE**

The usage of the **DRD\_FILE** option requires to define the full path to access the desired dcd file using the following keyword:

```
FILE = '<database_name>/driver_roads.tbl/path_1.dcd'
```

The usage of database aliases is supported. The units of the drd file should be MKS.

**MEASURED**

The set of desired points may be directly added to the SDF file into the **MEASURED** sub block. In order to make the SDF file more readable, it's suggested to use the **DRD\_FILE** or **DCD\_FILE** option instead of the measure one.

An example of measure path definition is the following:

```
[PATH_1]
TYPE = 'MEASURED'
INTERPOLATION = 'CUBIC'
(MEASURED)
{x y z}
0.0 0.0 0.0
-1000.0 0.0 0.0
```

**5.10.7 MAP Block**

The map block is a data structure designed to hold information of several types. The most common of data that can be stored inside a map are:

- Speed vs time
- Speed vs path\_s
- Acceleration vs time
- Acceleration vs path\_s
- File

Where acceleration may define both a longitudinal than a lateral target.

```
[MAP_1]
MAP_TYPE = 'SPEED_S'
(MAP)
{track_s speed}
```

```

0.0 20.0
100.0 20.0
200.0 30.0
300.0 20.0

```

the meaning of the data stored in the table depends on the option chosen for the **MAP\_TYPE** keyword. The available options are:

- SPEED\_S
- SPEED\_T
- ACCELERATION\_S
- ACCELERATION\_T
- DRD\_FILE
- DCD\_FILE

Where S means that the first column of the table is path s coordinate while T means time (absolute or relative according to the **TIME\_REF\_MODE** keyword).

The **DRD\_FILE** option should be completed by the specification of the file name containing the speed profile:

```
FILE = '<database_name>/driver_roads.tbl/sample_drd2.drd'
```

Note that a drd2 file format is required.

Also the **DCD\_FILE** option needs to define a file name using the following keyword:

```
FILE = '<database_name>/driver_roads.tbl/vx_sample.dcd'
```

## 5.11 VDF

The VI-Driver control algorithm should be instructed using a specific ASCII event file in teim orbit format. The following pages describe the required and optional blocks defining the tasks the controller should fulfill.

### 5.11.1 VIGRADE\_HEADER Block

The header block contains the general information about the tiem orbit file like the version and the format. No user settings are currently supported in this block

```
[VIGRADE_HEADER]
FILE_TYPE = 'vdf'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'
```

### 5.11.2 UNITS Block

The information included in each VDF file are expressed in the units system declared in the units block. The VDF files generated from VI-Driver are in MKS units system.

```
[UNITS]
LENGTH = 'meter'
ANGLE = 'degree'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
```

### 5.11.3 DEBUG\_PARAMETERS Block

The debug block controls the driver core debugging level and the output format for the debugging information.

```
[DEBUG_PARAMETERS]
ACTIVATION = 'FALSE'
INITIALIZATION_DUMP='TRUE'
```

setting the ACTIVATION key= 'TRUE' and INITIALIZATION\_DUMP='TRUE' VI-Driver will generate a pair of VDF files in the working directory called:

- initialization\_0\_T0.vdf
- computation\_1\_T0.vdf

These files will contain all the non defaults settings currently used by the driver calculation.

### 5.11.4 GLOBAL\_SETTINGS Block

The global block specify the controller to use for the current simulation.

```
[GLOBAL_SETTINGS]
MODEL_TYPE = 'CAR_STANDARD'
```

According to the package being used one of the following 2 controllers could be enabled:

- CAR\_STANDARD
- MOTORCYCLE\_STANDARD

```
PATH_POSITIONING = 'ABSOLUTE'
```

Available options are:

- ABSOLUTE  
the path coordinates are used as they are, but it is possible to set a location and an orientation (around global Z axis) to the path by using the parameters in [PATH](#) block.
- AUTOMATIC  
Puts the first point of the path in the driver origin reference system and orients the path so that it is aligned with the vehicle starting yaw angle.
- RAWDATA  
The path is positioned without any correction with respect to the coordinates written in the drd file.

```
MANEUVER_STOP_MANAGEMENT_MODE = 0
```

This numerical mode sets the global policy for the maneuver stop and switch actions. The default value is zero (0).

Available options are:

- 0
- 1

The zero (0) value causes a switch to the available next maneuver, while a value of one (1) will cause the complete stop of the driver, as if there would no more maneuver's available.

## 5.11.5 CONTROLLER\_STANDARD Block

The controller block collects the controller sub modules to activate for a certain maneuver. Multiple CONTROLLER blocks could be included in each VDF file and each maneuver could point to a different one, so different setting can be enabled for each maneuver.

```
[CONTROLLER_STANDARD]
PREVIEW_TIME = 0.5
```

The preview time parameter define how "far" the controller looks ahead of the vehicle for planning the control action. The effect of increasing the preview time is that the controller will produce smoother control actions on the steering but on the other side the path tracking accuracy will be reduced.

```
MIN_PREVIEW_DISTANCE = 5
```

When the specified task includes very low speed driving conditions, it is possible to define a lower limit for the controller preview distance.

```
LAT_ANT_COMP_TIME = 0
```

In order to include the delay between the imposed control action and the vehicle response, a time constant could be specified. The controller will anticipate its own actions accordingly. Typical values range from 0 to 0.15 seconds. This parameter affects only the lateral behaviour of the controller.

```
LON_ANT_COMP_TIME = 0
```

Same as LAT\_ANT\_COMP\_TIME but for the longitudinal controller

```
STEERING_PDC_ACTIVE = 'TRUE'
STEERING_PDC_ACTDIST = 0.1
```

Activate the path distance compensation module of the controller. When this option is **TRUE** and the vehicle is within the **STEERING\_PDC\_ACTDIST** bandwidth respect to the assigned trajectory, the residual error will be handled by this module. When running path compensation event, it is recommended to switch off the path distance compensation module.

```
STEERING_YAW_PID_ACTIVE = 'TRUE'
```

Activate the yaw correction module. It is recommended to keep this module active especially for maneuver including fast lateral transients.

```
THROTTLE_FEED_FWD_TRQ_ACTIVE = 'TRUE'
```

Enable the longitudinal controller prediction model.

```
THROTTLE_SAT_TGTACCX_ACTIVE = 'TRUE'
```

```
THROTTLE_RATE_LIMITER_ACTIVE = 'TRUE'
```

Flag to activate the rate limiter defined by the parameter **THROTTLE\_MAX\_RATE**

```
THROTTLE_ROLLON_MAX_RATE = 0.4
```

maximum value allowed for the throttle derivative signal while pushing the pedal. Internally max pedal value is always 1.0 so setting **THROTTLE\_ROLLON\_MAX\_RATE** = 0.4 will require 2.5 seconds to fully ramp up the throttle.

**THROTTLE\_ROLLOFF\_MAX\_RATE = 0.4**

maximum value allowed for the throttle derivative signal while releasing the pedal. Internally max pedal value is always 1.0 so setting **THROTTLE\_ROLLOFF\_MAX\_RATE** = 0.4 will require 2.5 seconds to fully release the throttle.

**BRAKING\_RATE\_LIMITER\_ACTIVE = 'TRUE'**

flag to activate the rate limiter defined by the parameter **BRAKING\_MAX\_RATE**

**BRAKING\_ROLLON\_MAX\_RATE = 0.4**

maximum value allowed for the brake derivative signal while pushing the pedal. Internally max pedal value is always 1.0 so setting **BRAKING\_ROLLON\_MAX\_RATE** = 0.4 will require 2.5 seconds to fully ramp up the brake.

**BRAKING\_ROLLOFF\_MAX\_RATE = 0.4**

maximum value allowed for the brake derivative signal while releasing the pedal. Internally max pedal value is always 1.0 so setting **BRAKING\_ROLLOFF\_MAX\_RATE** = 0.4 will require 2.5 seconds to fully release the brake.

**STEERING\_RATE\_LIMITER\_ACTIVE = 'TRUE'**

flag to activate the rate limiter defined by the parameter **STEERING\_MAX\_RATE**

**STEERING\_MAX\_RATE = 0.4**

maximum value allowed for the steering derivative signal (angular velocity)

**STEERING\_STRAIGHT\_LINE\_SMOOTHING\_ACTIVE = 'FALSE'**

Enable optimized controller settings for driving straight line maneuvers on very uneven roads. This option is designed to reduce the steering oscillation produced by driver reaction to input variation.

**VARIABLE\_STEERING\_RATIO\_ACTIVE = 'FALSE'**

Enable the usage of a variable steering ratio coefficient provided as an input signal. The user is currently responsible for computing the runtime steering ratio.

**PREVIEW\_DISTANCE\_CALC = 'STANDARD'**

The standard methodology for computing the desired end point of the connecting contour (path closest point) does not deliver enough accuracy for geometry with high curvature ( radius smaller than about 5 meters ). For this reason an **ADVANCED** methodology has been implemented specifically for the low speed maneuver definition: when **PREVIEW\_DISTANCE\_CALC** is set to **ADVANCED** the path closest point is defined as the point of the path with an S coordinate equal to the S of the gyro closest point + the current preview distance. The **ADVANCED** methodology does not support closed path and is not suitable for maneuver in which the path distance may become large ( > 2 meter )

## 5.11.6 STEERING\_STANDARD Block

```
[STEERING_STANDARD]
MAX_VALUE = 720
MIN_VALUE = -720
```

Define the saturation value for the steering angle. This saturation affects both open loop and machine driven maneuvers.

## 5.11.7 THROTTLE\_STANDARD Block

[THROTTLE\_STANDARD]

MAX\_VALUE = 100

MIN\_VALUE = 0

Saturation limits for the throttle demand. Both open loop and machine driven throttle computation will be saturated in the specified range.

SCALING\_FACTOR = 100

Scaling factor for the throttle demand. The controller internal computation is always in the range 0-1 but the throttle demand could be scaled in order to fit the model being driven. Please make sure that the specified scaling match the saturation parameters above.

PID\_LONVEL\_PROP\_GAIN = 100

PID\_LONVEL\_INTG\_GAIN = 50

PID\_LONVEL\_DERIV\_GAIN = 0

Proportional/Derivative/Integral gains for the correction module of the longitudinal controller for longitudinal velocity mode.

PID\_LONACC\_PROP\_GAIN = 30

PID\_LONACC\_INTG\_GAIN = 200

PID\_LONACC\_DERIV\_GAIN = 0

Proportional/Derivative/Integral gains for the correction module of the longitudinal controller for longitudinal acceleration mode.

## 5.11.8 BRAKING\_STANDARD Block

[BRAKING\_STANDARD]

MAX\_VALUE = 100

MIN\_VALUE = 0

Saturation limits for the brake demand. Both open loop and machine driven brake computation will be saturated in the specified range.

SCALING\_FACTOR = 100

Scaling factor for the brake demand. The controller internal computation is always in the range 0-1 but the brake demand could be scaled in order to fit the model being driven. Please make sure that the specified scaling match the saturation parameters above.

## 5.11.9 GEAR\_STANDARD Block

[GEAR\_STANDARD]

MAX\_VALUE = 6

MIN\_VALUE = 0

Saturation limits for the gear demand. Both open loop and machine driven gear computation will be saturated in the specified range. The maximum and minimum gear should define a subset of the gear table defined in the PROPERTIES\_MAP sub-block.

**Note:** when running a simulation from Adams Car using a VDF Event, if the **Update VDF** is checked, the new VDF file will contain the standard parameters for the following table, and not the manually modified of the original VDF file submitted. This is because these parameters are not saved in the model, so they cannot be exported

## Appendix B: File Format

from Adams when updating the VDF file. So be sure to use directly the manually modified VDF file (without the Update VDF option checked) in order to see the effect of the parameter changes.

```
(PROPERTIES_MAP)
{ id opt upRPM dwRPM up_sftT up_tfallT up_criseT up_cfallT up_triseT up_dt1 up_dt2 dw_sftT dw_tfallT dw_criseT
dw_cfallT dw_triseT dw_dt1 dw_dt2 }
0 0 7000 10000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
1 0 7000 0 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
2 0 9000 4000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
3 0 10500 8000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
4 0 12000 9000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
5 0 12000 10000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
6 0 12000 10000 0.2 0.05 0.06 0.05 0.1 0 -0.04 0.25 0.06 0.07 0.05 0.1 0 -0.04
```

**id**

the gear identification number is the conventional name for a certain gear. The id could be also negative to indicate backward driving gears.

**opt**

Future development, currently this information is neglected.

The following parameters can be defined separately for up-shifting gear and down-shifting gear

**upRPM / dwRPM**

Triggering condition for the standard gearshifting process. When no advanced gearshifting controllers are enabled, the controller will switch to the next greater gear id as soon as the engine rpm will exceed the limit specified for the current gear id. The same will happen for the downshift operations.

**up\_sftT / dw\_sftT**

Define the duration of each gearshifting operation

**up\_tfallT / dw\_tfallT  
up\_criseT / dw\_criseT**

when a gearshifting operation begins, the throttle is closed and the clutch is opened. The duration of the throttle release is specified by the (up/\_dw\_)**tfallT** entry of the gear table. The (up/\_dw\_)**criseT** define the duration of the clutch disengaging process.

**up\_cfallT / dw\_cfallT  
up\_triseT / dw\_triseT**

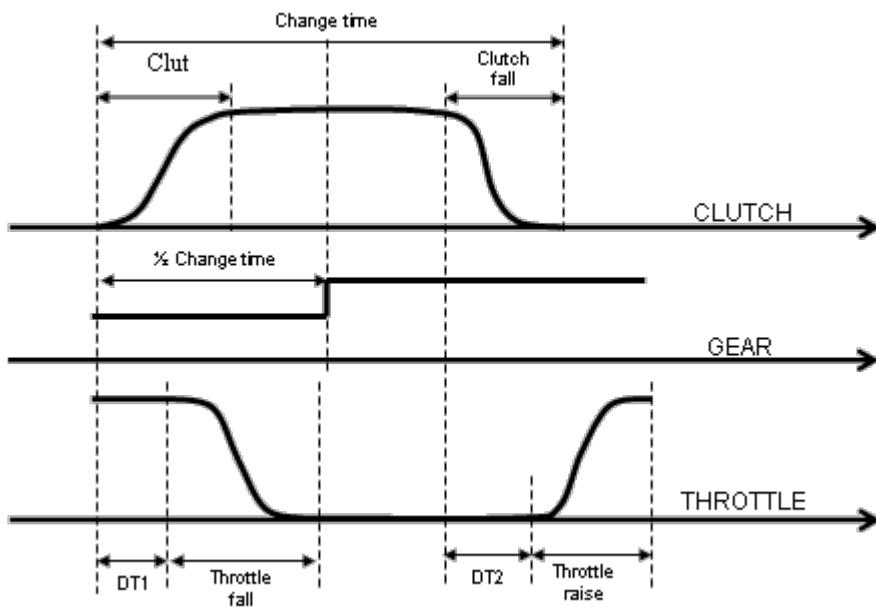
at the end of a gearshift operation, the clutch should be engaged and the duration of this process is controlled by the (up/\_dw\_)**cfallT** parameter while the (up/\_dw\_)**triseT** one will define the duration of the throttle opening

**up\_dt1 / dw\_dt1**

the throttle release process that occurs at the beginning of a gearshift operation could be delayed respect to the clutch disengaging one using the (up/\_dw\_)**dt1** parameter (time).

**up\_dt2 / dw\_dt2**

(up/\_dw\_)**dt2** is the delay of the throttle opening operation performed at the end of the gearshift with respect to the clutch engaging process.



#### UPSHIFTING\_AFTER\_UPSHIFTING\_DELAY = 0

Inhibits the up-shifting for a specified time delay after a previous up-shifting event. Default is zero, meaning that up-shifting time follows the properties map scheduling after a previous up-shifting event.

#### UPSHIFTING\_AFTER\_DOWNSHIFTING\_DELAY = 0

Inhibits the up-shifting for a specified time delay after a previous down-shifting event. Default is zero, meaning that up-shifting time follows the properties map scheduling after a previous down-shifting event.

#### DOWNSHIFTING\_AFTER\_UPSHIFTING\_DELAY = 0

Inhibits the down-shifting for a specified time delay after a previous up-shifting event. Default is zero, meaning that down-shifting time follows the properties map scheduling after a previous up-shifting event.

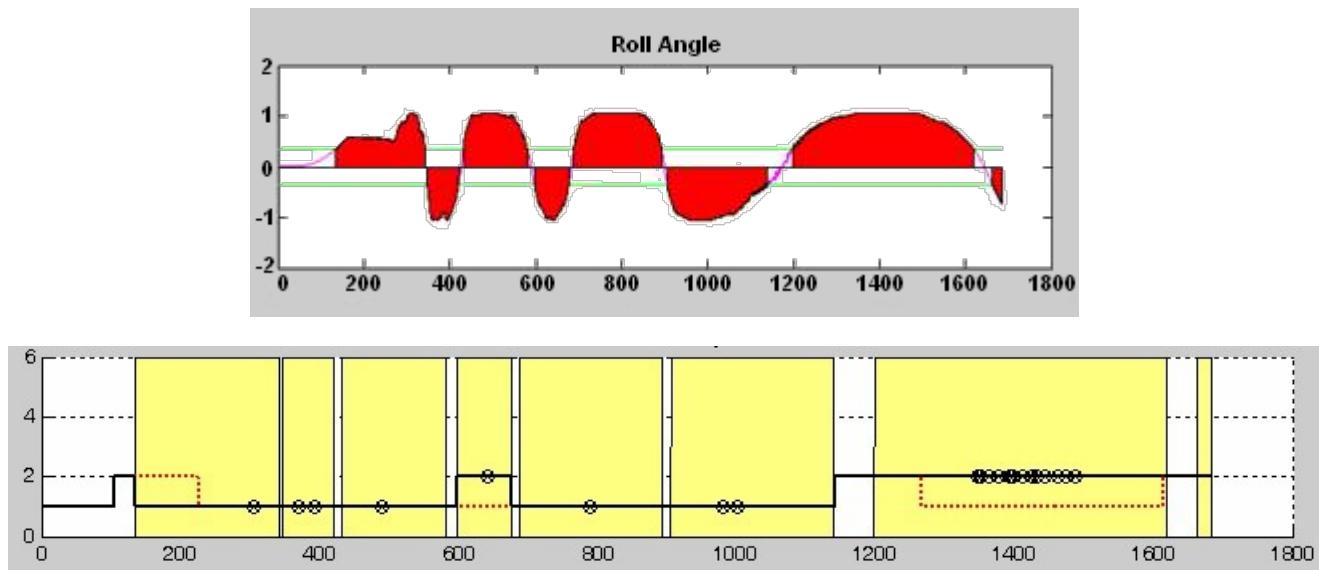
#### DOWNSHIFTING\_AFTER\_DOWNSHIFTING\_DELAY = 0

Inhibits the down-shifting for a specified time delay after a previous down-shifting event. Default is zero, meaning that down-shifting time follows the properties map scheduling after a previous down-shifting event.

## 5.11.10 GEAR\_EDS Block

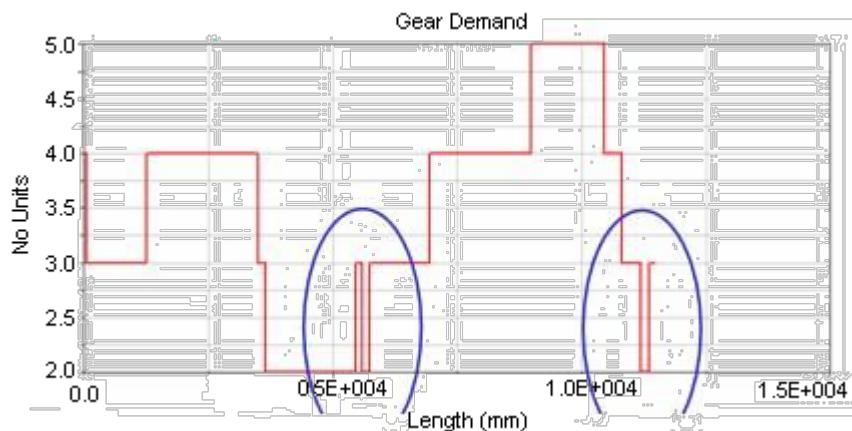
The EDS (early downshift controller) is an advanced gearshift module designed to pre-compute the gear history for the given track, speed map and vehicle. The aim of this adavnced controller is to optimize the gear usage in order to prevent gear shifting in "bad" conditions like high lean angle.

## Appendix B: File Format



The first step of the calculation estimates the motorcycle roll angle or the car yaw rate and the engine speed along the given track and identify the best gear for each turn apex, then several additional refining steps are performed over the obtained gear history map in order to:

- generate a continuous gear history (no gear jump allowed)
- anticipate all the downshift sections as much as possible taking care of the duration of each of them
- remove unuseful gear shifting (like too short ones)



The resulting gear history if finally used to drive the vehicle.

## [GEAR\_EDS\_TEST]

```
CONTROLLER_DATA = 'EDS_GEAR'
```

Identify the current controller block as an EDS (early down shift) one.

```
ACTIVATION = 'TRUE'
```

Activates the controller.

```
ROLLING_THRESHOLD = 20
```

Maximum roll angle for triggering gearshifting operation. When the estimated roll angle exceeds the specified boundary, the gear shifting operation will be suppressed.

```
MAX_RPM = 10500
MIN_RPM = 2000
```

Engine speed limits. The EDS logic will try to maintain the engine speed within these boundaries.

```
MIN_ACTIVE_TIME = 1
```

Defines the smaller engaging time for a gear. This parameter is meant to prevent multiple upshift/downshift sequence in a too short time.

```
SHIFTING_SAFETY_FACTOR = 2
```

Scaling factor for the shifting time to be applied during multiple downshift operations. Please note that this parameter should be always greater than 1 in order to preserve at least a full gearshift time for each downshift.

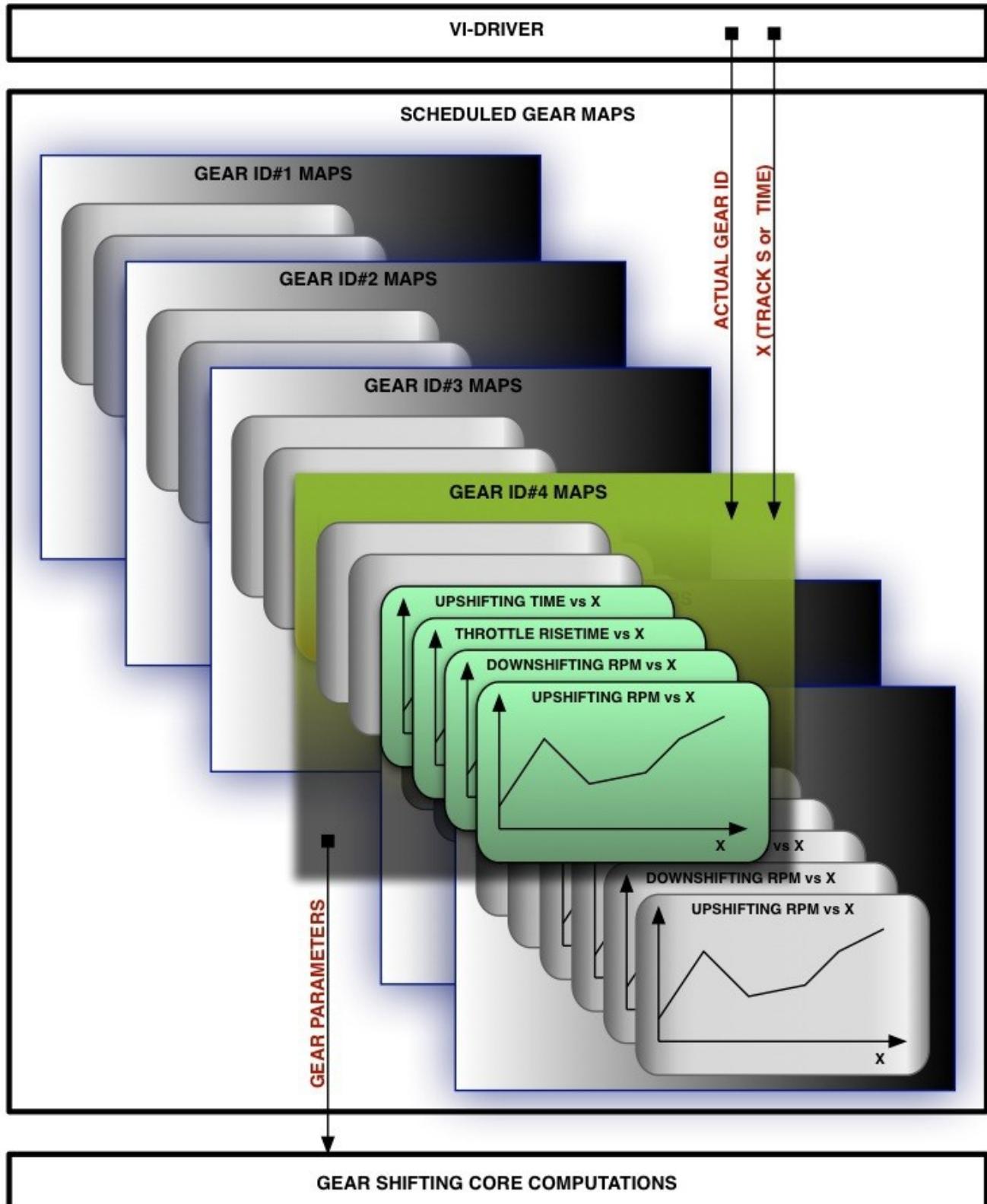
```
(OPTIMAL_GEAR_RPM)
{ gear_id rpm }
0 7000
1 5000
2 7000
3 10000
4 10000
5 11000
6 11000
```

For each gear it is possible to specify the recommended engine speed: the EDS controller will try to select at each turn apex the gear that will produce an engine speed closer to the optimal one.

## 5.11.11 GEAR\_SCHEDULED\_CONTROLLER Block

The SCHEDULED gear is an optional controller allowing the scheduling of the gear shifting according to specified maps. The gear shifting parameters are linearly interpolated from the given maps (function of track S or maneuver time), and used instead of "fixed" values to feed the gear shifting logic.

The synthetic working diagram is presented below.



The controller is activated if listed among the maneuver controllers. See [GEAR\\_SCHEDULED](#) in the MANEUVER data block.

The data block itself is rather complex, due to the fact that all the parameters for all the active gears have to be properly mapped.

**CONTROLLER\_DATA = 'GEAR\_MAPPED'**

The controller data type. Must be "GEAR\_MAPPED".

**ACTIVATION = 'TRUE'**

Activates the controller.

**MODEL = 'STANDARD'**

The model version. Only 'STANDARD' is supported for the time being.

**MAX\_GEAR\_ID = 5**

The maximum gear ID.

**MIN\_GEAR\_ID = 0**

The minimum gear ID.

**X\_DATA = 'PATH\_S'**

The independent variable value type, can be defined as 'PATH\_S' (the track S) or 'TIME' (the maneuver time).

**(GEAR\_I\_PROPERTIES\_MAP)**

| path_s | opt | upRPM | dwRPM | sftT | tfall1T | criseT | cfall1T | triseT | dt1 | dt2 | sftT | tfall1T | criseT | cfall1T | triseT | dt1 | dt2 |
|--------|-----|-------|-------|------|---------|--------|---------|--------|-----|-----|------|---------|--------|---------|--------|-----|-----|
| 0      | 0   | 4100  | 1350  | 0.5  | 0.02    | 0.03   | 0.26    | 0.3    | 0   | 0.1 | 0.51 | 0.02    | 0.03   | 0.25    | 0.3    | 0   | 0.1 |
| 500    | 0   | 4200  | 1300  | 0.5  | 0.02    | 0.03   | 0.28    | 0.3    | 0   | 0.1 | 0.52 | 0.02    | 0.03   | 0.25    | 0.3    | 0   | 0.1 |
| 1000   | 0   | 4300  | 1250  | 0.5  | 0.02    | 0.03   | 0.30    | 0.3    | 0   | 0.1 | 0.53 | 0.02    | 0.03   | 0.25    | 0.3    | 0   | 0.1 |
| 1500   | 0   | 4400  | 1200  | 0.5  | 0.02    | 0.03   | 0.32    | 0.3    | 0   | 0.1 | 0.54 | 0.02    | 0.03   | 0.25    | 0.3    | 0   | 0.1 |
| 2000   | 0   | 4500  | 1100  | 0.5  | 0.02    | 0.03   | 0.34    | 0.3    | 0   | 0.1 | 0.55 | 0.02    | 0.03   | 0.25    | 0.3    | 0   | 0.1 |

The *I*th gear properties table, where *I* is the gear ID, ranging sequentially from minimum to maximum (the total number of tables being MAX\_GEAR\_ID - MIN\_GEAR\_ID + 1).

There must be a table for each gear ID, and if the gear ID is negative an 'N' must be pre-pended to the absolute value ID (gear ID = -1 => GEAR\_N1\_PROPERTIES\_MAP).

The gear parameters in the table are the same of the standard gear controller, with the first column (GEAR\_ID) value substituted by the independent variable value (track s or time).

A full data block sample follows:

```
[GEAR_SCHEDULED_CONTROLLER]
CONTROLLER_DATA = 'GEAR_MAPPED'
ACTIVATION = 'TRUE'
MODEL = 'STANDARD'
MAX_GEAR_ID = 5
MIN_GEAR_ID = 0
X_DATA = 'PATH_S'
(GEAR_0_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfall1T criseT cfall1T triseT dt1 dt2 sftT tfall1T criseT cfall1T triseT dt1 dt2 }
 0 0 500 0 0.5 0.021 0.031 0.25 0.25 0 0.1 0.45 0.02 0.03 0.25 0.3 0 0.11
 500 0 550 0 0.5 0.022 0.032 0.25 0.25 0 0.1 0.46 0.02 0.03 0.25 0.3 0 0.12
 1000 0 600 0 0.5 0.023 0.033 0.25 0.30 0 0.1 0.47 0.02 0.03 0.25 0.3 0 0.13
 1500 0 650 0 0.5 0.025 0.034 0.25 0.35 0 0.1 0.48 0.02 0.03 0.25 0.3 0 0.14
 2000 0 700 0 0.5 0.030 0.035 0.25 0.40 0 0.1 0.50 0.02 0.03 0.25 0.3 0 0.15
(GEAR_1_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfall1T criseT cfall1T triseT dt1 dt2 sftT tfall1T criseT cfall1T triseT dt1 dt2 }
 0 0 4100 1100 0.4 0.02 0.03 0.25 0.32 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
 500 0 4150 1150 0.45 0.02 0.03 0.25 0.34 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
 1000 0 4200 1200 0.50 0.02 0.03 0.25 0.36 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
 1500 0 4250 1250 0.55 0.02 0.03 0.25 0.38 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
 2000 0 4300 1300 0.50 0.02 0.03 0.25 0.30 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
(GEAR_2_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfall1T criseT cfall1T triseT dt1 dt2 sftT tfall1T criseT cfall1T triseT dt1 dt2 }
```

## Appendix B: File Format

```

0 0 4200 1100 0.4 0.02 0.03 0.25 0.32 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
500 0 4250 1150 0.45 0.02 0.03 0.25 0.34 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1000 0 4300 1200 0.50 0.02 0.03 0.25 0.36 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1500 0 4350 1250 0.55 0.02 0.03 0.25 0.38 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
2000 0 4300 1300 0.50 0.02 0.03 0.25 0.30 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1

(GEAR_3_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfallT criseT cfallT triseT dt1 dt2 sftT tfallT criseT cfallT triseT dt1 dt2 }
0 0 4100 1000 0.4 0.02 0.03 0.25 0.32 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
500 0 4150 1050 0.45 0.02 0.03 0.25 0.34 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1000 0 4200 1100 0.50 0.02 0.03 0.25 0.36 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1500 0 4250 1150 0.55 0.02 0.03 0.25 0.38 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
2000 0 4300 1200 0.50 0.02 0.03 0.25 0.30 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1

(GEAR_4_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfallT criseT cfallT triseT dt1 dt2 sftT tfallT criseT cfallT triseT dt1 dt2 }
0 0 4100 1100 0.4 0.02 0.03 0.25 0.32 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
500 0 4150 1150 0.45 0.02 0.03 0.25 0.34 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1000 0 4200 1200 0.50 0.02 0.03 0.25 0.36 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1500 0 4250 1250 0.55 0.02 0.03 0.25 0.38 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
2000 0 4300 1300 0.50 0.02 0.03 0.25 0.30 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1

(GEAR_5_PROPERTIES_MAP)
{ path_s opt upRPM dwRPM sftT tfallT criseT cfallT triseT dt1 dt2 sftT tfallT criseT cfallT triseT dt1 dt2 }
0 0 4400 1300 0.4 0.02 0.03 0.25 0.32 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.2
500 0 4550 1350 0.45 0.02 0.03 0.25 0.34 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
1000 0 4600 1300 0.50 0.02 0.03 0.25 0.36 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.2
1500 0 4650 1350 0.55 0.02 0.03 0.25 0.38 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.1
2000 0 4500 1400 0.50 0.02 0.03 0.25 0.30 0 0.1 0.5 0.02 0.03 0.25 0.3 0 0.2

```

## 5.11.12 HUMAN\_CONTROLLER Block

The HUMAN controller is an additional module selecting a more "human" style, or "skill" of driving. The skills currently supported are:

- 1. ROBOT: no human action. This is basically switching-off the human module allowing a standard VI-Driver simulation without human corrections or delays.
- 2. PROFESSIONAL: the driving skills are tailored on a professional driver (pilot).
- 3. STANDARD: simulating the abilities of an average driver.
- 4. NOVICE: the overall capabilities of the driver are scaled down to the basic driving skills.

The controller is activated if listed among the maneuver controllers. See [HUMAN](#) in the MANEUVER data block.

The parameters data block is quite simple, defining the data type, activation and skill type.

**CONTROLLER\_DATA = 'HUMAN'**

The controller data type. Must be "HUMAN".

**ACTIVATION = 'TRUE'**

Activates the controller.

**MODEL = 'BASIC'**

The HUMAN controller model type. Currently only 'BASIC' is supported.

**SKILL = 'STANDARD'**

The skill type. The supported options are 'ROBOT', 'PROFESSIONAL', 'STANDARD', 'NOVICE'.

Example data block:

[HUMAN\_CONTROLLER]

```
CONTROLLER_DATA = 'HUMAN'
ACTIVATION = 'TRUE'
SKILL = 'STANDARD'
```

## 5.11.13 CLUTCH\_STANDARD Block

```
[CLUTCH_STANDARD]
MAX_VALUE = 1
MIN_VALUE = 0
```

Saturation limits for the clutch demand. Both open loop and machine driven throttle computation will be saturated in the specified range.

```
SCALING_FACTOR = 1
```

Scaling factor for the clutch demand. The controller internal computation is always in the range 0-1, but the clutch demand could be scaled in order to fit the model being driven. Please make sure that the specified scaling match the saturation parameters above.

```
MIN_RPM = 500
```

Minimum engine speed for enabling the automatic clutch opening suitable for vehicle idling simulation.

```
STARTUP_START_RPM = 4000
```

Desired engine speed for initiating the stand still startup maneuver. The throttle will be operated in order to have the engine running at the specified speed (rpm)

```
STARTUP_START_TIME = 3.0
```

Time delay respect the beginning of the startup maneuver at which the clutch starts to be operated in order to accelerate the vehicle.

```
STARTUP_TARGET_AX = 0
```

Desired acceleration during the startup transient. Please note that the specified target AX is added to the target acceleration defined in the longitudinal controller map.

```
STARTUP_CLUTCH_FALL_TIME = 2.0
```

Basic time used to release the clutch. The actual time will be in general greater because it depends on the vehicle properties and the startup conditions ( the driver objective is to control the engine speed using the clutch demand )

```
STARTUP_PID_PROP_GAIN = 0.0008
STARTUP_PID_INTG_GAIN = 0.005
STARTUP_PID_DERIV_GAIN = 0.0
```

Gains of the PID controlling the clutch release process

```
STARTUP_THROTTLE_PID_PROP_GAIN = 4
STARTUP_THROTTLE_PID_INTG_GAIN = 10
STARTUP_THROTTLE_PID_DERIV_GAIN = 0.0
```

Gains of the PID controlling the throttle action before the clutch is engaged

```
THROTTLE_CONTROL_ACTIVATION = 'TRUE'
```

Set this to false to disable the throttle control during the gearshift operation. Note that from v18 this flag affects also open loop throttle signals.

```
ACTIVATION = 'TRUE'
MODEL = 'STANDARD'
```

## 5.11.14 TIRE\_BASIC\_CONTROLLER Block

As extension to the basic set of controllers for steering, throttle, brake, etc, VI-Driver supports auxiliary controllers responsible for managing specific scenarios. The Tire controller is a an aggregate module designed to respond to the tire states generating correction for the throttle, brake, gear and clutch output. Specific parameters to tune the correction are included in the block.

**Note:** The tire controller isn't available for VI-Driver Matlab interface and VI-Driver FMI interface.

```
[TIRE_BASIC_CONTROLLER]
CONTROLLER_DATA = 'TIRE'
```

Declare the current block as data for the Tire controller.

```
ACTIVATION = 'TRUE'
```

Set the state of the current Tire controller to active

```
THROTTLE_CONTROL_ACTIVE = 'FALSE'
BRAKING_CONTROL_ACTIVE = 'FALSE'
CLUTCH_CONTROL_ACTIVE = 'FALSE'
```

Set the activation state for the throttle, brake and clutch correction. The trigger condition is the minimum vertical force on tires dropping below MIN\_FZ parameter (all tires should match the criteria). In such a case the throttle and brake are released and the clutch is disengaged until when the triggering condition disappear

```
BASIC_FILTER_CUTFRQ = 3.1831
```

Cutoff frequency for restoring throttle, brake and clutch signals when minimum vertical force check is greater than MIN\_FZ

```
MIN_FZ = 5
```

Threshold on minimum vertical force leading to controller activation. This parameter is used by all the controller sub modules

```
GEAR_CONTROL_ACTIVE = 'TRUE'
```

Set the activation flag for the gear correction. This sub module prevents gear operation (up shifting) when at least one tire is sliding (slip greater than GEAR\_LONSLIP\_THRESHOLD threshold) or minimum vertical force is smaller than MIN\_FZ

```
GEAR_LONSLIP_THRESHOLD = 0.2
```

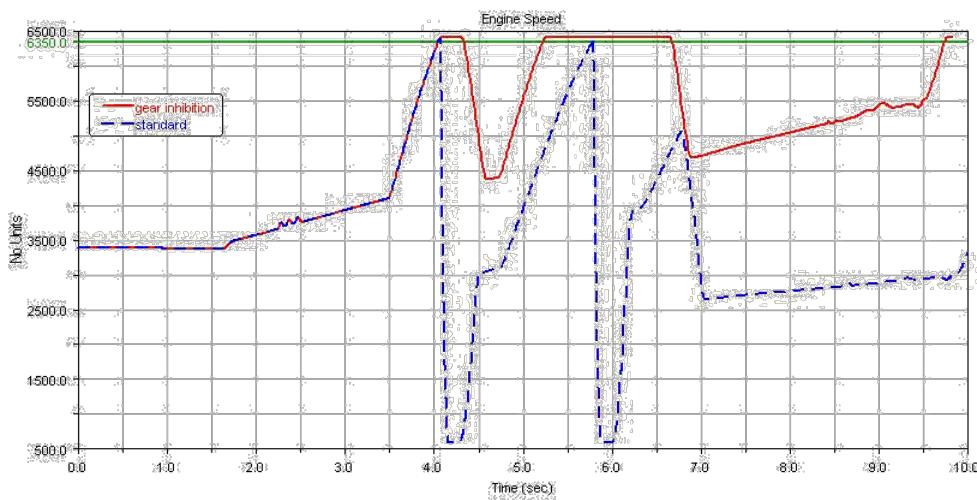
Longitudinal slip threshold leading to gear correction activation (valid range 0 to 1)

```
GEAR_INHIBITION_DELAY = 0.1
```

Minimum duration of the gear inhibition. When the inhibition condition occur, it remains active for at least GEAR\_INHIBITION\_DELAY time units

```
GEAR_MAX_INHIBITION_TIME = 3
```

Maximum duration of the gear shifting inhibition. This is a safety check to make sure that when the system could not recover from sliding within the specified time, the gear shifting inhibition is turned off.



```
LATERAL_SLIP_COMPUTATION_ACTIVE = 'FALSE'
```

The modified tire lateral slip computations activation flag. When active (TRUE) the tire lateral slip seen by the driver is modified according additional computations. Please note that currently only the motorcycle flavour of VI-Driver is affected by the the lateral slip states.

```
LATERAL_SLIP_COMPUTATION_SMOOTHING_TIME = 0.1
```

The modified tire lateral slip smoothing time (STANDARD computations type).

### 5.11.15 TIRE\_LONSLIP\_STANDARD Block

The longitudinal slip controller is designed to monitor the selected tires longitudinal slip and modulate the throttle demand on tire spinning.

**Note :** The tire longitudinal slip controller isn't available for VI-Driver Matlab interface and VI-Driver FMI interface.

```
[TIRE_LONSLIP_STANDARD]
CONTROLLER_DATA = 'TIRE_LONSLIP'
```

Declare the current block as data for the Tire controller.

```
ACTIVATION = 'TRUE'
```

Set the state of the current Tire controller to active

```
MAX_ALLOWED_SLIP = 0.05
MIN_ALLOWED_SLIP = -1
```

Controller activation threshold: the traction controller will start working when at least one of the selected tires is exceeding the maximum allowed slip. The minimum slip usually triggers the de-clutching (Rider mode)

```
ACTIVE_TIRE_1 = 1
ACTIVE_TIRE_2 = 2
ACTIVE_TIRE_3 = 3
ACTIVE_TIRE_4 = 4
```

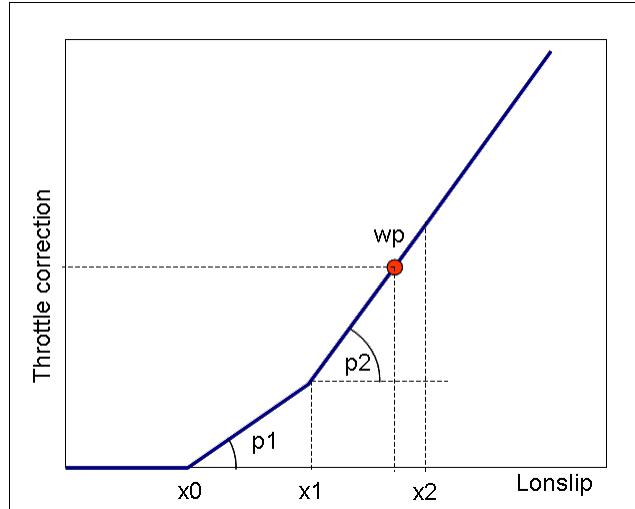
Tires ID to be monitored. For cars IDs 1 and 2 belongs to front tires while IDs 3 and 4 to the rear ones. In motorcycle mode ID 1 is the front tire and ID 2 is the rear one. IDs assignment may change according to

## Appendix B: File Format

the driver interface being used. For car applications, up to 4 tires can be monitored for slip control: in this case all the four ACTIVE\_TIRE\_ keys should be set to a different tire id (1,2,3,4)

```
CORRECTION_GAIN_1 = 100
REF_SLIP_1_COEFF = 0.85
REF_SLIP_2_COEFF = 0.9
```

The throttle correction factor is computed as a bilinear shape w. This parameters is used to scale the throttle demand according to the current slip level. A bilinear correction factor is used as shown in the following figure:

**Where:**

- $x_2$  is the max longitudinal slip
- $x_1$  is computed as  $x_2 * \text{REF\_SLIP\_2\_COEFF}$
- $x_0$  is computed as  $x_2 * \text{REF\_SLIP\_1\_COEFF}$
- $p_1$  is **CORRECTION\_GAIN\_1**
- $p_2$  is  $3 * p_1$

## 5.11.16 STARTUP Block

```
[STARTUP]
INITIAL_SPEED = 35
```

Assign the initial longitudinal speed for the maneuver.

```
INITIAL_STEERING = 0
```

Define the initial condition for the steering demand

```
INITIAL_THROTTLE = 0
```

Define the initial condition for the throttle demand

```
INITIAL_BRAKING = 0
```

Define the initial condition for the brake demand

```
INITIAL_GEAR = 3
```

Define the initial condition for the gear demand. **INITIAL\_CLUTCH = 0**

Define the initial condition for the clutch demand.

```
INITIAL_SETUP = 'OFF'
```

The vehicle initialization could be performed in two different ways:

1. OFF: a standard static analysis is performed and the initial condition for the driver demand are inherited from the INITIAL\_\* keys
2. STRAIGHT: a more sophisticated initialization process based on quasi-static analyses. In this mode only initial speed and gear are effective.

Please note that from this version the option **INITIAL\_SETUP = 'STANDARD'** is deprecated, but still supported if found in an event file.

**Note:** **INITIAL\_SETUP = 'STRAIGHT'** isn't available for VI-Driver Matlab interface and VI-Driver FMI interface.

## 5.11.17 MANEUVER\_LIST Block

The maneuvers list table store the sequence of maneuver that should be performed.

```
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'MANEUVER_1' 109.37 0.01
```

**name**

name of the maneuver to execute. it should match the name of an existing maneuver block defined in the current VDF file. Please note that the name entry is case sensitive

**abort\_time**

maximum duration of the current maneuver. the actual maneuver duration could be smaller than the abort time due to additional [end condition](#) specified in the maneuver block. Using -1 the check on the maneuver duration is disabled.

**sampling\_step**

driver internal calculation step. the driver states are updated every sampling\_step time units.

## 5.11.18 MANEUVER Block

The MANEUVER block is constituted by the following statements:

- **TASK = 'STANDARD'**
- **CONTROLLER\_SETTINGS = 'CONTROLLER\_STANDARD'**

The **CONTROLLER\_SETTINGS** key designates which [controller block](#) the active maneuver should reference.

and by some definition lists, grouped in the following sub-blocks (some are optional):

- [STEERING](#)
- [THROTTLE](#)
- [BRAKING](#)
- [GEAR](#)
- [CLUTCH](#)
- [MACHINE](#)
- [TIRE](#)
- [OPTIONS](#)

---

*Appendix B: File Format*

- [END CONDITION](#)

The `TASK = 'STARTUP'` declares the current maneuver is designed to start the vehicle from standstill condition. When throttle and clutch are set to machine, VI-Driver will operate them to increase the engine speed and then move the vehicle releasing the clutch

## Control Channels

The control channels are:

- STEERING
- THROTTLE
- BRAKING
- GEAR
- CLUTCH

Each control channel points to an controller specific block defining the specific controller setup:

```
(STEERING)
CONTROLLER_NAME = 'MANEUVER_1_STEERING'
```

where the `MANEUVER_1_STEERING` is consistent with the following format:

```
[MANEUVER_1_STEERING]
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
```

the supported options for the block are:

```
METHOD = 'MACHINE'
```

For each control channel the `METHOD` key states in which way the channel itself should be computed. Each channel can be set to the desired method independently from the other channels. The available options are:

- [MACHINE](#)
- [OPENLOOP](#)

```
SIGNAL = 'SINE_AMP_1'
```

When the `open-loop` mode is requested, an additional key should be provided in order to identify the [open loop block](#) to get signal parameters from ('`SINE_AMP_1`').

```
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
```

Define the activity and the duration of the signal smoothing at the beginning of the maneuver. When the smoothing is enabled a step function is applied to the computed channel value (both openloop and machine) in order to guarantee the signal continuity at each maneuver startup. The smoothing time activity and duration are specific for each channel.

Older VDF format with direct definition of the controller properties inside the maneuver block are still supported.

```
(STEERING)
METHOD = 'MACHINE'
SIGNAL = 'SINE_AMP_1'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
```

VI-EventBuilder will automatically update older VDF files to the new format during the editing process.

## MACHINE

```
(MACHINE)
PATH = 'PATH'
```

Identify the block containing the path information for the machine controller. The path key should reference an existing [PATH block](#) in the VDF file. When the steering controller is set to open loop the path key is not required.

```
SPEED = 'SPEED'
```

Identify the block containing the target for the longitudinal controller.

The speed key should reference an existing block of one of the following types:

- **TYPE='MAP'**

Using the map mode, an additional table for entering the speed function of path abscissa will be shown. The map block is responsible for defining the map class (speed, acceleration, etc). When the longitudinal controller is set to open loop the path key is not required. See [map block](#) for more details.

- **TYPE='MAINTAIN'**

The target speed will be constant and equal to the initial speed.

- **TYPE='DRD\_FILE'**

The target speed is extracted from the speed column of a DRD file. The Speed DRD File could be same supplied as Path DRD file. Please refer to the DRD file format for more information about the DRD files.

## TIRE

```
(TIRE)
CONTROLLER_NAME = 'TIRE_BASIC_CONTROLLER'
```

This is an optional block defining which data block to use to configure the tire controller. For more info refer to [TIRE\\_BASIC\\_CONTROLLER](#).

## OPTIONS

```
(OPTIONS)
CONNECT_PATH = 'FALSE'
```

Set this value to `TRUE` to connect the path of the current maneuver to the path of the previous maneuver. This parameter is neglected for the first maneuver.

```
FOLLOW_TARGET_SPEED = 'FALSE'
```

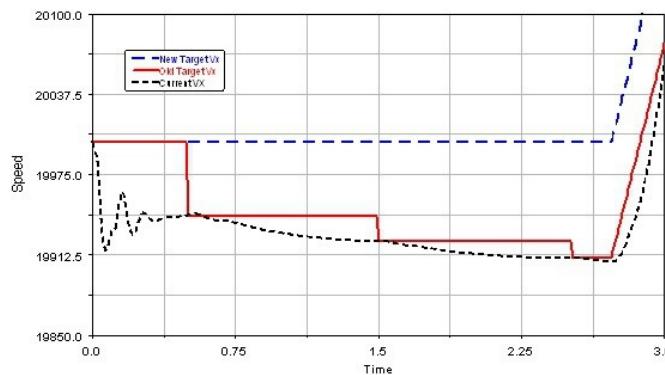
This option is useful when switching to a maneuver with an **acceleration** map or a **maintain** speed: the new target speed value will be set to actual vehicle longitudinal velocity. Set to `TRUE` to have initial target speed value equal to the **last target speed** value of the previous maneuver.

**Note:** If the map is defined as a **speed** map, this option is useless: the target speed will always be set to the value defined in the map.

**Note:** If the previous maneuver is an open-loop maneuver, this option is useless: the target speed will always be set to the actual velocity (if the previous maneuver is an open-loop maneuver, the target speed is set to 0, so the new target speed would be always zero).

## Appendix B: File Format

**Note:** in the previous release, when **FOLLOW\_TARGET\_SPEED** was set to **TRUE**, when switching among maneuvers with **maintain** speed, the target speed value was set to the last **actual** speed vehicle longitudinal velocity instead of the right last **target** speed vehicle longitudinal velocity.

**SPEED\_MAP\_ABS\_TIME = 'FALSE'**

When using a speed map defined as function of time, time is relative to that maneuver beginning time, and not to the first maneuver start time.

To have maps defined as function of absolute time, set this value to **TRUE**.

**SPEED\_MAP\_GLOBAL\_S = 'FALSE'**

When using a speed map defined as function of arclength, the local maneuver s\_coordinate is used. To have maps defined as function of glocal s\_coordinate, set this value to **TRUE**.

**END\_OF\_PATH\_CHECK = 'TRUE'**

Simulation stops when the path ends. Set this to '**FALSE**' to run more than one lap.

When this option is set to true, the simulation stops when the preview point (not the vehicle) is at 5 meters before the end of the path.

**PATH\_POSITIONING = 'ABSOLUTE'**

See the [Global Settings Block](#) for this option. Please note that when the **CONNECT\_PATH = 'TRUE'** option is set for maneuver id greater than 1, whatever setting of **PATH\_POSITIONING** will be masked by the path connection phase.

## GEAR\_SCHEDULED

(**GEAR\_SCHEDULED**)  
**CONTROLLER\_NAME = 'GEAR\_SCHEDULED\_CONTROLLER'**

The scheduled gear controller activation and basic parameters description. It points to the data block containing the scheduled gear data. For more info refer to [GEAR\\_SCHEDULED](#) controller.

## HUMAN

(**HUMAN**)  
**CONTROLLER\_NAME = 'HUMAN\_CONTROLLER'**

Activates the human controller addressing the related parameters description. It points to the data block containing the human controller actual data. For more info refer to [HUMAN](#) controller.

## END\_CONDITION

(END\_CONDITIONS)

List the set of [end condition](#) to check for the active maneuver.

**Note:** If the end condition block specified here doesn't exist, the end condition will be ignored.

```
{ name }
'DISTANCE_TRAVELLED'
'TIME_SPENT'
```

## 5.11.19 MAP Block

[SPEED]

TYPE = 'MAP'

Identify the block as a map block. An alternative method to supply a map is using a [DRD file](#). In this case the block should contain the key TYPE = 'DRD\_FILE'

X\_DATA = 'PATH\_S'

Independent map data type. The X\_DATA key supports the following options:

- PATH\_S
- TIME

Y\_DATA = 'LONVEL'

Dependent map data type. The Y\_DATA key supports the following options:

- LONVEL
- LONACC

SCALING\_FACTOR = 1.0

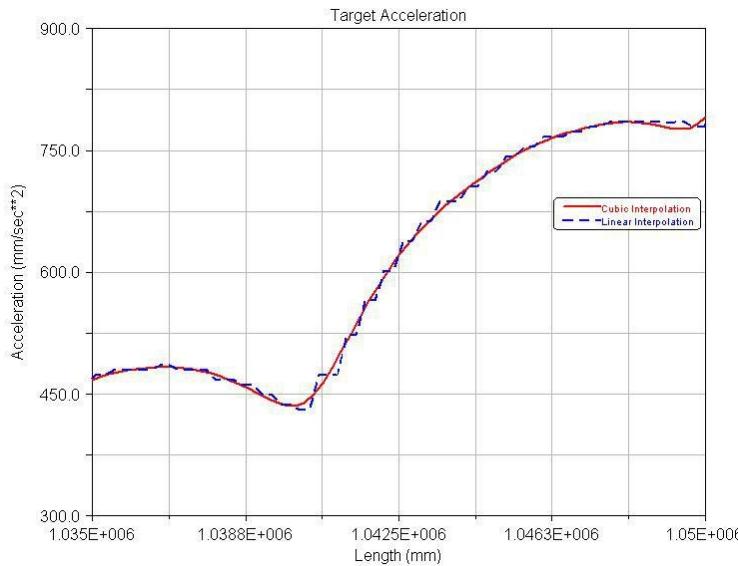
Scale the speed values by the given factor.

INTERPOLATION = 'LINEAR'

Define the interpolation method to be applied to the current map. The available options are:

- LINEAR
- 
- CUBIC\_SPLINE
- QUINTIC\_SPLINE

Please note that using Cubic and Quintic spline interpolation may introduce undesired oscillations in the given map especially when a small number of points are provided. The following plot shows the resulting target acceleration profile in case of linear and cubic interpolation:



```
SMOOTHING_FACTOR = 0.0
```

In addition to the pure interpolation a smoothing functionality could be applied to the map preprocessing: in this case an additional parameter is required in order to identify the amount of smoothing allowed.

Please note that `SMOOTHING_FACTOR=0.0` make the smoothing algorithm work as a pure interpolator (no path deformation).

#### (VALUES)

```
{ x y }
 0 35
 5 35
 10 35
 15 35
 20 35
 25 35
 30 35
 ...
 ...
3705 31.26
3710 31.71
3715 32.15
3720 32.57
3725 32.97
```

#### DRD MAP Block

```
[SPEED]
TYPE = 'DRD_FILE'
FILE_NAME = 'mdids://shared/driver_roads.tbl/track.drd'
```

Specify the drd from which building the map speed = f(s).

```
SCALING_FACTOR = 1.0
```

Scale the speed values by the given factor.

**INTERPOLATION = 'LINEAR'**

Define the interpolation method to be applied to the current map. The available options are:

- LINEAR
- 
- CUBIC\_SPLINE
- QUINTIC\_SPLINE

**Notes:**

- using Cubic and Quintic spline interpolation may introduce undesired oscillations in the given map especially when a small number of points are provided.
- when speed map is defined sharing a datablock with a path with a non linear interpolation mode, the same interpolation mode will be applied to the speed profile.

**SMOOTHING\_FACTOR = 0.0**

In addition to the pure interpolation a smoothing functionality could be applied to the map preprocessing: in this case an additional parameter is required in order to identify the amount of smoothing allowed.

Please note that `SMOOTHING_FACTOR=0.0` make the smoothing algorithm work as a pure interpolator (no path deformation).

## 5.11.20 PATH Block

**[PATH]****TYPE = 'DRD\_FILE'**

Identify the block as a path block

The supported path type are:

- [DRD\\_FILE](#)
- [NUMERIC\\_DATA](#)
- [SHAPE](#)

### DRD Type

**[PATH]****TYPE = 'DRD\_FILE'****FILE\_NAME = 'mdids://shared/developer\_roads.tbl/track.drd'**

Specify the drd from which building a path

**INTERPOLATION = 'NONE'**

The driver controller requires a quite detailed path description in order to work properly. The recommended path resolution is of 0.25 meter. The consequence of using such a small step size is that the drd size could increase dramatically for a full track. The solution is to let the driver interpolate over the given input (sampled at the standard resolution of about 1 meter) to generate a more detailed path description. The supported options for the interpolation feature are:

- NONE
- CUBIC\_SPLINE

**MIN\_SEGMENT\_LENGTH = 0.25**

Output step size for the interpolation algorithm.

## NUMERIC\_DATA Type

```
[PATH]
TYPE = 'NUMERIC_DATA'
(TABLE)
{ x y z lat_inc }
x1 y1 z1 A1
x2 y2 z2 A2
...
xn yn zn An
```

Specify the values for x,y,z and lateral inclination for the path.

## SHAPE STRAIGHT Type

```
[PATH]
TYPE = 'SHAPE'
SHAPE_NAME = 'STRAIGHT'
```

Specify the shape of the path to build, currently supported shapes are [STRAIGHT](#) and [SKIDPAD](#)

## SHAPE SKIDPAD Type

```
[PATH]
TYPE = 'SHAPE'
SHAPE_NAME = 'SKIDPAD'
```

Specify the shape of the path to build, currently supported shapes are [STRAIGHT](#) and [SKIDPAD](#)

**RADIUS = 100**

Skidpad radius

**TURN\_SIDE = 'LEFT'**

LEFT and RIGHT value supported

**STARTING\_LENGTH = 200**

Straight part of the path before starting skidpad. Multiply PREVIEW\_TIME, INITIAL\_SPEED to obtain minimum STARTING\_LENGTH

**Tip:** set STARTING\_LENGTH greater than PREVIEW\_TIME \* SPEED for better stability.

**CLOSING\_LENGTH = 0**

The closing straight part of the path.

**AMPLITUDE = 360**

Path length is defined from angle amplitude to travel. End of path is the greater value between OVERALL\_LENGTH and AMPLITUDE.

Default value is  $2.1 \times \pi$ .

**OVERALL\_LENGTH = 1000**

Path length, considering STARTING\_LENGTH and turning path.

End of path is the greater value between OVERALL\_LENGTH and AMPLITUDE

```
STEP_SIZE = 0.1
```

```
SMOOTHING_AMPLITUDE = 20
```

To allow smooth curvature changes, the Euler spiral is used in the "smoothing amplitude" part of the curved path, to connect the straight part of the path to the skidpad.

```
CLOSING_SMOOTHING_AMPLITUDE = 0
```

This parameter defines the smoothing amplitude of the final section of the skidpad (from constant curvature 1/R to zero curvature). A value of zero (default) means the skidpad keeps always a constant curvature.

## 5.11.21 END\_CONDITION Block

```
[DISTANCE_TRAVELLED]
```

```
MEASURE_NAME = 'TRAVELED_DISTANCE'
```

Name of the measure provided in the driver interface.

The driver built in end conditions are:

- **TIME** time elapsed in the current maneuver
- **GLOBAL\_TIME** time elapsed since the first maneuver has started
- **TRAVELED\_DISTANCE** the distance travelled in the current maneuver
- **GLOBAL\_TRAVELED\_DISTANCE** the global distance travelled
- **PATH\_S** the absolute path s

In the base product some external end conditions could be predefined: in this case the external conditions overwrite the built-in conditions with the same name.

For VI-CarRealTime, external end conditions are:

- **RADIUS** the trajectory radius at the gyro position
- **CURVATURE** the trajectory curvature at the gyro position
- **GYRO\_DX** the longitudinal distance traveled at the gyro position
- **GYRO\_DY** the lateral distance traveled at the gyro position
- **VELOCITY** the vehicle speed
- **LON\_ACCEL** the vehicle longitudinal acceleration
- **LAT\_ACCEL** the vehicle lateral acceleration
- **YAW\_ANGLE** the vehicle yaw angle
- **YAW\_RATE** the vehicle yaw rate
- **YAW\_ACCEL** the vehicle yaw acceleration
- **PITCH\_ANGLE** the vehicle pitch angle
- **ROLL\_ANGLE** the vehicle roll angle
- **ROLL\_VEL** the vehicle roll velocity
- **SIDE\_SLIP\_ANG** the vehicle side slip angle
- **STEERING\_ANG** the driver steering angle
- **STEER\_ANG\_VEL** the driver steering angle velocity
- **ENGINE\_SPEED** the engine
- **ENGINE\_TORQUE** the engine torque
- **ENGINE\_ACC** the engine revolute acceleration
- **CLUTCH\_TORQUE** the clutch torque
- **CLUTCH\_DEMAND** the driver clutch demand

**Note:** Each measure is expressed according to the units block of the vdf file.

```
TEST_TYPE = '>>'
```

Check to perform on the run time value of the given measure. Supported checks are:

- greater than '>>'

## Appendix B: File Format

- greater equal than '>='
- smaller than '<<'
- smaller equal than '<='
- equal to '==' (be careful to default [TOLERANCE](#) value)
- different from '<>' (be careful to default [TOLERANCE](#) value)
- absolute value greater than '|>|'
- absolute value greater equal than '|>=|'
- absolute value smaller than '|<<|'
- absolute value smaller equal than '|<=|'
- absolute value equal to '|=|' (be careful to default [TOLERANCE](#) value)
- absolute value different from '|<>|' (be careful to default [TOLERANCE](#) value)

**REF\_VALUE = 10000**

Triggering value for the end condition.

**Optional values**

The following optional values, if not specified, are set by default to zero.

**TOLERANCE = 1e-6** (default value = 0)

The tolerance used checking the end conditions.

When using the equal test type, be careful to set this option to a value greater then zero, otherwise the trigger end condition could never be caught, for example:

```
MEASURE_NAME = 'TIME'
TEST_TYPE = '=='
REF_VALUE = 2
```

In this case time could have the value 1.999999999 (double precision roundoff) and the check condition will never be caught.

To avoid similar situations, set a tolerance different from zero, or use a different test type ('>>' or '>='), for example).

**FILTERING\_TIME = 1** (default value = 0)

The end condition will have to be true for a time equal to the filtering time value, before stopping the simulation.

Useful in following scenario, for example suppose to have created a condition sensor to monitor the lateral acceleration, and the end condition uses this sensor value:

```
MEASURE_NAME = 'lat_accel'
TEST_TYPE = '>>'
REF_VALUE = 2
FILTERING_TIME = 1
```

The simulation will stop only when the lateral acceleration will have been greater than the [REF\\_VALUE](#) for the entire [FILTERING\\_TIME](#).

**DELAY = 0.5** (default value = 0)

When the end condition will be reached, the simulation will wait a [DELAY](#) time before stopping.

**NEXT\_MANEUVER\_NAME = 'MAN\_NAME'**

Force the execution of a specific maneuver when this end condition triggers.

**Note:** the following lines, taken from a .msg analysis file, show a situation that could cause some doubts:

```
<<< VI-Driver MANEUVER INFO >>>
Maneuver 1 (MANEUVER_1) end condition(s) reached...
(time = 0.11s / simulation time = 0.11s)
```

```
Satisfied end condition info:
== condition 1 ==
name : MANEUVER_1_END_COND_BLOCK
ID : 0
test type : >=
reference value : 0.1
tolerance value : 0
measure value : 0.11
```

The discrete driver end condition checking is switching to maneuver 2 at the first valid computation after time 0.1 so at time 0.11 it is ALREADY processing maneuver 2 (as shown by the maneuver id monitor).

## 5.11.22 OPEN SIGNALS Block

Several predefined open loop signal cold be generated using a signal block. Each signal block could then be assigned to a control channel of the desired maneuver:

```
METHOD = 'OPENLOOP'
SIGNAL = 'OPEN_SIGNAL_EXAMPLE'
```

where OPEN\_SIGNAL\_EXAMPLE is the name of an existing open loop signal block. A signal block contains some general signal definition keys that identify the signal to be produced and the calculation mode:

```
[OPEN_SIGNAL_EXAMPLE]
TYPE = 'STEP'
```

The supported open loop signals are:

- [CONSTANT](#)
- [STEP/STEP5](#)
- [SINE](#)
- [COSINE](#)
- [SWEPT](#)
- [IMPULSE](#)
- [RAMP](#)
- [DCD](#)

In addition to the signal specific parameters some general signal modifier keys are available:

```
CALC_MODE='RELATIVE'
```

Define if the final value for the open loop signal should be computed as relative to the initial value. The supported options are:

- RELATIVE
- ABSOLUTE

```
AUTO_SCALING = 'TRUE'
```

When the auto scaling option is set to TRUE, the signal computed is automatically scaled according to the scaling factor defined for the channel to which the open loop signal is assigned. For example if the [throttle scaling factor](#) is 100 and auto scaling is TRUE, a sine signal with amplitude 1 will be sent to the

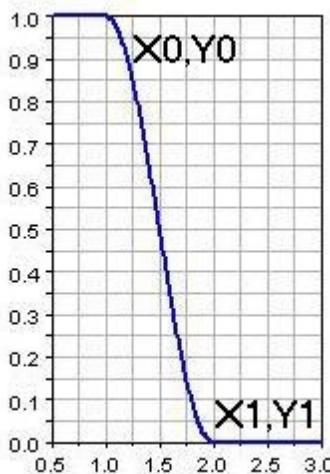
*Appendix B: File Format*

model with amplitude 100. Deactivating the AUTO\_SCALING feature, the open loop signal is sent to the model with no modifications.

## STEP Block

```
[STEP_EXAMPLE]
TYPE = 'STEP'
CALC_MODE='RELATIVE'
X0 = 0
X1 = 1
Y0 = 0
Y1 = .20
```

The STEP open signal generates a continuous connecting function between the given pair of points (X0,Y0 and X1,Y1).

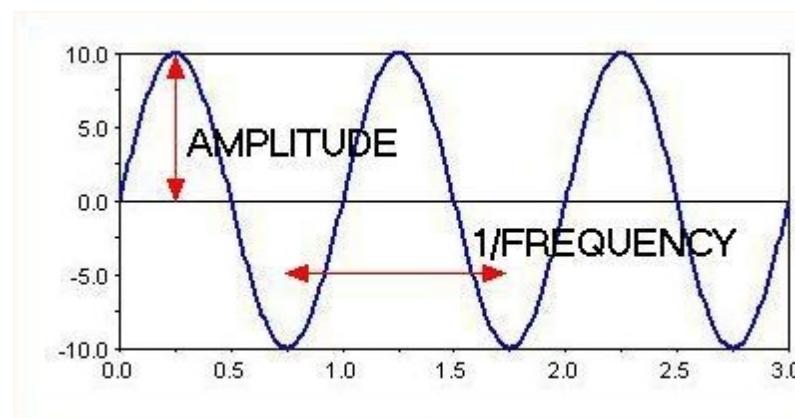


the connection function applied from X0 to X1 is a cubic polynomial unless the type is set to STEP5: in this case a 5th order polynomial is used.

## SINE Block

```
[SINE_EXAMPLE]
TYPE = 'SINE'
CALC_MODE='RELATIVE'
AMPLITUDE = 10
FREQUENCY = 1
INITIAL_PHASE = 0
NUMBER_OF_CYCLES = 10
```

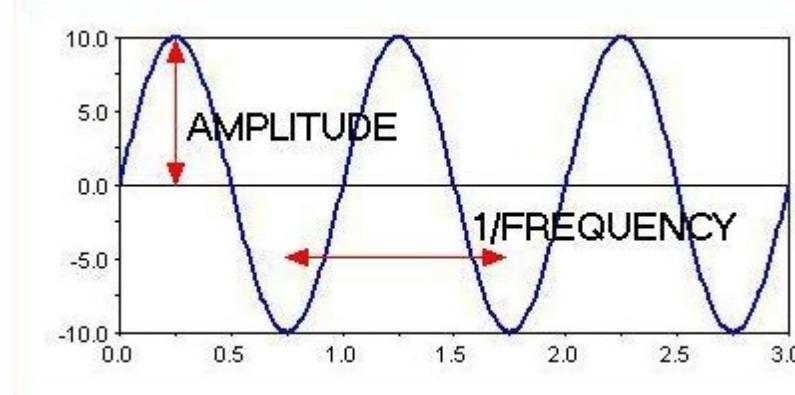
The SINE signal is described using the three basic parameters: amplitude, frequency and phase. The phase units depends on the units defined for angles while frequency is measured in 1/time (leading to hertz when time unit is second). The NUMBER\_OF\_CYCLES defines how many complete signal periods ( $2\pi X$ ) have to be computed (if the key/value pair is not specified the signal is periodically continuous).



## COSINE Block

```
[COSINE_EXAMPLE]
TYPE = 'COSINE'
CALC_MODE='RELATIVE'
AMPLITUDE = 10
FREQUENCY = 1
INITIAL_PHASE = -1.57
NUMBER_OF_CYCLES = 10
```

The COSINE signal is described using the three basic parameters: amplitude, frequency and phase. The phase units depends on the units defined for angles while frequency is measured in 1/time (leading to hertz when time unit is second). The NUMBER\_OF\_CYCLES defines how many complete signal periods ( $2\pi X$ ) have to be computed (if the key/value pair is not specified the signal is periodically continuous).



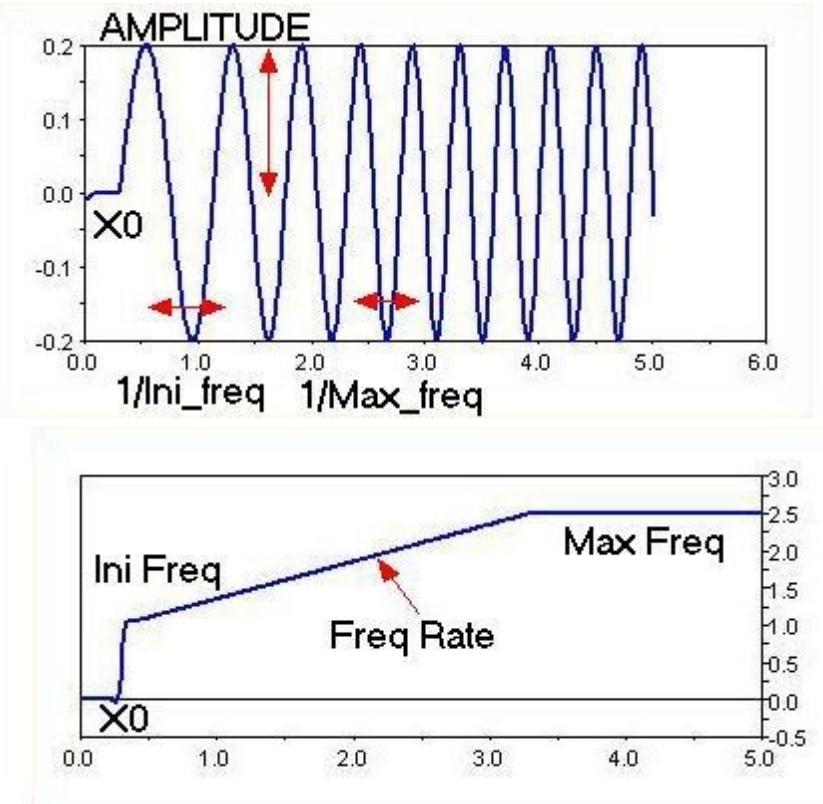
## SWEPT Block

```
[SWEPT_EXAMPLE]
TYPE = 'SWEPT'
CALC_MODE='RELATIVE'
X0 = 0.3
AMPLITUDE = 0.2
INITIAL_FREQUENCY = 1
FREQUENCY_RATE = 0.5
MAX_FREQUENCY = 2.5
```

The swept signal is a variable frequency sine signal characterized by a linear frequency rate. The initial and maximum frequency define the boundaries of the frequency sweep. The starting time for the swept signal generation could be set using the X0 parameter: until when the maneuver time is smaller than X0, a constant

## Appendix B: File Format

signal equal to the maneuver initial condition is generated. In addition the amplitude of the oscillation is defined by mean of the amplitude key.



The signal frequency is saturated at MAX\_FREQUENCY, so the swept signal becomes a standard sine signal when the frequency reach the specified upper boundary.

**CONSTANT Block**

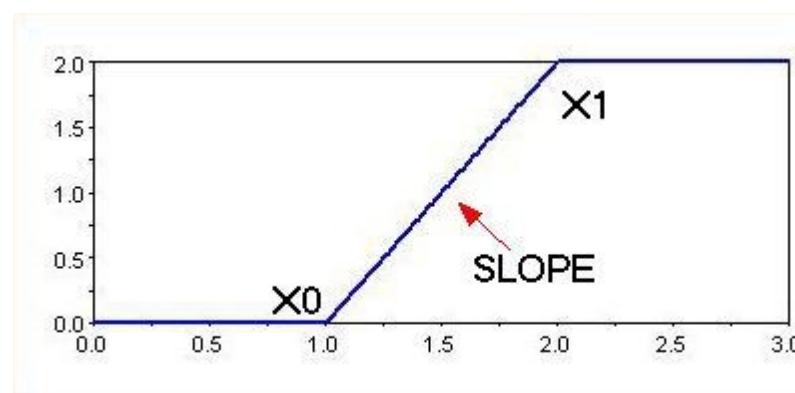
```
[CONSTANT_EXAMPLE]
TYPE = 'CONSTANT'
CALC_MODE='RELATIVE'
VALUE = 0
```

The constant signal type requires one single parameter to be defined that is the value of the constant signal. The supplied value is combined with the CALC\_MODE setting in order to actually produce the signal output. In the example above the combination of CALC\_MODE = 'RELATIVE' and VALUE = 0 produces an output equal to the maneuver initial condition for the current channel.

**RAMP Block**

```
[RAMP_EXAMPLE]
CALC_MODE='RELATIVE'
TYPE = 'RAMP'
X0 = 0.5
X1 = 3
SLOPE = .10
```

The RAMP signal is made of three section: when time is smaller than X0 the signal produced is the maneuver initial condition for the current channel, between X0 and X1 a constant slope function is computed and above X1 the signal maintain the level reached when time is equal to X1.

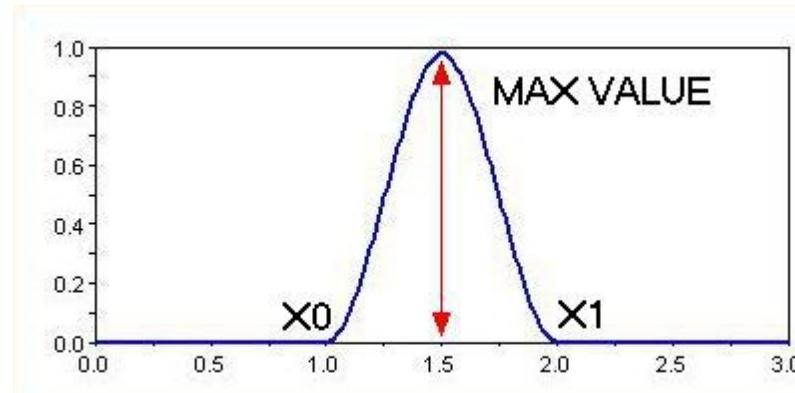


## IMPULSE Block

```
[IMPULSE_EXAMPLE]
TYPE = 'IMPULSE'
CALC_MODE='RELATIVE'
X0 = 3
X1 = 4
MAX_VALUE = 0.1
```

The impulse signal is obtained combining 2 step functions. the parameters for the impulse signal are:

- the start time (X0)
- the end time (X1)
- the amplitude (MAX\_VALUE)



The actual impulse peak value depends on the selected CALC\_MODE: when CALC\_MODE = 'RELATIVE', the maximum impulse value is the sum of MAX\_VALUE and maneuver initial value for the current channel.

## DCD Block

```
[DCD_EXAMPLE]
TYPE = 'MEASURED'
AUTO_SCALING = 'FALSE'
FILE_NAME = 'data_driven.dcd'
```

The MEASURED signal type is designed to extract measured values from an external data file (DCD format). The data file must contain information for all the control channels and the signal manager will automatically extract the one belonging to the channel the active signal is connected to.

An example of vdf file (*chicane\_R100\_25\_OPENLOOP.vdf*) referencing a dcd file (*chicane\_R100\_25\_data\_driven\_vdf.dcd*) is shipped with carrealtime\_shared database.

## Appendix B: File Format

The following example shows the structure of a DCD file containing measured channel information:

```
[MDI_HEADER]
FILE_NAME = data_driven.dcd
FILE_TYPE = 'dcd'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'Example .dcd file containing open loop data'
$-----UNITS
[UNITS]
LENGTH = 'meters'
FORCE = 'newton'
ANGLE = 'radians'
MASS = 'kg'
TIME = 'sec'
$-----OPEN_LOOP
[OPEN_LOOP]
ORDINAL = 'time'

(DATA)
{ time steering throttle brake gear clutch }
-0.1000E-00 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
-0.9000E-01 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
-0.8000E-01 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
```

The **ORDINAL** option can be set to **time**, **distance** or **path\_s** where path\_s is the actual position on the target path and distance is the distance travelled by the vehicle during the maneuver. The following example shows an OPEN\_LOOP Block defined in function of distance.

```
$-----OPEN_LOOP
[OPEN_LOOP]
ORDINAL = 'distance'

(DATA)
{ travel steering throttle brake gear clutch }
0.0001E-002 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
7.6631E-002 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
2.7647E-001 0.1465E-02 0.3016E+00 0.0000E+00 0.3000E+01 0.0000E+00
```

## USER\_Block

```
[USER_SIGNAL_EXAMPLE]
TYPE = 'USER'
MEASURE_NAME = 'USER_DEFINED_MEASURE'
CALC_MODE='ABSOLUTE'
```

The USER signal is a method to connect an external routine producing generic signals to the desired control channel. The MEASURE\_NAME parameter identifies the user signal should be used.

For the Adams flavor of VI-Driver the user signals are provided using the condition sensor element (as for end conditions).

The following channels:

```
MEASURE_NAME = 'USER_INPUT_STEERING'
MEASURE_NAME = 'USER_INPUT_THROTTLE'
MEASURE_NAME = 'USER_INPUT_BRAKING'
MEASURE_NAME = 'USER_INPUT_CLUTCH'
MEASURE_NAME = 'USER_INPUT_GEAR'
```

are automatically registered by VI-grade in the following VI-Driver interfaces:

- Adams Car
- CarRealTime
- Matlab
- FMI

for BikeRealTime, the braking signal is defined as:

```
MEASURE_NAME = 'USER_INPUT_BRAKE_FRONT'
MEASURE_NAME = 'USER_INPUT_BRAKE_REAR'
```

## 5.11.23 VDF Example 1

The following example shows how to setup VI-Driver for running a multimaneuver event in which different the target for the longitudinal controller is defined using different maps while the lateral controller is always set to follow a straight line. For the first and third maneuvers a time dependent speed map is used while an acceleration map function of time is used for the second maneuver. The transition among the different maneuvers is defined by specific end condition trigger.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'vdf'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'meter'
ANGLE = 'degree'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
$-----DEBUG_PARAMETERS
[DEBUG_PARAMETERS]
ACTIVATION = 'TRUE'
INITIALIZATION_DUMP = 'TRUE'
$-----GLOBAL_SETTINGS
[GLOBAL_SETTINGS]
MODEL_TYPE = 'CAR_STANDARD'
INTERFACE_TYPE = 'GENERIC'
$-----CONTROLLER_STANDARD
[CONTROLLER_STANDARD]
PREVIEW_TIME = 0.5
MIN_PREVIEW_DISTANCE = 5
LAT_ANT_COMP_TIME = 0
LON_ANT_COMP_TIME = 0
STEERING_PDC_ACTIVE = 'TRUE'
STEERING_YAW_PID_ACTIVE = 'TRUE'
THROTTLE_FEED_FWD_TRQ_ACTIVE = 'TRUE'
THROTTLE_SAT_TGTACX_ACTIVE = 'TRUE'
$-----STEERING_STANDARD
[STEERING_STANDARD]
```

## Appendix B: File Format

```

MAX_VALUE = 720
MIN_VALUE = -720
$-----THROTTLE_STANDARD
[THROTTLE_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
$-----BRAKING_STANDARD
[BRAKING_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
$-----GEAR_STANDARD
[GEAR_STANDARD]
MAX_VALUE = 5
MIN_VALUE = 0
(ATTRIBUTES_MAP)
{ id opt upRPM dwRPM sftT tfallT criseT cfallT triseT dt1 dt2 }
0 0 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
1 5 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
2 2.905 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
3 1.688 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
4 0.981 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
5 0.57 4100 1250 0.5 0.02 0.03 0.25 0.3 0 0.1
$-----STARTUP
[STARTUP]
STARTING_TIME = 0
INITIAL_SPEED = 20
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRAKING = 0
INITIAL_BRAKING2 = 0
INITIAL_GEAR = 3
INITIAL_CLUTCH = 0
INITIAL_SETUP = 'STRAIGHT'
$-----MANEUVERS_LIST
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'MANEUVER_1' 2 0.01
'MANEUVER_2' 2 0.01
'MANEUVER_3' 2 0.01
$-----MANEUVER_1
[MANEUVER_1]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'
(STEERING)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(THROTTLE)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(BRAKING)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(GEAR)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1

```

```

(CLUTCH)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(MACHINE)
PATH = 'PATH'
SPEED = 'SPEED'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'FALSE'
SPEED_MAP_ABS_TIME = 'FALSE'
CONNECT_PATH = 'FALSE'
(END_CONDITIONS)
{ name }
'DISTANCE_TRAVELLED'
'TIME_SPENT'
$-----MANEUVER_2
[MANEUVER_2]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'
(STEERING)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(THROTTLE)
METHOD = 'OPENLOOP'
SIGNAL = 'THROTTLE_RAMP'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(BRAKING)
METHOD = 'OPENLOOP'
SIGNAL = 'BRAKING_CONSTANT'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(GEAR)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(CLUTCH)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(MACHINE)
PATH = 'PATH'
SPEED = 'SPEED2'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'FALSE'
SPEED_MAP_ABS_TIME = 'FALSE'
CONNECT_PATH = 'FALSE'
(END_CONDITIONS)
{ name }
'DISTANCE_TRAVELLED'
'TIME_SPENT'
$-----MANEUVER_3
[MANEUVER_3]
TASK = 'STANDARD'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'
(STEERING)
METHOD = 'MACHINE'

```

## Appendix B: File Format

```
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(THROTTLE)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(BRAKING)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(GEAR)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(CLUTCH)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(MACHINE)
PATH = 'PATH'
SPEED = 'SPEED3'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'FALSE'
SPEED_MAP_ABS_TIME = 'FALSE'
CONNECT_PATH = 'FALSE'
(END_CONDITIONS)
{ name }
'DISTANCE_TRAVELLED'
'TIME_SPENT'
$-----SPEED
[SPEED]
TYPE = 'MAP'
X_DATA = 'TIME'
Y_DATA = 'LONVEL'
(VALUES)
{ x y }
0 20
0.5 20
2 21.5
$-----SPEED2
[SPEED2]
TYPE = 'MAP'
X_DATA = 'TIME'
Y_DATA = 'LONACC'
(VALUES)
{ x y }
0 1
2 1
$-----SPEED3
[SPEED3]
TYPE = 'MAP'
X_DATA = 'TIME'
Y_DATA = 'LONVEL'
(VALUES)
{ x y }
0 23.5
2 25.5
$-----PATH
[PATH]
TYPE = 'SHAPE'
SHAPE_NAME = 'STRAIGHT'
```

```
$-----DISTANCE_TRAVELLED
[DISTANCE_TRAVELLED]
MEASURE_NAME = 'DISTANCE'
TEST_TYPE = '>>'
REF_VALUE = 10000
$-----TIME_SPENT
[TIME_SPENT]
MEASURE_NAME = 'TIME'
TEST_TYPE = '>>'
REF_VALUE = 10000
$-----VISPEEDGEN
[VISPEEDGEN]
GUESSED_GRIP_LAT = 1
GUESSED_GRIP_LON = 0.7
$-----GRIP_SCALING_MAP
[GRIP_SCALING_MAP]
MODEL = 'MAP'
X_DATA = 'PATH_S'
Y_DATA = 'SCALING'
(VALUES)
{ x y }
0 1
100 1
1000 1
10000 1
```

## 5.11.24 VDF Example 4

This example implements a standstill startup event using a single minimaneuver. for this porpoise the event initial speed is low and the brakes are engaged to stop completely the vehicle. From the standing still condition, the engine rpm are regulated and the clutch is engaged to accelerate the vehicle as defined by the set of clutch STARTUP parameters.

Note: when the maneuver task is set to startup the longitudinal controller neglects the instructions provided by the MACHINE -> SPEED attribute of the maneuver.

```
$-----VIGRADE_HEADER
[VIGRADE_HEADER]
FILE_TYPE = 'VDF'
FILE_VERSION = 9
FILE_FORMAT = 'ASCII'
$-----UNITS
[UNITS]
LENGTH = 'METER'
ANGLE = 'RADIAN'
FORCE = 'NEWTON'
MASS = 'KILOGRAM'
TIME = 'SECOND'
$-----DEBUG_PARAMETERS
[DEBUG_PARAMETERS]
ACTIVATION = 'FALSE'
INITIALIZATION_DUMP = 'TRUE'
$-----GLOBAL_SETTINGS
[GLOBAL_SETTINGS]
MODEL_TYPE = 'CAR_STANDARD'
$-----CONTROLLER_STANDARD
[CONTROLLER_STANDARD]
PREVIEW_TIME = 0.5
MIN_PREVIEW_DISTANCE = 5
LAT_ANT_COMP_TIME = 0
```

## Appendix B: File Format

```

LON_ANT_COMP_TIME = 0
STEERING_PDC_ACTIVE = 'TRUE'
STEERING_YAW_PID_ACTIVE = 'TRUE'
THROTTLE_FEED_FWD_TRQ_ACTIVE = 'TRUE'
THROTTLE_SAT_TGTACCX_ACTIVE = 'TRUE'
$-----STEERING_STANDARD
[STEERING_STANDARD]
MAX_VALUE = 12.566371
MIN_VALUE = -12.566371
$-----THROTTLE_STANDARD
[THROTTLE_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
PID_LONVEL_PROP_GAIN = 0
PID_LONVEL_INTG_GAIN = 0
$-----BRAKING_STANDARD
[BRAKING_STANDARD]
MAX_VALUE = 100
MIN_VALUE = 0
SCALING_FACTOR = 100
$-----GEAR_STANDARD
[GEAR_STANDARD]
MAX_VALUE = 6
MIN_VALUE = 1
(PROPERTIES_MAP)
{ id opt upRPM dwRPM sftT tfallT criseT cfallT triseT dt1 dt2 }
1 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
2 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
3 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
4 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
5 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
6 0 6900 1500 0.5 0.02 0.03 0.25 0.3 0 0.1
$-----CLUTCH_STANDARD
[CLUTCH_STANDARD]
MAX_VALUE = 1
MIN_VALUE = 0
SCALING_FACTOR = 1
MIN_RPM = 100
STARTUP_START_RPM = 3500
STARTUP_START_TIME = 3.5
STARTUP_TARGET_AX = 1
ACTIVATION = 'TRUE'
MODEL = 'STANDARD'
$-----STARTUP
[STARTUP]
STARTING_TIME = 0
INITIAL_SPEED = 5
INITIAL_STEERING = 0
INITIAL_THROTTLE = 0
INITIAL_BRAKING = 1
INITIAL_BRAKING2 = 1
INITIAL_GEAR = 1
INITIAL_CLUTCH = 1
INITIAL_SETUP = 'OFF'
$-----MANEUVERS_LIST
[MANEUVERS_LIST]
{ name abort_time sampling_step }
'BRAKE' 10 0.01
$-----TIRE_BASIC_CONTROLLER
[TIRE_BASIC_CONTROLLER]

```

```
CONTROLLER_DATA = 'TIRE'
BASIC_FILTER_CUTFRQ = 3.1831
MIN_FZ = 5
GEAR_LONSLIP_THRESHOLD = 0.2
GEAR_INHIBITION_DELAY = 0.1
GEAR_MAX_INHIBITION_TIME = 3
ACTIVATION = 'TRUE'
THROTTLE_CONTROL_ACTIVE = 'FALSE'
BRAKING_CONTROL_ACTIVE = 'FALSE'
GEAR_CONTROL_ACTIVE = 'TRUE'
CLUTCH_CONTROL_ACTIVE = 'FALSE'
$-----BRAKE
[BRAKE]
TASK = 'STARTUP'
CONTROLLER_SETTINGS = 'CONTROLLER_STANDARD'
(STEERING)
METHOD = 'OPENLOOP'
SIGNAL = 'CONSTANT_RELATIVE_0'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(THROTTLE)
METHOD = 'MACHINE'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(BRAKING)
METHOD = 'OPENLOOP'
SIGNAL = 'STEP5'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(GEAR)
METHOD = 'MACHINE'
SIGNAL = 'CONSTANT_RELATIVE_0'
SMOOTHING_ACTIVE = 'FALSE'
SMOOTHING_TIME = 0
(CLUTCH)
METHOD = 'MACHINE'
SIGNAL = 'CONSTANT_RELATIVE_0'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(BRAKING2)
METHOD = 'OPENLOOP'
SIGNAL = 'STEP5'
SMOOTHING_ACTIVE = 'TRUE'
SMOOTHING_TIME = 0.1
(TIRE)
CONTROLLER_NAME = 'TIRE_BASIC_CONTROLLER'
(MACHINE)
PATH = 'MANEUVER_1_PATH'
SPEED = 'MANEUVER_1_SPEED'
(OPTIONS)
FOLLOW_TARGET_SPEED = 'TRUE'
SPEED_MAP_ABS_TIME = 'FALSE'
SPEED_MAP_GLOBAL_S = 'FALSE'
CONNECT_PATH = 'TRUE'
END_OF_PATH_CHECK = 'TRUE'
PATH_POSITIONING = 'AUTOMATIC'
(END_CONDITIONS)
{ name }
'CLUTCH_DEMAND'
$-----CLUTCH_DEMAND
[CLUTCH_DEMAND]
```

## Appendix B: File Format

```

MEASURE_NAME = 'CLUTCH_DEMAND'
TEST_TYPE = '==='
REF_VALUE = 0
FILTERING_TIME = 0.5
$-----CONSTANT_RELATIVE_0
[CONSTANT_RELATIVE_0]
TYPE = 'CONSTANT'
VALUE = 0
CALC_MODE = 'RELATIVE'
$-----STEP5
[STEP5]
TYPE = 'STEP5'
X0 = 3
X1 = 3.5
Y0 = 1
Y1 = 0
CALC_MODE = 'ABSOLUTE'
$-----MANEUVER_1_PATH
[MANEUVER_1_PATH]
OVERALL_LENGTH = 20000
STEP_SIZE = 2000
TYPE = 'SHAPE'
SHAPE_NAME = 'STRAIGHT'
$-----MANEUVER_1_SPEED
[MANEUVER_1_SPEED]
TYPE = 'MAINTAIN'
SCALING_FACTOR = 1
CALC_MODE = 'ABSOLUTE'
AUTO_SCALING = 'TRUE'
AUTO_SATURATION = 'TRUE'

```

## 5.12 TIR

The tire model is completely defined via input data ASCII file, based on teim orbit format. The file contains a set of common and specific data blocks. The specific data blocks may vary according to the rtire model type and additional options.

### 5.12.1 Common Blocks

All common blocks are located in the first part of the file.

The mandatory blocks are identified with the following names:

- [MDI\\_HEADER](#)
- [UNITS](#)
- [MODEL](#)
- [DIMENSION](#)
- [VERTICAL](#)
- [LONG\\_SLIP\\_RANGE](#)
- [SLIP\\_ANGLE\\_RANGE](#)
- [INCLINATION\\_ANGLE\\_RANGE](#)
- [VERTICAL\\_FORCE\\_RANGE](#)

#### MDI\_HEADER block

Supports general file information, like file type version and format. Usually all comments about the tire model are stored here.

Attributes for **MDI\_HEADER** block:

**FILE\_TYPE**

file type identification string, must be always 'tir' for VI-Road roads.

**FILE\_VERSION**

file version number, should be 7.0 or higher.

**FILE\_FORMAT**

basic format, only 'ASCII' supported.

**(COMMENTS)**

the comments attribute is followed by the {comment\_string} in the next line, then by any number of comments rows enclosed between ' ' delimiters.

Example of **MDI\_HEADER** block:

```
[MDI_HEADER]
FILE_TYPE = 'tir'
FILE_VERSION = 5.0
FILE_FORMAT = 'ASCII'
```

## UNITS block

Stores data units information.

Attributes for **UNITS** block:

**LENGTH**

length units

**ANGLE**

angle units

**FORCE**

force units

**MASS**

mass units

**TIME**

time units

See ADAMS manuals for supported units keys.

Example of **UNITS** block:

```
[UNITS]
LENGTH = 'meter'
ANGLE = 'radian'
FORCE = 'newton'
MASS = 'kg'
TIME = 'second'
```

## MODEL block

Contains tire model specific information.

Attributes for MODEL block:

### **PROPERTY\_FILE\_FORMAT**

the tire module model definition, must be set as 'USER' to access VI-grade tire models.

### **FUNCTION\_NAME**

define the tire subroutine name. It should be VI-TIRE.

### **MODEL\_TYPE**

the tire model type. Can be 'PAC\_MC'. The PAC\_MC is the Pacejka motorcycle model.

### **USE\_MODE**

the computation mode switch, selecting the tire actual state and slip conditions.

| USE_MODE | STATE     | SLIP CONDITIONS                   | FORCES/MOMENTS         |
|----------|-----------|-----------------------------------|------------------------|
| 0        | STEADY    | NONE (vertical spring)            | Fz only                |
| 1        | STEADY    | PURE LONGITUDINAL                 | Fx, Fz, My             |
| 2        | STEADY    | PURE LATERAL                      | Fy, Fz, Mx, Mz         |
| 3        | STEADY    | PURE LONGITUDINAL AND LATERAL     | Fx, Fy, Fz, Mx, My, Mz |
| 4        | STEADY    | COMBINED LONGITUDINAL AND LATERAL | Fx, Fy, Fz, Mx, My, Mz |
| 11       | TRANSIENT | PURE LONGITUDINAL                 | Fx, Fz, My             |
| 12       | TRANSIENT | PURE LATERAL                      | Fy, Fz, Mx, Mz         |
| 13       | TRANSIENT | PURE LONGITUDINAL AND LATERAL     | Fx, Fy, Fz, Mx, My, Mz |
| 14       | TRANSIENT | COMBINED LONGITUDINAL AND LATERAL | Fx, Fy, Fz, Mx, My, Mz |

### **LONSLIP\_MODE**

the longitudinal slip mode. Supported values are 0 (standard) and 1 (advanced). The advanced slip mode can be used under special conditions, to improve the tire handling during such maneuvers like breaking with ABS system.

### **VXLOW**

the minimal tire longitudinal speed. Under that speed the tire forces are scaled down to zero (at zero speed).

### **LONGV1**

the tire parameters measurements reference speed.

### **TYRESIDE**

it defines the measured (tested) side of the tire. The parameter values can be 'LEFT', 'RIGHT' or 'SYMMETRIC' and it is applied to Pacejka tire models in two ways:

- 1.to mirror tire forces and torques when the tire suspension side is different from the measured side
- 2.to reset all side-dependent tire coefficients when TYRESIDE = 'SYMMETRIC' is detected

**CP2WC\_MODE**

CP2WC computation mode is used to optionally modify the forces and torques transformation from road contact point (CP) to wheel center (WC). Supported values are 0 (default) and 1. When the flag is set to 0 the actual CP (tire section profile considered) is used as reference point for forces and torques application on the ground and the transport arm from CP to WC is the loaded radius. When the flag is set to 1 the theoretical CP (intersection of the wheel disk with the road plane) is used as reference point for forces and torques application on the ground and the transport arm from CP to WC is CPWC (distance between theoretical CP and wheel center).

**CALLING\_APP\_CHECK**

enables the calling application identification, passing this information to the tire (if required). When this optional attribute is set to 'TRUE' the tire may operate with knowledge of the calling application/solver executable (eg. CARREALTIME/BIKEREALTIME/SPEEDGEN). This ability helps switching the actual tire model behaviors to match specific computations requirements.

This attribute has to be active in order to pass the extended MODEL\_SWITCH parameter values to the STI interface of user tire models (see Tire User Interface documentation). The MODEL\_SWITCH == 2 will be passed during static solver calls (SPEEDGEN) only if CALLING\_APP\_CHECK = 'TRUE'.

The default value is 'FALSE'.

Example of **MODEL** block:

```
[MODEL]
PROPERTY_FILE_FORMAT = 'USER'
FUNCTION_NAME = 'vitoools::vi_tire'
MODEL_TYPE = 'PAC_MC'
USE_MODE = 14
LONSLIP_MODE = 1
VXLOW = 1
LONGVL = 16.7
TYRESIDE = 'SYMMETRIC'
CP2WC_MODE = 0
```

**DIMENSION block**

Contains the tire geometry dimensional information.

Attributes for **DIMENSION** block:

**UNLOADED\_RADIUS (R0)**

the tire unloaded (free) radius.

**WIDTH**

the nominal section width of the tire.

**RIM\_RADIUS**

the nominal rim radius.

**RIM\_WIDTH**

the rim width.

---

*Appendix B: File Format*

Example of **DIMENSION** block:

```
[DIMENSION]
UNLOADED_RADIUS = 0.322
WIDTH = 0.18
RIM_RADIUS = 0.216
RIM_WIDTH = 0.135
```

## VERTICAL block

Defines the tire vertical properties (force and effective rolling radius) parameters.

Attributes for **VERTICAL** block:

### **VERTICAL\_STIFFNESS**

the tire linear vertical stiffness.

### **VERTICAL\_DAMPING**

the tire linear vertical damping.

### **BREFF (B<sub>Reff</sub>)**

the low load stiffness effective rolling radius.

### **DREFF (D<sub>Reff</sub>)**

the effective rolling radius peak value.

### **FREFF (F<sub>Reff</sub>)**

high load stiffness effective rolling radius.

### **FNOMIN (F<sub>znom</sub>)**

the tire nominal load.

Example of **VERTICAL** block:

```
[VERTICAL]
VERTICAL_STIFFNESS = 2e+005
VERTICAL_DAMPING = 50
BREFF = 8.4
DREFF = 0.27
FREFF = 0.07
FNOMIN = 1475
```

## LONG\_SLIP\_RANGE block

Defines the tire longitudinal slip limits.

Attributes for **LONG\_SLIP\_RANGE** block:

### **KPUMIN**

minimum longitudinal slip value.

**KPUMAX**

maximum longitudinal slip value.

Example of **LONG\_SLIP\_RANGE** block:

```
[LONG_SLIP_RANGE]
KPUMIN = -1.0
KPUMAX = 1.0
```

**SLIP\_ANGLE\_RANGE block**

Defines the tire lateral slip angle limits.

Attributes for **SLIP\_ANGLE\_RANGE** block:

**ALPMIN ( $\alpha_{\min}$ )**

minimum slip angle value.

**ALPMAX ( $\alpha_{\max}$ )**

maximum slip angle value.

Example of **SLIP\_ANGLE\_RANGE** block:

```
[SLIP_ANGLE_RANGE]
ALPMIN = -1.5708
ALPMAX = 1.5708
```

**INCLINATION\_ANGLE\_RANGE block**

Defines the tire lateral slip angle limits.

Attributes for **INCLINATION\_ANGLE\_RANGE** block:

**CAMMIN**

minimum slip angle value.

**CAMMAX**

maximum slip angle value.

Example of **INCLINATION\_ANGLE\_RANGE** block:

```
[INCLINATION_ANGLE_RANGE]
CAMMIN = -1.1
CAMMAX = 1.1
```

**VERTICAL\_FORCE\_RANGE block**

Defines the tire vertical force limits.

Attributes for **VERTICAL\_FORCE\_RANGE** block:

## Appendix B: File Format

**FZMIN** ( $F_{z\min}$ )

minimum tire vertical load.

**FZMAX** ( $F_{z\max}$ )

maximum tire vertical load.

Example of **VERTICAL\_FORCE\_RANGE** block:

```
[VERTICAL_FORCE_RANGE]
FZMIN = 100.0
FZMAX = 3500.0
```

## 5.12.2 Optional Blocks

The optional blocks may exist or not, depending on the tire model.

- [SCALING\\_COEFFICIENTS](#)
- [LONGITUDINAL\\_COEFFICIENTS](#)
- [LATERAL\\_COEFFICIENTS](#)
- [ALIGNING\\_COEFFICIENTS](#)
- [ROLLING\\_COEFFICIENTS](#)
- [OVERTURNING\\_COEFFICIENTS](#)
- [SECTION\\_PROFILE\\_TABLE](#)
- [SPEED\\_RADIUS\\_LOAD\\_DATA](#)

### SCALING\_COEFFICIENTS block

Defines the Pacejka tire model scaling coefficients.

Attributes for the **SCALING\_COEFFICIENTS** block:

**LFZO** ( $\lambda_{Fz0}$ )

the scaling factor of the nominal load.

**LCX** ( $\lambda_{Cx}$ )

the scaling factor of the  $F_x$  shape factor.

**LMUX** ( $\lambda_{\mu x}$ )

the scaling factor of the  $F_x$  peak friction coefficients.

**LEX** ( $\lambda_{Ex}$ )

the scaling factor of the  $F_x$  curvature factor.

**LKX** ( $\lambda_{Kx}$ )

the scaling factor of the  $F_x$  slip stiffness.

**LVX** ( $\lambda_{Vx}$ )

the scaling factor of the  $F_x$  vertical shift.

**LGAX** ( $\lambda_{\gamma_x}$ )

the scaling factor of the camber for  $F_x$ .

**LCY** ( $\lambda_{c_y}$ )

the scaling factor of the  $F_y$  shape factor.

**LMUY** ( $\lambda_{\mu_y}$ )

the scaling factor of the  $F_y$  peak friction coefficients.

**LEY** ( $\lambda_{e_y}$ )

the scaling factor of the  $F_y$  curvature factor.

**LKY** ( $\lambda_{k_y}$ )

the scaling factor of the  $F_y$  cornering stiffness.

**LCC** ( $\lambda_{cc}$ )

the scaling factor of the camber shape factor.

**LKC** ( $\lambda_{k_c}$ )

the scaling factor of the camber stiffness (K-factor).

**LEC** ( $\lambda_{e_c}$ )

the scaling factor of the camber curvature factor.

**LHY** ( $\lambda_{h_y}$ )

the scaling factor of the  $F_y$  horizontal shift.

**LGAY** ( $\lambda_{\gamma_y}$ )

the scaling factor of the camber force stiffness.

**LTR** ( $\lambda_t$ )

the scaling factor of the peak of pneumatic trail.

**LRES** ( $\lambda_{M_r}$ )

the scaling factor of the peak of residual torque.

**LGAZ** ( $\lambda_{\gamma_z}$ )

the scaling factor of the camber torque stiffness.

## Appendix B: File Format

**LXAL** ( $\lambda_{x\alpha}$ )

the scaling factor of the lateral slip influence on  $F_x$ .

**LYKA** ( $\lambda_{y_k}$ )

the scaling factor of the longitudinal slip influence on  $F_y$ .

**LVYKA** ( $\lambda_{v_y k}$ )

the scaling factor of the longitudinal slip induced  $F_y$ .

**LS** ( $\lambda_s$ )

the scaling factor of the moment of arm of  $F_x$ .

**LSGKP** ( $\lambda_{\delta_k}$ )

the scaling factor of the  $F_x$  relaxation length.

**LSGAL** ( $\lambda_{\delta_\alpha}$ )

the scaling factor of the  $F_y$  relaxation length.

**LGYR** ( $\lambda_{gyr}$ )

the scaling factor of the gyroscopic torque.

**LMX** ( $\lambda_{M_x}$ )

the scaling factor of the overturning moment.

**LVMX** ( $\lambda_{v_{Mx}}$ )

the scaling factor of the  $M_x$  vertical shift.

**LMY** ( $\lambda_{M_y}$ )

the scaling factor of the rolling resistance torque.

Example of **SCALING\_COEFFICIENTS** block:

```
[SCALING_COEFFICIENTS]
LFZO = 1
LCX = 1
LMUX = 1
LEX = 1
LKX = 1
LVX = 1
LGAX = 1
LCY = 1
LMUY = 1
```

```

LEY = 1
LKY = 1
LCC = 1
LKC = 1
LEC = 1
LHY = 1
LGAY = .8
LTR = 1
LRES = 1
LGAZ = 1
LXAL = 1
LYKA = 1
LVYKA = 1
LS = 1
LSGKP = 1
LSGAL = 1
LGYR = 1
LMX = 1
LVMX = 1
LMY = 1

```

## LONGITUDINAL\_COEFFICIENTS block

Defines the Pacejka tire model longitudinal coefficients.

Attributes for the **LONGITUDINAL\_COEFFICIENTS** block:

### **P<sub>Cx1</sub>** (**p<sub>Cx1</sub>**)

the shape factor for the longitudinal force.

### **P<sub>Dx1</sub>** (**p<sub>Dx1</sub>**)

the longitudinal friction  $\mu_x$  at nominal vertical load ( $F_{Z_{nom}}$ ).

### **P<sub>Dx2</sub>** (**p<sub>Dx2</sub>**)

the variation of the longitudinal friction  $\mu_x$  with load.

### **P<sub>Dx3</sub>** (**p<sub>Dx3</sub>**)

the variation of the longitudinal friction  $\mu_x$  with camber.

### **P<sub>Ex1</sub>** (**p<sub>Ex1</sub>**)

the longitudinal curvature  $E_{fx}$  at nominal vertical load ( $F_{Z_{nom}}$ ).

### **P<sub>Ex2</sub>** (**p<sub>Ex2</sub>**)

the variation of the longitudinal curvature  $E_{fx}$  with load.

### **P<sub>Ex3</sub>** (**p<sub>Ex3</sub>**)

the variation the longitudinal curvature  $E_{fx}$  with load squared.

*Appendix B: File Format***PEX4 (p<sub>Ex4</sub>)**

the factor in longitudinal curvature E<sub>fx</sub> while driving.

**PKX1 (p<sub>Kx1</sub>)**

the longitudinal slip stiffness K<sub>fx</sub>/F<sub>z</sub> at nominal vertical load (F<sub>Znom</sub>).

**PKX2 (p<sub>Kx2</sub>)**

the variation of the longitudinal slip stiffness K<sub>fx</sub>/F<sub>z</sub> with load.

**PKX3 (p<sub>Kx3</sub>)**

the exponent in the longitudinal slip stiffness K<sub>fx</sub>/F<sub>z</sub> with load.

**PVX1 (p<sub>Vx1</sub>)**

the vertical shift S<sub>vx</sub>/F<sub>z</sub> at nominal vertical load (F<sub>Znom</sub>).

**PVX2 (p<sub>Vx2</sub>)**

the variation of the vertical shift S<sub>vx</sub>/F<sub>z</sub> with load.

**RBX1 (r<sub>Bx1</sub>)**

the slope factor for combined slip F<sub>x</sub> reduction.

**RBX2 (r<sub>Bx2</sub>)**

the variation of the slope factor for combined slip F<sub>x</sub> reduction with longitudinal slip.

**RBX3 (r<sub>Bx3</sub>)**

the influence of camber on stiffness for combined slip F<sub>x</sub>.

**RCX1 (r<sub>Cx1</sub>)**

the shape factor for combined slip F<sub>x</sub> reduction.

**REX1 (r<sub>Ex1</sub>)**

the curvature factor of combined slip F<sub>x</sub>.

**REX2 (r<sub>Ex2</sub>)**

the variation of the curvature factor of combined slip F<sub>x</sub> with load.

**RHX1 (r<sub>Hx1</sub>)**

the shift factor for combined slip F<sub>x</sub> reduction.

**PTX1 (p<sub>tx1</sub>)**

the longitudinal relaxation length at nominal vertical load ( $F_{Z\text{nom}}$ ).

**PTX2 (p<sub>tx2</sub>)**

variation of the longitudinal relaxation length with load.

**PTX3 (p<sub>tx3</sub>)**

variation of the longitudinal relaxation length with exponent of load.

Example of **LONGITUDINAL\_COEFFICIENTS** block:

```
[LONGITUDINAL_COEFFICIENTS]
PCX1 = 1.7655
PDX1 = 1.2839
PDX2 = -0.0078226
PDX3 = 0
PEX1 = 0.4743
PEX2 = 9.3873e-005
PEX3 = 0.066154
PEX4 = 0.00011999
PKX1 = 25.383
PKX2 = 1.0978
PKX3 = 0.19775
PVX1 = 2.1675e-005
PVX2 = 4.7461e-005
RBX1 = 12.084
RBX2 = -8.3959
RBX3 = 2.1971e-009
RCX1 = 1.0648
REX1 = 0.0028793
REX2 = -0.00037777
RHX1 = 0
PTX1 = 0.83
PTX2 = 0.42
PTX3 = 0.21
```

**LATERAL\_COEFFICIENTS block**

Defines the Pacejka tire model lateral coefficients.

Attributes for the **LATERAL\_COEFFICIENTS** block:

**PCY1 (p<sub>cy1</sub>)**

the shape factor for the lateral forces.

**PCY2 (p<sub>cy2</sub>)**

the shape factor for the camber force.

**PDY2 (p<sub>dy1</sub>)**

the lateral friction  $\mu_y$  at nominal vertical load ( $F_{Z\text{nom}}$ ).

*Appendix B: File Format***PDY2 (p<sub>Dy2</sub>)**

the variation of the lateral friction  $\mu_y$  with load.

**PDY3 (p<sub>Dy3</sub>)**

the variation of the lateral friction  $\mu_y$  with camber squared.

**PEY1 (p<sub>Ey1</sub>)**

the lateral curvature ( $E_{fy}$ ) at nominal vertical load ( $F_{Znom}$ ).

**PEY2 (p<sub>Ey2</sub>)**

the variation of the lateral curvature ( $E_{fy}$ ) with camber squared.

**PEY3 (p<sub>Ey3</sub>)**

the asymmetric variation of curvature ( $E_{fy}$ ) at nominal vertical load ( $F_{Znom}$ ).

**PEY4 (p<sub>Ey4</sub>)**

the asymmetric variation of curvature ( $E_{fy}$ ) with camber.

**PEY5 (p<sub>Ey5</sub>)**

the camber curvature ( $E_{fc}$ ).

**PKY1 (p<sub>Ky1</sub>)**

the maximum value of stiffness ( $K_{fy}/F_{Znom}$ ).

**PKY2 (p<sub>Ky2</sub>)**

the curvature of stiffness ( $K_{fy}$ ).

**PKY3 (p<sub>Ky3</sub>)**

the peak stiffness factor.

**PKY4 (p<sub>Ky4</sub>)**

the peak stiffness variation with camber squared.

**PKY5 (p<sub>Ky5</sub>)**

the variation of lateral stiffness with camber squared.

**PKY6 (p<sub>Ky6</sub>)**

the camber stiffness factor ( $K_{fc}$ )

**PKY7 (p<sub>Ky6</sub>)**

the variation of camber stiffness factor ( $K_{fc}$ ) with load.

**PHY1 (p<sub>Hy1</sub>)**

the horizontal shift ( $S_{hy}$ ) at nominal vertical load ( $F_{Znom}$ ).

**RBY1 (r<sub>By1</sub>)**

the slope factor for combined slip  $F_y$  reduction.

**RBY2 (r<sub>By2</sub>)**

the variation of the slope factor for combined slip  $F_y$  reduction with lateral slip.

**RBY3 (r<sub>By3</sub>)**

the shift of lateral slip for combined  $F_y$  slope reduction.

**RBY4 (r<sub>By4</sub>)**

the influence of camber on stiffness for combined slip  $F_y$ .

**RCY1 (r<sub>Cy1</sub>)**

the shape factor for combined slip  $F_y$  reduction.

**REY1 (r<sub>Ey1</sub>)**

the curvature factor of combined slip  $F_y$ .

**REY2 (r<sub>Ey2</sub>)**

the variation of the curvature factor of combined slip  $F_y$  with load.

**RHY1 (r<sub>Hy1</sub>)**

the shift factor for combined slip  $F_y$  reduction.

**RHY2 (r<sub>Hy2</sub>)**

the variation of the shift factor for combined slip  $F_y$  reduction with load.

**RVY1 (r<sub>Vy1</sub>)**

the longitudinal slip induced side force  $S_{vyk} / \mu_y * F_z$  at nominal vertical load ( $F_{Znom}$ ).

**RVY2 (r<sub>Vy2</sub>)**

the variation of  $S_{vyk} / \mu_y * F_z$  with load.

## Appendix B: File Format

**RVY3 (r<sub>vy3</sub>)**

the variation of  $S_{vyk} / \mu_y * F_z$  with camber.

**RVY4 (r<sub>vy4</sub>)**

the variation of  $S_{vyk} / \mu_y * F_z$  with lateral slip.

**RVY5 (r<sub>vy5</sub>)**

the variation of  $S_{vyk} / \mu_y * F_z$  with longitudinal slip.

**RVY6 (r<sub>vy6</sub>)**

the variation of  $S_{vyk} / \mu_y * F_z$  with longitudinal slip arctangent ( $\text{atan}(k)$ ).

**PTY1 (p<sub>ty1</sub>)**

the peak value of the lateral relaxation length.

**PTY2 (p<sub>ty2</sub>)**

the shape factor for the lateral relaxation length.

**PTY3 (p<sub>ty3</sub>)**

value of  $F_z/F_{Znom}$  at peak relaxation length value.

Example of **LATERAL\_COEFFICIENTS** block:

```
[LATERAL_COEFFICIENTS]
PCY1 = 1.1086
PCY2 = 0.66464
PDY1 = 1.3898
PDY2 = -0.0044718
PDY3 = 0.21428
PEY1 = -0.80276
PEY2 = 0.89416
PEY3 = 0
PEY4 = 0
PEY5 = -2.8159
PKY1 = -19.747
PKY2 = 1.3756
PKY3 = 1.3528
PKY4 = -1.2481
PKY5 = 0.3743
PKY6 = -0.91343
PKY7 = 0.2907
PHY1 = 0
RBY1 = 10.694
RBY2 = 8.9413
RBY3 = 0
RBY4 = -1.8256e-010
RCY1 = 1.0521
REY1 = -0.0027402
```

|      |                |
|------|----------------|
| REY2 | = -0.0094269   |
| RHY1 | = -7.864e-005  |
| RHY2 | = -6.9003e-006 |
| RVY1 | = 0            |
| RVY2 | = 0            |
| RVY3 | = -0.00033208  |
| RVY4 | = -4.7907e+015 |
| RVY5 | = 1.9          |
| RVY6 | = -30.082      |
| PTY1 | = 0.75         |
| PTY2 | = 1            |
| PTY3 | = 0.6          |

## ALIGNING\_COEFFICIENTS block

Defines the Pacejka tire model aligning coefficients.

Attributes for the **ALIGNING\_COEFFICIENTS** block:

### QBZ1 ( $q_{Bz1}$ )

the trail slope factor for the trail ( $B_{pt}$ ) at nominal vertical load ( $F_{Znom}$ ).

### QBZ2 ( $q_{Bz2}$ )

the variation of the trail slope ( $B_{pt}$ ) with load.

### QBZ3 ( $q_{Bz3}$ )

the variation of the trail slope ( $B_{pt}$ ) with load squared.

### QBZ4 ( $q_{Bz4}$ )

the variation of the trail slope ( $B_{pt}$ ) with camber.

### QBZ5 ( $q_{Bz5}$ )

the variation of the trail slope ( $B_{pt}$ ) with absolute camber.

### QBZ9 ( $q_{Bz9}$ )

the slope factor ( $B_r$ ) of residual torque ( $M_{zr}$ ).

### QCZ1 ( $q_{Cz1}$ )

the shape factor ( $C_{pt}$ ) for the pneumatic trail.

### QDZ1 ( $q_{Dz1}$ )

the peak trail ( $D_{pt} = D_{pt} * (F_z/F_{Znom} * R0)$ ).

### QDZ2 ( $q_{Dz2}$ )

the variation of peak trail ( $D_{pt}$ ) with load.

**QDZ3 (q<sub>Dz3</sub>)**

the variation of peak trail ( $D_{pt}$ ) with camber.

**QDZ4 (q<sub>Dz4</sub>)**

the variation of peak trail ( $D_{pt}$ ) with camber squared.

**QDZ6 (q<sub>Dz6</sub>)**

the peak residual torque ( $D_{mr} = D_{mr} / (F_z * R0)$ ).

**QDZ7 (q<sub>Dz7</sub>)**

the variation of peak factor ( $D_{mr}$ ) with load.

**QDZ8 (q<sub>Dz8</sub>)**

the variation of peak factor ( $D_{mr}$ ) with camber.

**QDZ9 (q<sub>Dz9</sub>)**

the variation of peak factor ( $D_{mr}$ ) with camber and load.

**QDZ10 (q<sub>Dz10</sub>)**

the variation of peak factor ( $D_{mr}$ ) with camber squared.

**QDZ11 (q<sub>Dz11</sub>)**

the variation of peak factor ( $D_{mr}$ ) with camber squared and load.

**QEZ1 (q<sub>Ez1</sub>)**

the trail curvature ( $E_{pt}$ ) at nominal vertical load ( $F_{Znom}$ ).

**QEZ2 (q<sub>Ez1</sub>)**

the variation of trail curvature ( $E_{pt}$ ) with load.

**QEZ3 (q<sub>Ez3</sub>)**

the variation of trail curvature ( $E_{pt}$ ) with load squared.

**QEZ4 (q<sub>Ez4</sub>)**

the variation of trail curvature ( $E_{pt}$ ) with sign of  $\alpha_t$ .

**QEZ5 (q<sub>Ez5</sub>)**

the variation of trail curvature ( $E_{pt}$ ) with camber and sign of  $\alpha_t$ .

**QHZ1 (q<sub>Hz1</sub>)**

the trail horizontal shift ( $S_{ht}$ ) at nominal vertical load ( $F_{Znom}$ ).

**QHZ2 (q<sub>Hz2</sub>)**

the variation of trail horizontal shift ( $S_{ht}$ ) with load.

**QHZ3 (q<sub>Hz3</sub>)**

the variation of trail horizontal shift ( $S_{ht}$ ) with camber.

**QHZ4 (q<sub>Hz4</sub>)**

the variation of trail horizontal shift ( $S_{ht}$ ) with camber and load.

**SSZ1 (s<sub>sz1</sub>)**

the nominal value of s/R0 (effect of  $F_x$  on  $M_z$ ).

**SSZ2 (s<sub>sz2</sub>)**

the variation of distance s/R0 with  $F_y/F_{Znom}$ .

**SSZ3 (s<sub>sz3</sub>)**

the variation of distance s/R0 with camber.

**SSZ4 (s<sub>sz4</sub>)**

the variation of distance s/R0 with load and camber.

**QTZ1 (q<sub>Tz1</sub>)**

the gyroscopic torque constant.

**MBELT (M<sub>belt</sub>)**

the tire belt mass (kg).

Example of **ALIGNING\_COEFFICIENTS** block:

```
[ALIGNING_COEFFICIENTS]
QBZ1 = 9.246
QBZ2 = -1.4442
QBZ3 = -1.8323
QBZ4 = 0
QBZ5 = 0.15703
QBZ9 = 8.3146
QCZ1 = 1.2813
QDZ1 = 0.063288
QDZ2 = -0.015642
```

## Appendix B: File Format

```

QDZ3 = -0.060347
QDZ4 = -0.45022
QDZ6 = 0
QDZ7 = 0
QDZ8 = -0.08525
QDZ9 = -0.081035
QDZ10 = 0.030766
QDZ11 = 0.074309
QEZ1 = -3.261
QEZ2 = 0.63036
QEZ3 = 0
QEZ4 = 0
QEZ5 = 0
QHZ1 = 0
QHZ2 = 0
QHZ3 = 0
QHZ4 = 0
SSZ1 = 0
SSZ2 = 0.0033657
SSZ3 = 0.16833
SSZ4 = 0.017856
QTZ1 = 0
MBELT = 0

```

**ROLLING\_COEFFICIENTS block**

Defines the Pacejka tire model rolling resistance coefficients.

Attributes for the **ROLLING\_COEFFICIENTS** block:

**QSY1 (q<sub>sy1</sub>)**

the rolling resistance torque coefficient.

**QSY2 (q<sub>sy2</sub>)**

the rolling resistance torque coefficient F<sub>x</sub> dependency.

**QSY3 (q<sub>sy3</sub>)**

the rolling resistance torque coefficient speed dependency.

**QSY4 (q<sub>sy4</sub>)**

the rolling resistance torque coefficient speed<sup>4</sup> dependency.

Example of **ROLLING\_COEFFICIENTS** block:

```
[ROLLING_COEFFICIENTS]
QSY1 = 0.01
QSY2 = 0
QSY3 = 0
QSY4 = 0
```

## OVERTURNING\_COEFFICIENTS block

Defines the Pacejka tire model overturning torque coefficients.

Attributes for the OVERTURNING\_COEFFICIENTS block:

**QSX1 (q<sub>Sx1</sub>)**

the overturning moment due to lateral force.

**QSX2 (q<sub>Sx2</sub>)**

the overturning moment due to camber angle.

**QSX3 (q<sub>Sx3</sub>)**

the overturning moment due to F<sub>y</sub>.

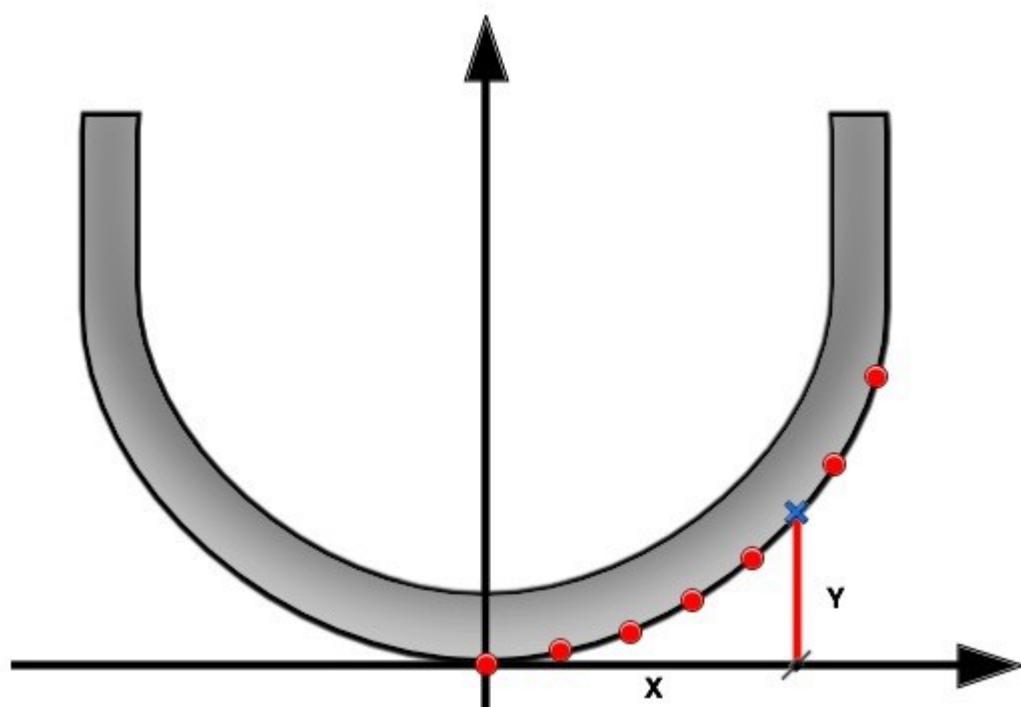
Example of OVERTURNING\_COEFFICIENTS block:

```
[OVERTURNING_COEFFICIENTS]
QSX1 = 0
QSX2 = 0.16056
QSX3 = 0.095298
```

## SECTION\_PROFILE\_TABLE block

Defines the tire section profile.

The table consists of a series of equally spaced points coordinates { X, Y }, representing the half of the symmetric tire section profile.



Example of **SECTION\_PROFILE\_TABLE** block:

```
[SECTION_PROFILE_TABLE]
{ x y }
0.000000 0.0000000
0.00600 0.0001877
0.01200 0.0007561
0.01800 0.0017216
0.02400 0.0031114
0.03000 0.0049639
0.03600 0.0073282
0.04200 0.0102646
0.04800 0.0138449
0.05400 0.0181517
0.06000 0.0232793
0.06600 0.0293337
0.07200 0.0364323
0.07800 0.0447047
0.08400 0.0542923
```

## SPEED\_RADIUS\_LOAD\_DATA block

Defines the tire vertical load 3D map.

The vertical load is computed as function of the loaded radius and the longitudinal speed of the tire. If this data block exists in the file, then the map is used (there is no need of specific activation flag).

Example of **SPEED\_RADIUS\_LOAD\_DATA** block:

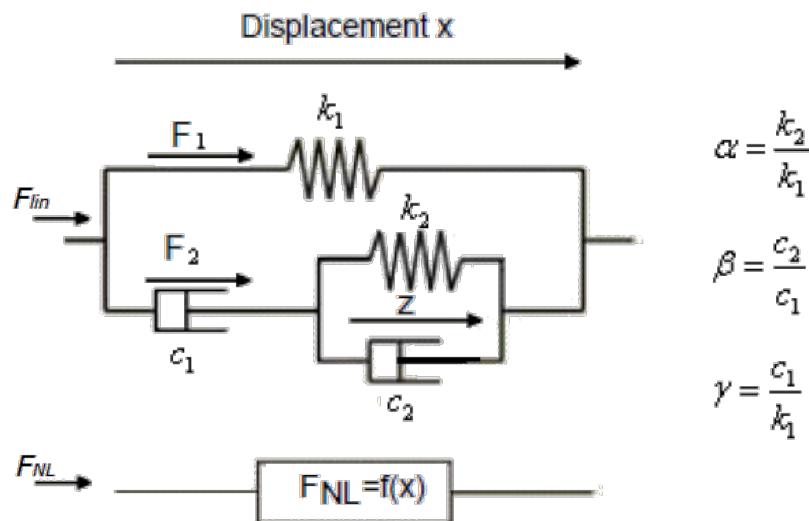
```
[SPEED_RADIUS_LOAD_DATA]
(Z_DATA)
{speed}
 0.0
 33.0
 66.0
(XY_DATA)
{tire radius load }
 0.20 105000 110000 115000
 0.25 75000 80000 85000
 0.3 14000 14500 15000
 0.31 10000 10500 11000
 0.32 7000 7500 8000
 0.33 4000 4500 5000
 0.335 2500 2800 3000
 0.337 2100 2300 2500
 0.339 1400 1500 1600
 0.341 1000 1050 1100
 0.343 400 430 450
 0.345 0 0 0
```

## 5.13 UBF File Format

- [Elastomer Model](#)
- [Hydromount Model](#)
- [Property File](#)

### 5.13.1 Elastomer Model

The elastomer model provides a force that is the sum of two components as illustrated in the figure below.



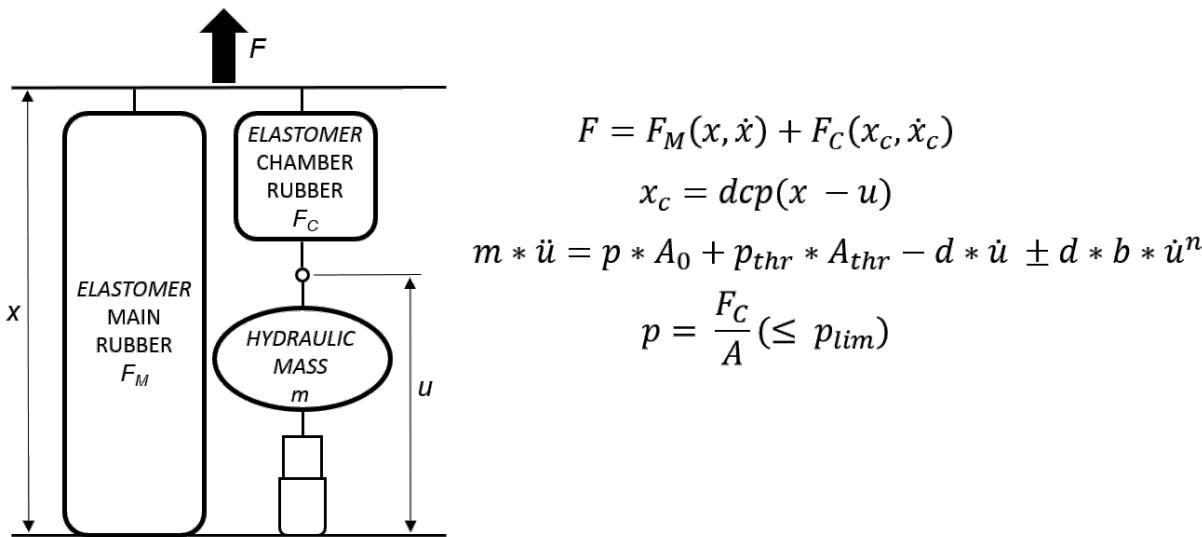
- $F_{lin}$ : linear (frequency dependent) force component, where  $k_1$  is the main static stiffness that can be also a nonlinear curve and the other stiffness and damping terms are a consequence of  $\alpha$ ,  $\beta$  and  $\gamma$ .
- $F_{NL}$ : nonlinear (amplitude dependent) force component which is given by the following formula (friction force model)

$$F_{NL} = \int_{t(x_s)}^t \left[ \left( |x - x_s| \cdot \frac{RDL \cdot \rho}{\rho + 1} \right) \left( \frac{F_{max1} \pm F_{last}}{F_{max2}} \right) \cdot \dot{x} \right] dt$$

where  $x$  is the current position,  $x_s$  is the position at the previous reversal point (position where the velocity sign has changed),  $F_{last}$  is the force value at last integration step and  $F_{max1}$  and  $F_{max2}$  determine the maximum friction force. Finally  $RDL$  and  $\rho$  are the model internal parameters for logarithmic friction force.

### 5.13.2 Hydromount Model

The hydromount model involves two elastomer models (main rubber and chamber rubber) and a damped fluid mass which are arranged as in the scheme below.



Looking at the equations beside the scheme the main features can be briefly summarized:

- The total force is the sum of the elastomer forces  $F_M$  and  $F_C$
- The input displacement  $x_c$  for the chamber rubber is affected by a decoupling function  $dcp$  which represents basically a clearance effect ( $F_C = 0$  under the given displacement threshold)
- The displacement  $u$  of the fluid mass  $m$  is governed by pressure and damping forces. The parameters and input terms in the fluid equation are:
  - $p$ : fluid pressure  $F_C/A$ , with  $A$  denoting the area of the chamber cross section
  - $A_0$ : area of the channel cross section
  - $p_{thr}$ : pressure exceeding the threshold value of the bypass valve next to the channel
  - $A_{thr}$ : area of the bypass valve cross section
  - $d$ : linear damping constant
  - $b$ : nonlinear damping constant
  - $n$ : nonlinear damping exponent
  - $p_{lim}$ : pressure limit (threshold of the pressure-relief valve)

### 5.13.3 Property File

The GENERAL block contains the SHAPE parameter that can have the following values regarding the directions coupling: 'none' (no coupling), 'cylindrical' (X and Y coupled), 'spherical' (X-Y-Z coupled).

The rest of the file has a structure divided in direction blocks (FX, FY, FZ, TX, TY, TZ) and model sub-blocks. Several sub-block of different types can be grouped inside each parent block.

The following are the basic rules which are applied to the file.

- "00" suffix indicates the main sub-block (i.e. the sub-block to be read first)
- "01", "02", ..., "99" are reserved for secondary sub-blocks (they are read after the associated main block)
- each model (ELASTOMER\_FREQ\_AMPL, ELASTOMER\_FREQ, HYDROBUSHING\_FREQ\_AMPL, HYDROBUSHING\_FREQ, HYDROMOUNT\_FREQ\_AMPL, HYDROMOUNT\_FREQ) has its own set of default values for the parameters that are missing in the file
- some model can override the parameters that are in the property file because it must have a predefined behavior

- (CURVE) sub-block is intended for non linear static stiffness curve (only one for each direction)
- (CHAMBER\_CURVE\_IMPACT\_PRELOAD) sub-block is an optional additional curve (one for each translation) that represents the nonlinear variation of the chamber stiffness with respect to the input displacement. It is defined as  $(F_{\text{corrected}} - F_{\text{original}})/F_{\text{original}}$

Property file parameters for each model are listed below with the indication of the units and a short description. The tables show also which default values are applied. The symbol (o) is used in the cases where the parameter is overwritten by the showed value, so the property file value is discarded (predetermined behavior). You can notice for instance that FLUID\_PRESSURE\_LIMIT\_BYPASS\_AREA is forced to be zero in the HYDROMOUNT\_FREQ\_AMPL model because the basic difference with respect to the HYDROBUSHING\_FREQ\_AMPL model is indeed the absence of pressure limitation.

### ELASTOMER parameters

- USE\_CURVE ('yes'/'no'): option to activate the nonlinear stiffness curve
- STIFFNESS\_STATIC (*stiffness* ): main static stiffness
- ALPHA (-): spring ratio k2/k1
- BETA (-): damper ratio c2/c1
- GAMMA (*time* ): damper-spring ratio c1/k1
- LAMBDA (*degree* ): loss angle
- RHO\_RUBBER (*1/displacement* ): nonlinear force term
- RDL\_RUBBER (*displacement* ): nonlinear force term
- FRICTION\_MAX\_1 (*displacement* ): nonlinear force limitation
- FRICTION\_MAX\_2 (*displacement* ): nonlinear force limitation
- FRICTION\_SIGNUM\_GRADIENT (-): velocity zero-crossing factor
- SCALE\_FREQ (-): linear force gain
- SCALE\_AMPL (-): nonlinear force gain
- SCALE\_STIFFNESS\_CURVE (-): main static stiffness gain

|                          | ELASTOMER_FREQ_AMPL | ELASTOMER_FREQ |
|--------------------------|---------------------|----------------|
| USE_CURVE                | 'no'                | 'no'           |
| STIFFNESS_STATIC         | -                   | -              |
| ALPHA                    | -                   | -              |
| BETA                     | -                   | -              |
| GAMMA                    | -                   | -              |
| LAMBDA                   | -                   | -              |
| RHO_RUBBER               | -                   | 3.8E-004 1/m   |
| RDL_RUBBER               | -                   | 600.0 m        |
| FRICTION_MAX_1           | 10*RDL_RUBBER       | 10*RDL_RUBBER  |
| FRICTION_MAX_2           | FRICTION_MAX_1      | FRICTION_MAX_1 |
| FRICTION_SIGNUM_GRADIENT | 1.0E9               | 0 (o)          |
| SCALE_FREQ               | 1.0                 | 1.0            |
| SCALE_AMPL               | 1.0                 | 0 (o)          |
| SCALE_STIFFNESS_CURVE    | 1.0                 | 1.0            |

## Elastomer default values

### HYDROBUSHING/HYDROMOUNT parameters

- MAIN\_RUBBER\_USE\_CURVE ('yes'/'no'): option to activate the nonlinear main rubber stiffness curve
- MAIN\_RUBBER\_STIFFNESS\_STATIC (*stiffness* ): main rubber static stiffness
- ALPHA (-): spring ratio k2/k1
- BETA (-): damper ratio c2/c1
- GAMMA (*time* ): damper-spring ratio c1/k1
- LAMBDA (*degree* ): loss angle
- MAIN\_RUBBER\_rho (1/*displacement* ): nonlinear force term
- MAIN\_RUBBER\_RDL (*displacement* ): nonlinear force term
- MAIN\_RUBBER\_FRICTION\_MAX\_1 (*displacement* ): nonlinear force limitation
- MAIN\_RUBBER\_FRICTION\_MAX\_2 (*displacement* ): nonlinear force limitation
- MAIN\_RUBBER\_FRICTION\_SIGNUM\_GRADIENT (-): velocity zero-crossing factor
- CHAMBER\_RUBBER\_STIFFNESS\_STATIC (*stiffness* ): main chamber static stiffness
- CHAMBER\_RUBBER\_LAMBDA (*degree* ): loss angle
- CHAMBER\_RUBBER\_rho (1/*displacement* ): nonlinear force term
- CHAMBER\_RUBBER\_RDL (*displacement* ): nonlinear force term
- CHAMBER\_RUBBER\_FRICTION\_MAX\_1 (*displacement* ): nonlinear force limitation
- CHAMBER\_RUBBER\_FRICTION\_MAX\_2 (*displacement* ): nonlinear force limitation
- CHAMBER\_RUBBER\_FRICTION\_SIGNUM\_GRADIENT (-): velocity zero-crossing factor
- FLUID\_CHAMBER\_AREA (*area* ): piston area
- FLUID\_CHANNEL\_AREA (*area* ): channel cross section
- FLUID\_MASS (*mass* ): fluid track mass
- FLUID\_DAMPING\_LINEAR (*damping* ): channel linear damping
- FLUID\_DAMPING\_EXPONENT (-): channel nonlinear damping exponent
- FLUID\_DAMPING\_NONLINEAR ((1/*velocity*)<sup>(FLUID\_DAMPING\_EXPONENT-1)</sup> ): channel nonlinear damping
- FLUID\_DAMPING\_EXPONENT\_GRADIENT (-): fluid velocity zero-crossing factor
- DECOUPLING\_GRADIENT (-): decoupling membrane gain
- DECOUPLING\_CLEARANCE (*displacement* ): decoupling membrane clearance
- FLUID\_PRESSURE\_LIMIT (*pressure* ): decompression limit
- FLUID\_PRESSURE\_LIMIT\_SIGNUM\_GRADIENT (-): pressure zero-crossing factor
- CHAMBER\_PRELOAD\_OPERATING\_POINT (*displacement*): offset at operating point when evaluating CHAMBER\_CURVE\_IMPACT\_PRELOAD
- CHAMBER\_PRELOAD\_SENSITIVITY\_SIGNUM\_GRADIENT (1/*displacement* ): impact preload scaling factor rate. It is used only if CHAMBER\_CURVE\_IMPACT\_PRELOAD is not present to construct an alternative linear curve
- MAIN\_RUBBER\_SCALE\_FREQ (-): linear force gain

- MAIN\_RUBBER\_SCALE\_AMPL (-): nonlinear force gain
- SCALE\_MAIN\_RUBBER\_CURVE (-): main rubber stiffness gain
- CHAMBER\_RUBBER\_SCALE\_FREQ (-): linear force gain
- CHAMBER\_RUBBER\_SCALE\_AMPL (-): nonlinear force gain
- SCALE\_CHAMBER\_STIFFNESS (-): main chamber stiffness gain
- FLUID\_SCALE\_LINEAR\_DAMPING (-): fluid linear damping gain
- FLUID\_SCALE\_NONLINEAR\_DAMPING (-): fluid nonlinear damping gain
- FLUID\_SCALE\_TUNED\_MASS\_DAMPER (-): fluid force gain
- SCALE\_FREQ\_SHIFT (-): fluid resonance scaling factor
- FLUID\_PRESSURE\_LIMIT\_BYPASS\_AREA (*area*): bypass cross section
- FLUID\_PRESSURE\_LIMIT\_BYPASS\_THRESHOLD (*pressure*): bypass pressure limit

|                                         | HYDROBUSHING_FREQ_AMPL        | HYDROBUSHING_FREQ             |
|-----------------------------------------|-------------------------------|-------------------------------|
| MAIN_RUBBER_USE_CURVE                   | 'no'                          | 'no'                          |
| MAIN_RUBBER_STIFFNESS_STATIC            | -                             | -                             |
| ALPHA                                   | -                             | -                             |
| BETA                                    | -                             | -                             |
| GAMMA                                   | -                             | -                             |
| LAMBDA                                  | -                             | -                             |
| MAIN_RUBBER_rho                         | -                             | 3.8E-004 1/m                  |
| MAIN_RUBBER_RDL                         | -                             | 600.0 m                       |
| MAIN_RUBBER_FRICTION_MAX_1              | 10*MAIN_RUBBER_RDL            | 10*MAIN_RUBBER_RDL            |
| MAIN_RUBBER_FRICTION_MAX_2              | MAIN_RUBBER_FRICTION_MAX_1    | MAIN_RUBBER_FRICTION_MAX_1    |
| MAIN_RUBBER_FRICTION_SIGNUM_GRADIENT    | 1.0E9                         | 0.0 (o)                       |
| CHAMBER_RUBBER_STIFFNESS_STATIC         | -                             | -                             |
| CHAMBER_RUBBER_LAMBDA                   | -                             | -                             |
| CHAMBER_RUBBER_rho                      | -                             | 3.8E-004 1/m                  |
| CHAMBER_RUBBER_RDL                      | -                             | 600.0 m                       |
| CHAMBER_RUBBER_FRICTION_MAX_1           | 10*CHAMBER_RUBBER_RDL         | 10*CHAMBER_RUBBER_RDL         |
| CHAMBER_RUBBER_FRICTION_MAX_2           | CHAMBER_RUBBER_FRICTION_MAX_1 | CHAMBER_RUBBER_FRICTION_MAX_1 |
| CHAMBER_RUBBER_FRICTION_SIGNUM_GRADIENT | 1.0E9                         | 0.0 (o)                       |
| FLUID_CHAMBER_AREA                      | -                             | -                             |
| FLUID_CHANNEL_AREA                      | -                             | -                             |
| FLUID_MASS                              | -                             | -                             |
| FLUID_DAMPING_LINEAR                    | -                             | 0.0                           |
| FLUID_DAMPING_EXPONENT                  | 2.0                           | 1.0 (o)                       |
| FLUID_DAMPING_NONLINEAR                 | -                             | -                             |
| FLUID_DAMPING_EXPONENT_GRADIENT         | 1.0E9                         | 0.0 (o)                       |

## Appendix B: File Format

|                                             |             |                |
|---------------------------------------------|-------------|----------------|
| DECOUPLING_GRADIENT                         | 0.0         | <b>0.0 (o)</b> |
| DECOUPLING_CLEARANCE                        | 1.0E-6 m    | 1.0E-6 m       |
| FLUID_PRESSURE_LIMIT                        | 1.0E6 N/m^2 | 1.0E6 N/m^2    |
| FLUID_PRESSURE_LIMIT_SIGN<br>UM_GRADIENT    | 1.0E9       | <b>0.0 (o)</b> |
| CHAMBER_PRELOAD_OPERATING_POINT             | -           | -              |
| CHAMBER_PRELOAD_SENSITIVITY_SIGNUM_GRADIENT | 0.0         | <b>0.0 (o)</b> |
| MAIN_RUBBER_SCALE_FREQ                      | 1.0         | 1.0            |
| MAIN_RUBBER_SCALE_AMPL                      | 1.0         | <b>0.0 (o)</b> |
| SCALE_MAIN_RUBBER_CURVE                     | 1.0         | 1.0            |
| CHAMBER_RUBBER_SCALE_EQ                     | 1.0         | 1.0            |
| CHAMBER_RUBBER_SCALE_AMPL                   | 1.0         | <b>0.0 (o)</b> |
| SCALE_CHAMBER_STIFFNESS                     | 1.0         | 1.0            |
| FLUID_SCALE_LINEAR_DAMPING                  | 1.0         | 1.0            |
| FLUID_SCALE_NONLINEAR_DAMPING               | 1.0         | 1.0            |
| FLUID_SCALE_TUNED_MASS_DAMPER               | 1.0         | 1.0            |
| SCALE_FREQ_SHIFT                            | 1.0         | 1.0            |
| FLUID_PRESSURE_LIMIT_BYPASS_AREA            | 0.0         | <b>0.0 (o)</b> |
| FLUID_PRESSURE_LIMIT_BYPASS_THRESHOLD       | 5.0E3 N/m^2 | 5.0E3 N/m^2    |

**Hydrobushing default values**

|                                      | HYDROMOUNT_FREQ_AMPL       | HYDROMOUNT_FREQ            |
|--------------------------------------|----------------------------|----------------------------|
| MAIN_RUBBER_USE_CURVE                | 'no'                       | 'no'                       |
| MAIN_RUBBER_STIFFNESS_STATIC         | -                          | -                          |
| ALPHA                                | -                          | -                          |
| BETA                                 | -                          | -                          |
| GAMMA                                | -                          | -                          |
| LAMBDA                               | -                          | -                          |
| MAIN_RUBBER_RHO                      | -                          | 3.8E-004 1/m               |
| MAIN_RUBBER_RDL                      | -                          | 600.0 m                    |
| MAIN_RUBBER_FRICTION_MAX_1           | 10*MAIN_RUBBER_RDL         | 10*MAIN_RUBBER_RDL         |
| MAIN_RUBBER_FRICTION_MAX_2           | MAIN_RUBBER_FRICTION_MAX_1 | MAIN_RUBBER_FRICTION_MAX_1 |
| MAIN_RUBBER_FRICTION_SIGNUM_GRADIENT | 1.0E9                      | <b>0.0 (o)</b>             |
| CHAMBER_RUBBER_STIFFNESS_STATIC      | -                          | -                          |
| CHAMBER_RUBBER_LAMBDA                | -                          | -                          |
| CHAMBER_RUBBER_RHO                   | -                          | 3.8E-004 1/m               |
| CHAMBER_RUBBER_RDL                   | -                          | 600.0 m                    |

|                                             |                               |                               |
|---------------------------------------------|-------------------------------|-------------------------------|
| CHAMBER_RUBBER_FRICTION_MAX_1               | 10*CHAMBER_RUBBER_RDL         | 10*CHAMBER_RUBBER_RDL         |
| CHAMBER_RUBBER_FRICTION_MAX_2               | CHAMBER_RUBBER_FRICTION_MAX_1 | CHAMBER_RUBBER_FRICTION_MAX_1 |
| CHAMBER_RUBBER_FRICTION_SIGNUM_GRADIENT     | 1.0E9                         | 0.0 (o)                       |
| FLUID_CHAMBER_AREA                          | -                             | -                             |
| FLUID_CHANNEL_AREA                          | -                             | -                             |
| FLUID_MASS                                  | -                             | -                             |
| FLUID_DAMPING_LINEAR                        | -                             | 0.0                           |
| FLUID_DAMPING_EXPONENT                      | 2.0                           | 1.0 (o)                       |
| FLUID_DAMPING_NONLINEAR                     | -                             | -                             |
| FLUID_DAMPING_EXPONENT_GRADIENT             | 1.0E9                         | 0.0 (o)                       |
| DECOUPLING_GRADIENT                         | 0.0                           | 0.0 (o)                       |
| DECOUPLING_CLEARANCE                        | 1.0E-6 m                      | 1.0E-6 m                      |
| FLUID_PRESSURE_LIMIT                        | 1.0E6 N/m^2                   | 1.0E6 N/m^2                   |
| FLUID_PRESSURE_LIMIT_SIGNUM_GRADIENT        | 1.0E9                         | 0.0 (o)                       |
| CHAMBER_PRELOAD_OPERATING_POINT             | -                             | -                             |
| CHAMBER_PRELOAD_SENSITIVITY_SIGNUM_GRADIENT | 0.0                           | 0.0 (o)                       |
| MAIN_RUBBER_SCALE_FREQ                      | 1.0                           | 1.0                           |
| MAIN_RUBBER_SCALE_AMPL                      | 1.0                           | 0.0 (o)                       |
| SCALE_MAIN_RUBBER_CURVE                     | 1.0                           | 1.0                           |
| CHAMBER_RUBBER_SCALE_EQ                     | 1.0                           | 1.0                           |
| CHAMBER_RUBBER_SCALE_AMPL                   | 1.0                           | 0.0 (o)                       |
| SCALE_CHAMBER_STIFFNESS                     | 1.0                           | 1.0                           |
| FLUID_SCALE_LINEAR_DAMPING                  | 1.0                           | 1.0                           |
| FLUID_SCALE_NONLINEAR_DAMPING               | 1.0                           | 1.0                           |
| FLUID_SCALE_TUNED_MASS_DAMPER               | 1.0                           | 1.0                           |
| SCALE_FREQ_SHIFT                            | 1.0                           | 1.0                           |
| FLUID_PRESSURE_LIMIT_BYPASS_AREA            | 0.0 (o)                       | 0.0 (o)                       |
| FLUID_PRESSURE_LIMIT_BYPASS_THRESHOLD       | 5.0E3 N/m^2                   | 5.0E3 N/m^2                   |

**Hydromount default values**

# 6 Appendix C: Advanced Topics

In this section the following advanced topics are discussed:

- [Environment Variables in VI-CarRealTime](#)
- [Using Ftire at best in VI-CarRealTime](#)
- [Building Custom Logics](#)

## 6.1 Environment Variables In VI-CarRealTime

Before running VI-CarRealTime users can set some environment variables in order to activate special features:

- **VICRT\_STATIC\_DT**: the variable represents the integration time step used during static analysis.
- **VICRT\_DISCRETE\_STEERING**: if the variable is set to 1 driver steering demand will be updated at driver sampling rate and will be kept constant for the whole driver sampling time interval, if the variable is set to 0 driver steering demand will be updated at solver integration rate using a linear integration.
- **VICRT\_DRV\_OUTSTEP**: by default the frequency of driving controller is assumed to be equal to results output step. Using this variable you can set directly the rate of driver making it independent by output step.
- **VI\_RUN\_WINDOW\_OFF**: if the variable is set to 1, the external window popping up for dumping solver output is disabled; output is dumped to message window.
- **VICRT\_INST\_DIR**: the variable is used to optionally set a custom path for VI-CarRealTime installation; it supersedes the value contained in system registry.
- **VIROAD\_INST\_DIR**: the variable is used to optionally set a custom path for VI-Road installation; it supersedes the value contained in system registry.
- **VIANIMATOR\_INST\_DIR**: the variable is used to optionally set a custom path for VI-Animator installation; it supersedes the value contained in system registry.
- **VISUSPENSIONGEN\_INST\_DIR**: the variable is used to optionally set a custom path for VI-SuspensionGen installation; it supersedes the value contained in system registry.
- **VITIRELIMITS\_INST\_DIR**: the variable is used to optionally set a custom path for VI-TireLimits installation; it supersedes the value contained in system registry.
- **VICRT\_APPT\_CMD**: the variable is used to set the Adams/PPT command to be executed when Adams/PPT is used as Post Processor. This environment variable set the Adams/PPT Command field in [Preferences](#) editor table.
- **COSIN\_PREFIX**: the variable is used to optionally set a custom path for cosin installation; it supersedes the default cosin path (`vicrt_installdir/acarrt/cosin` ). The environment variable path must be defined without quotes.

- **VICRT\_REMOVE\_BRAKE\_LOWSPEED\_MODEL:** if the variable is set to 1, the spring-damper rotational model acting on wheel spin axis at low speed is disabled.
- **VI\_KINGPIN\_EDIT:** if the variable is set to 1, the user has the possibility to edit the kingpin axis geometry maps.

Other environment variables are used in the context of Adams Car plugin:

- **VICRT\_SUSP\_ASY\_FROM\_FILES:** if the variable is set to 1, the creation of suspension assemblies for model export is forced to happen reading the subsystem from the database.
- **VICRT\_GYRO\_SENSORS:** if the variable is set to 1 the plugin creates an additional set of standard sensor outputs.
- **VICRT\_PLG\_ABS\_TCS\_OFF:** if the variable is set to 1, the built in controllers (TCS and ABS) are set to inactive after export.
- **VICRT\_GYRO\_SENSOR\_LOC:** if the variable is set to 1, the standard body sensor is created at gyro location.
- **VICRT\_FV\_SOLVER\_CHOICE:** the variable allows to define the solver type in full vehicle analysis executed during vehicle model export. Allowed values are FORTRAN and CPLUSPLUS; if not set the same solver type defined in Adams Car settings will be used.
- **VICRT\_PLG\_V17:** if the variable is bigger than 0, the plug-in behavior of version 17 is restored. It means that springs and bumpers are always deactivated when running [unsprung mass evaluation analysis](#).
- **VICRT\_EXPORT\_ELAPSED\_TIME:** if the variable is bigger than 0, the export log file will contain also lines defining the time elapsed to compute each section of the post-processing export procedure.
- **VICRT\_PLG\_EXPERIMENTAL:** if the variable is bigger than 0, for auxiliary export analysis the bumpstop elements activity is controlled by [Deactivate Bumpers](#) flag. Otherwise the bumpstop elements are switched off for such analysis.
- **VICRT\_PLG\_LEGACY\_AUX:** if the variable value is bigger than 0, the plug-in uses the old procedure to compute auxiliary vertical and auxiliary roll forces.
- **VICRT\_PLG\_STEER\_LEGACY:** if the variable value is bigger than 0, the plug-in uses the v18 procedure to compute rack steering kinematics and forces.
- **VICRT\_FTIRE\_MT\_CALL\_MODE=HIL:** when specified force FTIRE execution in HIL mode, required when executed in combination with VI-DriveSim.

The following environment variables can be used when compiling a custom plug-in:

- **MYSLIBS:** enter the full name of the user library to be included in the building process (i.e. set `MYSLIBS=C:\users\project\mylib_imp.lib`).
- **CRT\_KEEP\_FILES:** if the variable is set to 1, the temporary files generated by the building process are not erased.

## 6.2 Using FTire At Best In VI-CarRealTime

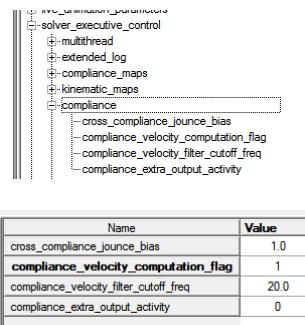
The basic action required to run FTire and FTire Realtime in combination with VI-CarRealTime is the assignment of a specific FTire .tir file to all the model wheels.

### Compliance Model

FTire computation requires the full roto-translational matrix of the wheel for both displacements and velocities: since the VI-CarRealTime suspension compliance's model is stationary, the velocity terms of the hub matrix may not match perfectly the displacement one, leading to FTire integration failures. In order to avoid this condition, it is recommended to enable the [compliance velocity flag](#) in the solver executive control block. The deviation

## Appendix C: Advanced Topics

from the stationary model is controlled by the [compliance velocity filter cutoff freq](#): 20hz is a typical value.

**Parallel Computation**

In order to achieve realtime performance for an FTire simulation, each FTire instance should be computed in parallel to the other. This behaviour is controlled by the [tire calc active flag](#) in the multithread subgroup. Two levels are supported:

1. tire parallelization
2. tire and vehicle parallelization (maximum efficiency)

**FTire Execution Mode**

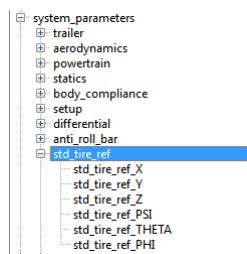
FTire core supports several run\_time\_mode that can be selected from the cosin tire editor or directly in the .tir file. The recommended mode for VI-CarRealTime is run\_time\_mode=6. With more recent FTIRE versions multiple cosimulation schemes are supported. In order to obtain the maximum computational efficiency from the F-Tire module, please make sure the MODEL block of your tir file contains the key:

```
cosimulation_mode = 1
```

Please refer to FTire documentation for more information related to the FTIRE numerical settings.

**Road/Path Orientation**

In most cases the FTire roads (RGR and CRG) are aligned to the positive global X axis so the most convenient way to make them consistent with the VI-CarRealTime default vehicle orientation is to remove the **pi** rotation along the Z axis of the std\_tire\_ref marker (**std\_tire\_ref\_PSI**):



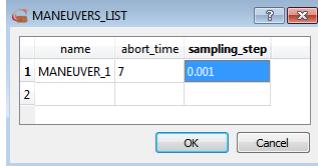
It may be also useful to translate the road origin by a few meters in order to make sure that all wheels are on the road surface during the static simulation: this can be achieved shifting the **std\_tire\_ref\_X** (when the **std\_tire\_ref\_PSI** is 0, the shift is typically negative and viceversa).

Road geometry to be used for visualization in VI-CarRealTime can be produced with VI-Road GUI for VI-Road RDF and CRG; for other road formats, wavefront objs can be obtained from the Cosin/Road module.

In case the event being executed use a DRD file to define the driver path, the drd definition should match the road one.

### VI-Driver Sampling Step

The steering control action produced by the closed loop driver is discrete with a sampling rate defined by the sampling time parameter stored in the MANEUVER\_LIST table of the VDF file. For FTire simulation it is recommended to define the sampling time equal to the solver integration step.



The path definition is also extremely important: please make sure that the assigned trajectory is clean. Reducing the driver interpolation step for the path is also a way to reduce undesired tire vibration.

#### Other remarks:

In case of simulation failure due to FTire instability it is recommended to check whether a smaller FTire integration step helps solving the problem. The FTire integration step is defined in the tir file as `maximum_time_step_RT`

**Note:** starting from version 2018.3, FTire requires a specific use mode in order to run in HIL environment like the VI-DriveSim one. In order to enable the HIL mode the [environment variable](#) `VICRT_FTIRE_MT_CALL_MODE=HIL` should be defined.

## 6.3 Building Custom Logics

The custom logics are implemented using particular data structure called **behavior tree**.

**Note:** custom logics generate a xbt file that can be used in MaxPerformance events. Such feature is an add-on which requires an additional license key. Please refer to [Licenses](#) topic for the details.

### 6.3.1 Behavior tree

The chapter consists of the following sections:

- [Introduction](#)
- [Building blocks](#)
- [Python bindings](#)

#### Introduction

Behavior tree allows to create an High Level Description of the correction logic used by [MaxPerformance](#) event to find the maximum performance.

The main properties of this data structures are:

1. Recursive decomposition
2. Hierarchical AI
3. Robust error management

4. Composite tasks
5. Leaves represent the interfaces with the maximum performance tool
6. Condition-action structure

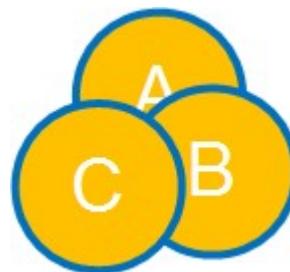
## Building blocks

The main elements of the behavior tree are:

- **Nodes :**  
used to manage logic and information flow, implementing Hierarchical AI;



- **Leaves :**  
represent conditions and actions on the environment;



- **Decorators :**  
are special nodes to filter/extend the flow management.



## Nodes

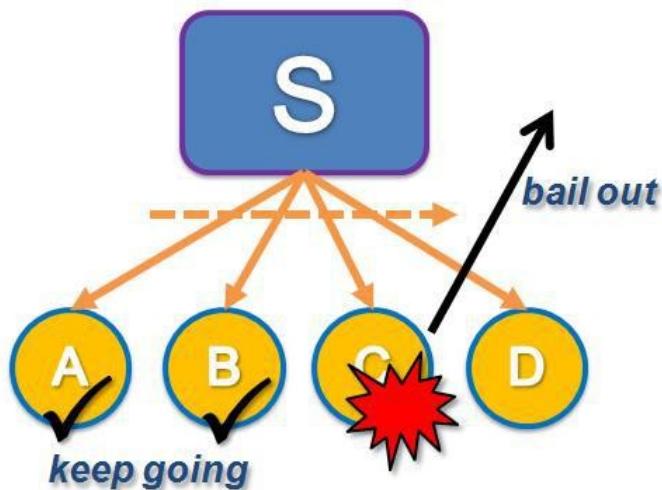
There are three types of node:

- [Sequence](#)
- [Selector](#)
- [Parallel](#)

## Sequence

Sequence node execute all the children starting from the left to the right in order, returning success if all the children succeed.

Sequence nodes are typically used to model work flows and logic sequences.



## Example:

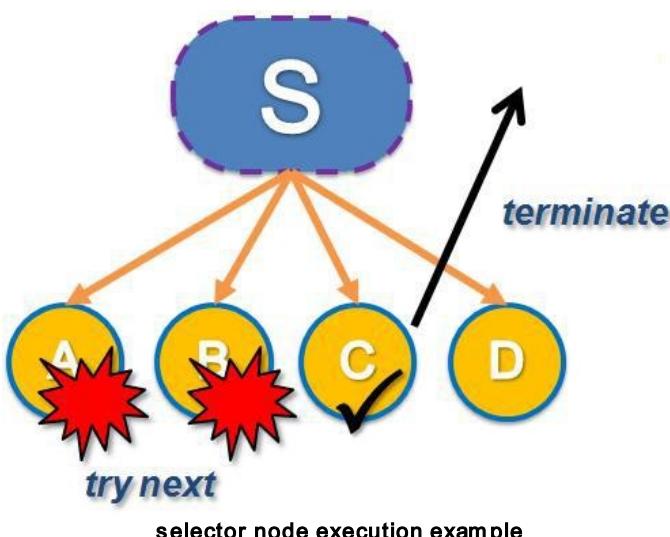
A set of conditions need to be checked in a particular order: "A", then "B", then "C" and finally "D". As soon as a condition check fails, the upper levels need to be notified.

It is enough to add the four conditions in the required order as children of a Sequence node, and the behavior tree will execute them in order, propagating the failure as soon as the first one fails.

## Selector

Selector node execute all the children starting from the left to the right in order, returning success if a children succeed.

Selector nodes are typically used to model prioritized sets of action.



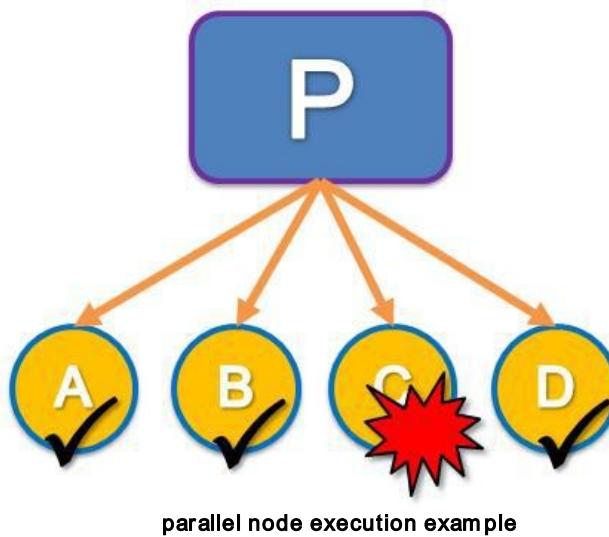
**Example:**

A list of possible corrections need to be tried in order, until the first one succeed: "A", then "B", then "C" and finally "D".

It is enough to add the four corrections in the required order as children of a Selector node, and the behavior tree will execute them in order, propagating the success as soon as the first one succeed.

**Parallel**

Parallel node perform a parallel execution of all the children, returning success or failure depending on the context. Parallel nodes are typically used to model concurrent behaviors.

**Leaves**

Leaves are nodes that represent condition, check or action and are application/task specific.

For example leaves can implement a check on path distance, tirelimits, understeer ratio, wheel lock, ecc. Or they can model actions such as scaling lateral performance, reducing throttle, increment longitudinal performance, ecc.

**Decorators**

Decorators are special elements that allow to "decorate" a node adding a specific behavior (i.e. filter message or stop on max iteration number)



The decorators are implemented using regular nodes (Sequence, Selector and Parallel) and specifying a "return policy".

For example a Selector node can be instructed to propagate failure if any of the executed children failed, or can be instructed to always propagate success or failure.

## Python bindings

[Introduction](#)

[Behavior tree reference](#)

[Utilities](#)

[Examples](#)

### Introduction

The custom logics behavior trees are represented as plain-text xml files (saved with the ".xbt" extension).

To provide an easier way to create logics, a set of Python bindings have been exposed, in the form of a standard python module.

The module "vitreebuilder" allows building correction logic trees using the Python programming language. Using VI-TreeBuilder is possible to create, load, manipulate and save (in *xml* or *gml* format) custom correction logic trees.

There are 8 objects that can be used:

- A node object, called "NodeSequence" , used to create nodes that visit all their children from the left to the right returning true if all children succeed by default. The return value depends on the policy selected from the available set. The root node must be of this type.
- A node object, called "NodeSelector" , used to create nodes that visit all his children from the left to the right, returning true if any children succeed by default. The return value depends on the policy selected from the available set.
- A node object, called "NodeParallel" , used to create nodes that visit all his children from the left to the right, returning true if any children succeed by default.
- A leaf object, called "LeafSimCondition" , used to create leaf nodes that checking condition over one or more simulation channel (with a maximum number of 5). The checking of the condition is performed

---

*Appendix C: Advanced Topics*

starting from the first (or last) simulation point until the stop condition or the last (or first) simulation point is reached.

- A leaf object, called “*LeafRelativeCondition*” , used to create leaf nodes that checking condition over one or more simulation channel (with a maximum number of 5). The checking of the condition is performed starting from the point is given by another condition, referenced by name, until the condition fails or the last (or first) simulation point is reached.
- A leaf object, called “*LeafSegmentCondition*” , used to create leaf nodes that checking condition over one or more simulation channel (with a maximum number of 5). The checking of the condition is performed only inside each segment until the condition fail or the last (or first) segment point is reached.
- A leaf object, called “*LeafCommandAction*” , used to create leaf nodes that modify the lateral and longitudinal performance according to the failure of a condition (referenced by name).
- A leaf object, called “*LeafCustomCommand*” , used to create leaf nodes that modify the lateral and longitudinal performance according to the failure of a condition (referenced by name). The acceleration, braking and lateral performance corrections can be specified using mathematical expressions.

To import the module the standard syntax is used:

```
import vitreebuilder
```

then the objects can be created using, for example:

```
root = vitreebuilder.NodeSequence("btroot")
```

### Behavior tree Python module reference

The following functions are available to create nodes and leaves, and to organize them in a tree hierarchy:

## Node sequence creation

To create a sequence node:

```
NodeSequence("node name", policy)
```

## Node selector creation

To create a selector node:

```
NodeSelector ("node name", policy)
```

## Node parallel creation

To create a parallel node:

```
NodeParallel ("node name", policy)
```

## Leaf condition

To create a condition leave:

```
createLeafSimCondition("condition name",
 "condition",
 "channel1",
 ["channel2", or ""],
 ["channel3", or ""],
 ["channel4", or ""],
 ["channel5", or ""])
```

## Leaf relative condition

To create a relative condition leave:

```
createLeafRelativeCondition("condition name",
 "condition",
 "relative condition name"
 "channel1",
 ["channel2", or ""],
 ["channel3", or ""],
 ["channel4", or ""],
 ["channel5", or ""])
```

## Leaf segment condition

To create a segment condition leaf:

```
createLeafSegmentCondition("condition name",
 "condition",
 "channel1",
 ["channel2", or ""],
 ["channel3", or ""],
 ["channel4", or ""],
 ["channel5", or ""])
```

## Leaf command action

To create an action leaf:

```
createLeafCommandAction("action name", "relative condition name", delta_Acc, delta_Dec, delta_Lon)
```

where the "delta\_%" values are added (with sign) to the actual performance scalings.

Every action has 2 additional parameters:

- weight that defines the "weight" of the action, in term of priority (1.0 by default);
- jointDistance that defines a threshold length used to extend the correction into the previous segment (50.0m by default).

These 2 additional parameters can be set using the follow syntax:

```
cond.weight = 1.0
cond.jointDistance = 25.0
```

## Leaf custom command action

To create an action leaf:

```
createLeafCustomCommand("action name",
 "relative condition",
 "lon Acc expression",
 "lon Dec expression",
 "lat expression",
 action weight,
 junction value,
 "channel1",
 ["channel2", or ""],
 ["channel3", or ""],
 ["channel4", or ""],
 ["channel5", or ""])
```

The "delta\_%" values are calculated by evaluating the "lon Acc", "lon Dec", "lat" expressions. The corrections are applied in the segment identified by the final point of the "relative condition" condition.

## Building behavior tree

To create a behavior tree hierarchy, child nodes or leaves are added using the `vitreebuilder.addChild()` command. For example:

```
btroot.addChild(node1)
node1.addChild(leaf1)
node1.addChild(leaf2)
```

## Node return policy

Nodes can have different return policy, in order to customize their default behavior. the following return policies can be used:

|                              |                                        |
|------------------------------|----------------------------------------|
| <code>BT_SUCED_IF_ALL</code> | Return success if all children succeed |
| <code>BT_SUCED_IF_ANY</code> | Return success if any children succeed |
| <code>BT_ALWAYS_SUCED</code> | Return success always                  |
| <code>BT_NEVER_SUCED</code>  | Return failure always                  |

## Expression function

Several functions can be used to create the expressions used in the condition and action leaves:

|                   |               |
|-------------------|---------------|
| <code>sin</code>  | sine          |
| <code>cos</code>  | cosine        |
| <code>tan</code>  | tangent       |
| <code>asin</code> | arcus sine    |
| <code>acos</code> | arcus cosine  |
| <code>atan</code> | arcus tangent |

|       |                                                  |
|-------|--------------------------------------------------|
| atan2 | quadrant based arcus tangent                     |
| sinh  | hyperbolic sine                                  |
| cosh  | hyperbolic cosine                                |
| tanh  | hyperbolic tangent                               |
| asinh | hyperbolic arcus sine                            |
| acosh | hyperbolic arcus cosine                          |
| atanh | hyperbolic arcus tangent                         |
| log2  | logarithm to base 2                              |
| log10 | logarithm to base 10                             |
| log   | logarithm to base 10                             |
| ln    | logarithm to base e (2.71828...)                 |
| exp   | e raised to the power x                          |
| sqrt  | square root of a value                           |
| sign  | sign function (-1 if $x < 0$ ; 1 if $x \geq 0$ ) |
| rint  | round to nearest integer                         |
| abs   | absolute value                                   |
| min   | var. min of all arguments                        |
| max   | var. max of all arguments                        |
| sum   | var. sum of all arguments                        |
| avg   | var. mean value of all arguments                 |

## Channel function and custom channel

In order to provide additional data to create checks and corrections, the following channels are available, in addition to all the standard channels available in a CarRealTime simulation:

|                                 |                                                 |
|---------------------------------|-------------------------------------------------|
| mxp.segment                     | Current segment                                 |
| mxp.last_segment                | Last segment number                             |
| mxp.segment_length              | Current segment length                          |
| mxp.segment_relative_s          | Current arch length relative to current segment |
| mxp.segment_scaling_lat         | Current segment lateral performance             |
| mxp.segment_scaling_IonAcc      | Current segment acceleration performance        |
| mxp.segment_scaling_IonDec      | Current segment deceleration performance        |
| mxp.segment_laptime             | Lap time at the end of the current segment      |
| mxp.segment_segmenttime         | Segment time at the end of the current segment  |
| mxp.segment_iterations          | Iterations made in the current segment          |
| mxp.previtx_segment_laptime     | Segment lap time in the previous iteration      |
| mxp.previtx_segment_segmenttime | Segment time in the previous iteration          |

A set of filters is also available. These filters can be applied on every channel and the filter name(s) and parameters are specified after the channel name, comma-separated, like in the following example:

```
cond1 = createLeafSimCondition("condition1",
 "atan2(c1,c2)>0.7",
 "chassis_lateral_velocity, avg, 10",
 "chassis_longitudinal_velocity, avg, 15")
cond2 = createLeafSimCondition("condition1",
 "c1>2 && abs(c2)<1.1",
 "mxp.segment",
 "path_distance, maxOnSeg")
```

In addition to the previous, are defined channel functions and additional channels that can be used defining conditions. The functions, unlike the previous, are applied to the entire channel, and are specified with their parameter (if needed) near channel name using ',' as separator ("channel, function, parameter").

The following filters are available:

|          |                                      |
|----------|--------------------------------------|
| avg, n   | Moving average filtering on n sample |
| min      | Min channel value                    |
| max      | Max channel value                    |
| minOnSeg | Min channel value on segment         |
| maxOnSeg | Max channel value on segment         |

## Creating tree files

In order to create the .xbt and the .gml (graphical representation of the tree) files representing the custom logic, the python script must be compiled executing the following command from a dos shell in the working directory:

**vig20 python script\_name.py**

### Utils

An additional Python module ("vitreebuilder\_utils"), containing commonly used check functions and corrections, is made available.

The following functions are available:

### checkTireMarginBySegment

Create a SegmentCondition that checks that at least one wheel keeps its force margin under a given threshold. The function takes as input the node name and the margin threshold.

```
checkTireMarginBySegment (name, marginThreshold)
```

### checkTireMarginLonBySegment

Create a SegmentCondition that checks that at least one wheel keeps its longitudinal force margin under a given threshold. The function takes as input the node name and the margin threshold.

```
checkTireMarginLonBySegment (name, marginThreshold)
```

### checkTireMarginLatBySegment

Create a SegmentCondition that checks that at least one wheel keeps its lateral force margin under a given threshold. The function takes as input the node name and the margin threshold.

```
checkTireMarginLatBySegment (name, marginThreshold)
```

## checkFullThrottleBySegment

Create a SegmentCondition that checks if the maximum throttle value in the current segment is 100%. This check is typically used identify the engine limit. The function takes as input the node name.

```
checkFullThrottleBySegment (name)
```

## checkFullBrakeBySegment

Create a SegmentCondition that checks if the maximum braking value in the current segment is bigger than 95%. This check is typically used identify the brake system limit. The function take as input the node name.

```
checkFullBrakeBySegment (name)
```

## checkPathDistance

Create a SimCondition that checks if the maximum path distance is under the given threshold. The function takes as input the node name and the path distance threshold.

```
checkPathDistance (name, threshold)
```

## customCommandAction

Create a CustomCommand that apply a performance scaling proportional to the longitudinal and lateral target accelerations. The function take as input the leaf name, the reference condition name and the correction step amount (delta).

```
customCommandAction (name, refCondName, delta)
```

## scaleLatAction

Create a CommandAction that scales the lateral performance by a fixed amount "delta". The function takes as input the leaf name, the reference condition name and the delta value.

```
scaleLatAction (name, refCondName, delta)
```

## scaleAccAction

Create a CommandAction that scales the acceleration performance by a fixed amount "delta". The function takes as input the leaf name, the reference condition name and the delta value.

```
scaleAccAction (name, refCondName, delta)
```

## scaleDecAction

Create a CommandAction that scales the braking performance by a fixed amount "delta". The function takes as input the leaf name, the reference condition name and the delta value.

```
scaleDecAction (name, refCondName, delta)
```

## Examples

The following examples are discussed:

- [Using tire limits](#)
- [Keep the vehicle in a given GG range](#)

## Using tirelimits

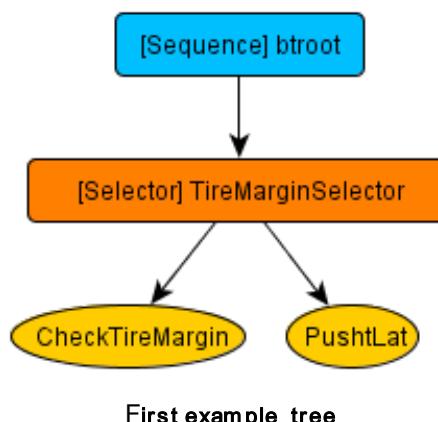
In this section, two examples are presented. They show how to push longitudinal and lateral performances using information coming from the "tirelimits" routines.

The first example uses tirelimits to push lateral performance if the minimum tire margin in each segment is bigger than a threshold.

This simple tree, using a selector node, implements the condition and action structure, executing the action only if the check on the tire margin fails.

In this case the margin is higher than the threshold and the condition fail.

The sub tree composed by the two leaf and the selector node is the only child of the root, the main node of the behavior tree that must be a sequence node.



The tree is represented by an xml file that can be created using a python script like the one explained below.

Every script must starts with the follow block of code that performs the import of the vitreebuilder module and its dependencies:

```

import sys,os

import vigrade.viscripting
import vigrade.viscripting.viframework
sys.path.append(os.path.dirname(vigrade.viscripting.__file__))
sys.path.append(os.path.dirname(vigrade.viscripting.viframework.__file__))

import vitreebuilder as tb

```

After the import of all the support libraries, the root node is created ("btroot"), and also the main selector node ("TireMarginSelector").

The selector uses the BT\_SUCCED\_IF\_ALL policy, in order to propagate the check failure to the root and make the simulation aware that an unfeasibility has been found.

```
create root node
root = tb.NodeSequence("btroot")

level 1 nodes
tireMarginSelector = tb.NodeSelector("TireMarginSelector",tb.BT_SUCCEED_IF_ALL)
```

The first leaf is the condition leaves that performs the tire margin check over each segment.

Starting from the first point of each segment, the condition checks if the minimum margin (measured in each segment) of almost one of the 4 wheel is smaller or equal to 50N returning false if not.

```
check tire force margin using tirelimits channels
checkTireMargin = tb.createLeafSegmentCondition("CheckTireMargin",
 "min(c1,c2,c3,c4)<=50",
 "+1",
 "tirelimits_FL.margin,minOnSeg",
 "tirelimits_RL.margin,minOnSeg",
 "tirelimits_FR.margin,minOnSeg",
 "tirelimits_RR.margin,minOnSeg")
```

The second leaf is the action executed in case the margin check fails. This action increments the lateral performance by a "deltaLat" value of 0.05 units, in the current segment only (jointDistance = 0).

```
corrections
deltaLat=0.05
pushLat = tb.createLeafCommandAction("PushtLat", "CheckTireMargin" , 0.0, 0.0,
 deltaLat)
pushLat.joinDistance = 0
```

Starting from the root to the leaves, using the addChild() function is now possible to connect every element in order to obtain the desired tree.

```
root.addChild(tireMarginSelector)
tireMarginSelector.addChild(checkTireMargin)
tireMarginSelector.addChild(pushLat)
```

At the end the tree is saved in XML (.xbt) and GML format. The first one is used by the MaxPerformance while the second is a standard format that can be used to graphically represent the behavior tree using external tools.

```
save to file
tb.saveBehaviorTree(root,"push_tirelimits_simple.xbt");
tb.saveBehaviorTreeGml(root,"push_tirelimits_simple.gml");
```

This second example creates a more complex custom logic that increment the lateral, the acceleration and the deceleration performances; this is obtained by composition of simpler sub-trees, each one responsible for a specific task.

The tree uses the global tire margin to perform a main check and, in case of failure, apply a composite correction action.

The correction is created using a parallel node that executes simultaneously two different sub-trees: one responsible for the lateral performance and the other for the longitudinal.

The lateral performance check is similar to the previous example, using a selector node to implement the condition and action structure, the lateral performance is "pushed" if the lateral margin is bigger than a threshold.

Regarding to the longitudinal performance, more aspects need to be considered:

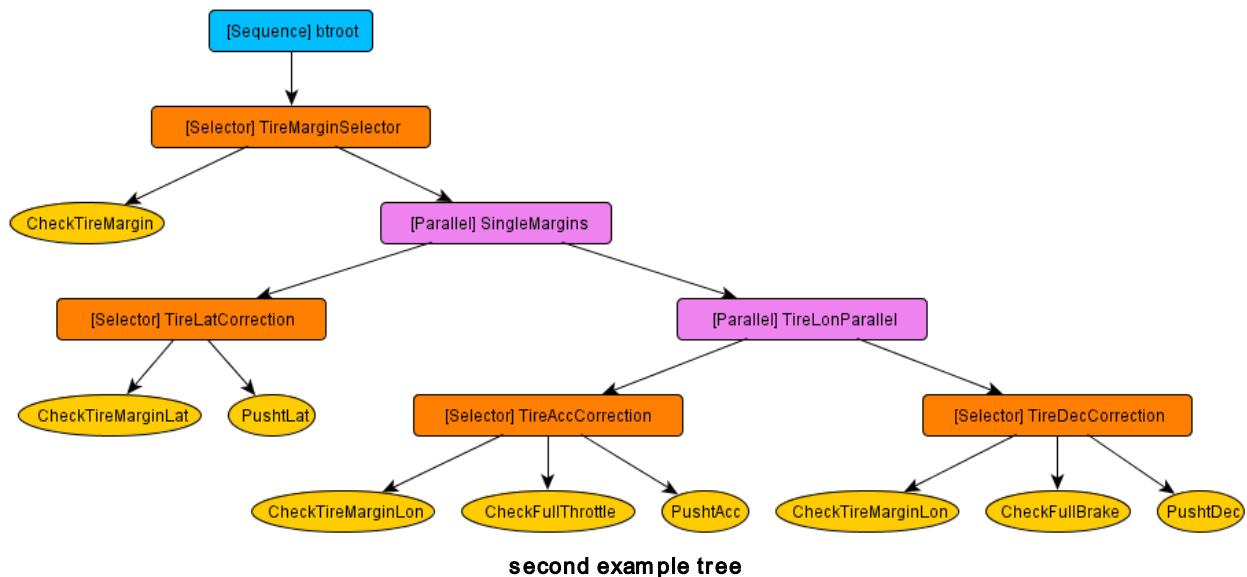
## Appendix C: Advanced Topics

- the longitudinal performance must be defined by decoupling acceleration and deceleration performances;
- the longitudinal performance may be limited by the vehicle (engine and brakes) or by the tires in each segment.

Each one of these two aspects is translated into a sub-tree that executes in parallel two different check:

- check if there is longitudinal margin in acceleration and the vehicle isn't engine-limited (no full throttle in the current segment). In case increment the acceleration performance;
- check if there is longitudinal margin in deceleration and the vehicle isn't limited by the brakes (no full brake in the current segment). In case increment the deceleration performance.

The resulting structure is the following:



```

import sys,os

import vigrade.viscripting
import vigrade.viscripting.viframework
sys.path.append(os.path.dirname(vigrade.viscripting.__file__))
sys.path.append(os.path.dirname(vigrade.viscripting.viframework.__file__))

import vitreebuilder as tb

create root
root = tb.NodeSequence("btroot")

level 1 nodes
tireMarginSelector = tb.NodeSelector("TireMarginSelector",tb.BT_SUCCEED_IF_ALL)
singleMarginParallel = tb.NodeParallel("SingleMargins")
tireLatSelector = tb.NodeSelector("TireLatCorrection")
tireLonParallel = tb.NodeParallel("TireLonParallel")
tireAccSelector = tb.NodeSelector("TireAccCorrection")
tireDecSelector = tb.NodeSelector("TireDecCorrection")

```

```

tire force margin using tirelimits channels
checkTireMargin = tb.createLeafSegmentCondition("CheckTireMargin",
 "min(c1,c2,c3,c4)<=50",
 "+1",
 "tirelimits_FL.margin,minOnSeg",
 "tirelimits_RL.margin,minOnSeg",
 "tirelimits_FR.margin,minOnSeg",
 "tirelimits_RR.margin,minOnSeg")

checkTireLonMargin = tb.createLeafSegmentCondition("CheckTireMarginLon",
 "min(c1,c2,c3,c4)<=50",
 "+1",
 "tirelimits_FL.marginLon,minOnSeg",
 "tirelimits_RL.marginLon,minOnSeg",
 "tirelimits_FR.marginLon,minOnSeg",
 "tirelimits_RR.marginLon,minOnSeg")

checkTireLatMargin = tb.createLeafSegmentCondition("CheckTireMarginLat",
 "min(c1,c2,c3,c4)<=50",
 "+1",
 "tirelimits_FL.marginLat,minOnSeg",
 "tirelimits_RL.marginLat,minOnSeg",
 "tirelimits_FR.marginLat,minOnSeg",
 "tirelimits_RR.marginLat,minOnSeg")

check if throttle-limited on the current segment
checkFullThrottle = tb.createLeafSegmentCondition("CheckFullThrottle",
 "c1==100",
 "+1",
 "driver_demands.throttle,maxOnSeg")

check if brake-limited on the current segment
checkFullBrake = tb.createLeafSegmentCondition("CheckFullBrake",
 "c1>=95",
 "+1",
 "driver_demands.brake,maxOnSeg")

corrections
deltaLat=0.05
deltaAcc=0.05
deltaDec=0.05
pushLat = tb.createLeafCommandAction("PushtLat", "CheckTireMargin" , 0.0, 0.0, deltaLat);
pushLat.joinDistance = 0;
pushAcc = tb.createLeafCommandAction("PushtAcc", "CheckTireMargin" , deltaAcc, 0.0, 0.0);
pushAcc.joinDistance = 0;
pushDec = tb.createLeafCommandAction("PushtDec", "CheckTireMargin" , 0.0, deltaDec, 0.0);
pushDec.joinDistance = 0;

#
create tree
#####
root.addChild(tireMarginSelector)
tireMarginSelector.addChild(checkTireMargin)
tireMarginSelector.addChild(singleMarginParallel)
singleMarginParallel.addChild(tireLatSelector)
tireLatSelector.addChild(checkTireLatMargin)
tireLatSelector.addChild(pushLat)
singleMarginParallel.addChild(tireLonParallel)
tireLonParallel.addChild(tireAccSelector)
tireAccSelector.addChild(checkTireLonMargin)

```

## Appendix C: Advanced Topics

```
tireAccSelector.addChild(checkFullThrottle)
tireAccSelector.addChild(pushAcc)
tireLonParallel.addChild(tireDecSelector)
tireDecSelector.addChild(checkTireLonMargin)
tireDecSelector.addChild(checkFullBrake)
tireDecSelector.addChild(pushDec)

save to file
tb.saveBehaviorTree(root,"push_tirelimits_full.xbt");
tb.saveBehaviorTreeGml(root,"push_tirelimits_full.gml");
```

**Keep the vehicle in a given GG range**

Using a custom correction logic it is also possible to keep the vehicle in a specific range of performances. This example shows how to find automatically a speed profile that maintains vehicle lateral and longitudinal accelerations inside a given range.

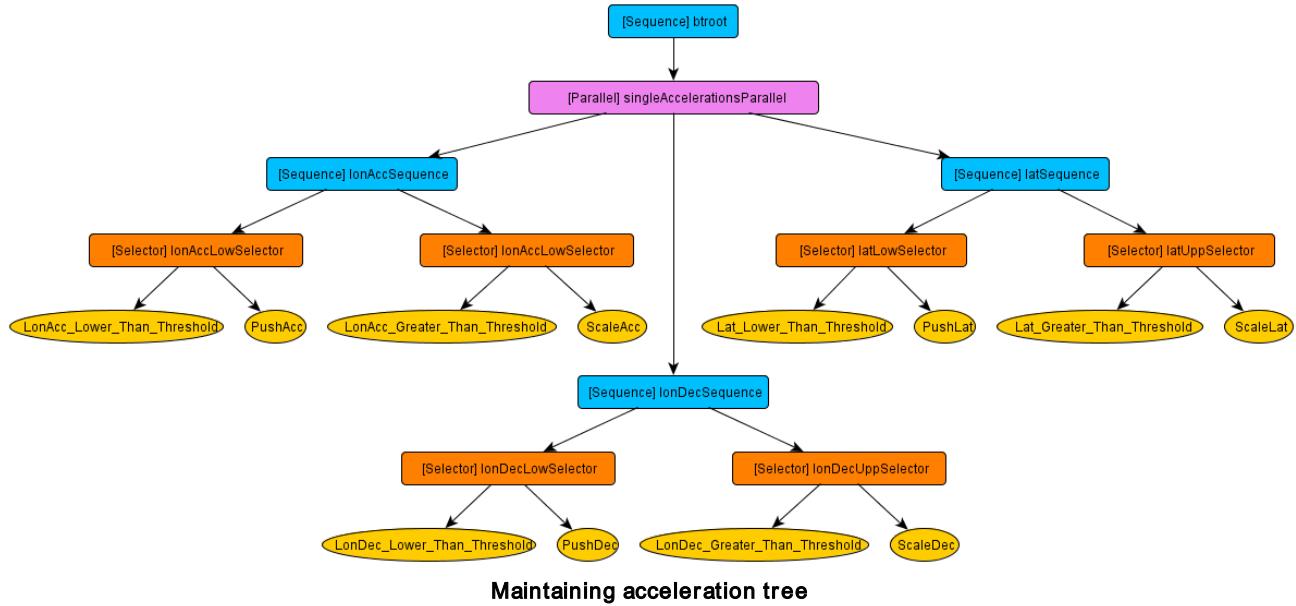
Like in the previous example, here corrections on the lateral, acceleration and deceleration performances are applied. The logics operates in parallel on the lateral, acceleration and deceleration dynamics:

- *Lateral dynamics* : a sequence node executes two checks on the lateral acceleration pushing the lateral performance if it is below the given range or, scaling down if it is above the range. If the acceleration is within the range "success" is returned.
- *Acceleration dynamics* : also here a sequence node is used; the check is performed on both the longitudinal acceleration and the throttle pedal (to verify if the performance is engine-limited). In case of check failure, the performance is pushed/scaled down accordingly.
- *Deceleration dynamics* : again a sequence node is used; the check is performed on both the longitudinal deceleration and the brake pedal (to verify if the performance is brake-limited). In case of check failure, the performance is pushed/scaled down accordingly.

```
checkLonAccLow = tb.createLeafSegmentCondition("LonAcc_Lower_Than_Threshold",
 "abs(c1)>=0.5 || c2==100",
 +1,
 "chassis_accelerations.longitudinal,maxOnSeg",
 "driver_demands.throttle,maxOnSeg")
```

Note in the above code example the usage of:

- the expression represents a composite condition, to verify the acceleration and throttle values. The logic "OR" allows to skip the acceleration check in case the engine limit is reached.
- both channels use filters; in this example the maximum inside each segment is calculated, using the keyword "maxOnSeg"



```

import sys,os

import vigrade.viscripting
import vigrade.viscripting.viframework
sys.path.append(os.path.dirname(vigrade.viscripting.__file__))
sys.path.append(os.path.dirname(vigrade.viscripting.viframework.__file__))

import vitreebuilder as tb

#
create objects
#

create root
root = tb.NodeSequence("btroot")

level 1 nodes
singleMarginParallel =
tb.NodeParallel("singleAccelerationsParallel",tb.BT_SUCCEED_IF_ALL)

level 2 nodes
lonAccSequence = tb.NodeSequence("lonAccSequence")
lonDecSequence = tb.NodeSequence("lonDecSequence")
latSequence = tb.NodeSequence("latSequence")

level 3 nodes
lonAccLowSelector = tb.NodeSelector("lonAccLowSelector",tb.BT_SUCCEED_IF_ALL)
lonAccUpSelector = tb.NodeSelector("lonAccUpSelector",tb.BT_SUCCEED_IF_ALL)
lonDecLowSelector = tb.NodeSelector("lonDecLowSelector",tb.BT_SUCCEED_IF_ALL)
lonDecUpSelector = tb.NodeSelector("lonDecUpSelector",tb.BT_SUCCEED_IF_ALL)
latLowSelector = tb.NodeSelector("latLowSelector",tb.BT_SUCCEED_IF_ALL)
latUpSelector = tb.NodeSelector("latUpSelector",tb.BT_SUCCEED_IF_ALL)

lateral and longitudinal accelerations

```

## Appendix C: Advanced Topics

```

checkLonAccLow = tb.createLeafSegmentCondition("LonAcc_Lower_Than_Threshold",
 "abs(c1)>=0.5 || c2==100",
 +1,
 "chassis_accelerations.longitudinal,maxOnSeg",
 "driver_demands.throttle,maxOnSeg")
checkLonAccUpp = tb.createLeafSegmentCondition("LonAcc_Greater_Than_Threshold",
 "abs(c1)<=0.6",
 +1,
 "chassis_accelerations.longitudinal,maxOnSeg")
checkLonDecLow = tb.createLeafSegmentCondition("LonDec_Lower_Than_Threshold",
 "abs(c1)>=0.5 || c2>=95",
 +1,
 "chassis_accelerations.longitudinal,minOnSeg",
 "driver_demands.brake,maxOnSeg")
checkLonDecUpp = tb.createLeafSegmentCondition("LonDec_Greater_Than_Threshold",
 "abs(c1)<=0.6",
 +1,
 "chassis_accelerations.longitudinal,minOnSeg")

checkLatLow = tb.createLeafSegmentCondition("Lat_Lower_Than_Threshold",
 "max(abs(c1),abs(c2))>=0.6",
 +1,
 "chassis_accelerations.lateral,maxOnSeg",
 "chassis_accelerations.lateral,minOnSeg")
checkLatUpp = tb.createLeafSegmentCondition("Lat_Greater_Than_Threshold",
 "max(abs(c1),abs(c2))<=0.7",
 +1,
 "chassis_accelerations.lateral,maxOnSeg",
 "chassis_accelerations.lateral,minOnSeg")

corrections
deltaLat=0.05
deltaAcc=0.05
deltaDec=0.05
pushLat = tb.createLeafCommandAction("PushLat", "Lat_Lower_Than_Threshold" , 0.0, 0.0,
 deltaLat);
pushLat.joinDistance = 0;
pushAcc = tb.createLeafCommandAction("PushAcc", "LonAcc_Lower_Than_Threshold" ,
 deltaAcc, 0.0, 0.0);
pushAcc.joinDistance = 0;
pushDec = tb.createLeafCommandAction("PushDec", "LonDec_Lower_Than_Threshold" , 0.0,
 deltaDec, 0.0);
pushDec.joinDistance = 0;
scaleLat = tb.createLeafCommandAction("ScaleLat", "Lat_Greater_Than_Threshold" , 0.0,
 0.0, -deltaLat);
scaleLat.joinDistance = 0;
scaleAcc = tb.createLeafCommandAction("ScaleAcc", "LonAcc_Greater_Than_Threshold" , -deltaAcc, 0.0, 0.0);
scaleAcc.joinDistance = 0;
scaleDec = tb.createLeafCommandAction("ScaleDec", "LonDec_Greater_Than_Threshold" , 0.0,
 -deltaDec, 0.0);
scaleDec.joinDistance = 0;

```

```
#
create tree
#####
root.addChild(singleMarginParallel)

singleMarginParallel.addChild(lonAccSequence)
lonAccSequence.addChild(lonAccLowSelector)
lonAccLowSelector.addChild(checkLonAccLow)
lonAccLowSelector.addChild(pushAcc)
lonAccSequence.addChild(lonAccUppSelector)
lonAccUppSelector.addChild(checkLonAccUpp)
lonAccUppSelector.addChild(scaleAcc)

singleMarginParallel.addChild(lonDecSequence)
lonDecSequence.addChild(lonDecLowSelector)
lonDecLowSelector.addChild(checkLonDecLow)
lonDecLowSelector.addChild(pushDec)
lonDecSequence.addChild(lonDecUppSelector)
lonDecUppSelector.addChild(checkLonDecUpp)
lonDecUppSelector.addChild(scaleDec)

singleMarginParallel.addChild(latSequence)
latSequence.addChild(latLowSelector)
latLowSelector.addChild(checkLatLow)
latLowSelector.addChild(pushLat)
latSequence.addChild(latUppSelector)
latUppSelector.addChild(checkLatUpp)
latUppSelector.addChild(scaleLat)

save to file
tb.saveBehaviorTree(root,"ggControl.xbt");
tb.saveBehaviorTreeGml(root,"ggControl.gml");
```

# 7 Installation Guide

---

## 7.1 VI-CarRealTime 20.0 installation on Windows

Welcome to the installation guide for VI-CarRealTime 20.0.

The following topics are available:

- [How To Get The Software](#)
- [Running the Installer](#)
- [Server Installation](#)
- [Client Installation](#)
- [Batch Installation](#)
- [Uninstalling](#)

### 7.1.1 How To Get The Software

In order to perform an installation of VI-CarRealTime, you need to pickup the installation packages from the VI-grade [website support](#) area (registration is required).

VI-CarRealTime is shipped primarily in 64 bit form:

- **VI\_Crt\_20\_0\_x64\_Setup.exe**: main product 64 bit installer, including the entire VI-CarRealTime suite.
- **VI\_Crt\_20\_0\_Installing.pdf**: this document.
- **VI\_Crt\_20\_0\_Release\_Notes.pdf**: product information.

In case you are interested in Adams Car export capability, you need to download the package for the desired Adams version:

- **VI\_Crt\_plugin\_2020\_20\_0\_x64\_Setup.exe**: VI-CarRealTime interface for Adams Car 2020.0 64 bit.
- **VI\_Crt\_plugin\_2019\_20\_0\_x64\_Setup.exe**: VI-CarRealTime interface for Adams Car 2019.0 64 bit
- **VI\_Crt\_plugin\_2018\_20\_0\_x64\_Setup.exe**: VI-CarRealTime interface for Adams Car 2018.0 64 bit.
- 

Finally, several extensions, related to the hardware in the loop (HIL) targets, are available as separate installation packages:

- **VI\_Crt\_scalexio\_20\_0\_r18b\_Setup.exe**: extension package for the dSPACE Scalexio target
- **VI\_Crt\_ni\_pxi\_20\_0\_x86\_Setup.exe**: extension package for the National Instruments PXI environment
- **VI\_Crt\_etas\_20\_0\_x86\_Setup.exe**: extension package for the ETAS target

## 7.1.2 Running the Installer

After reviewing the Release notes document, the installation could start following the steps below:

1. Double click over the **VI\_Crt\_20\_0\_Setup.exe** file, and follow the instructions that will appear on the screen.

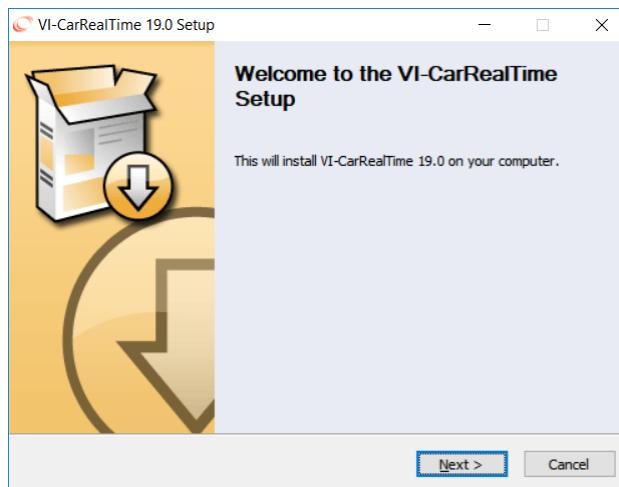
[Product Information](#)

[License Agreement](#)

[Installation Type](#)

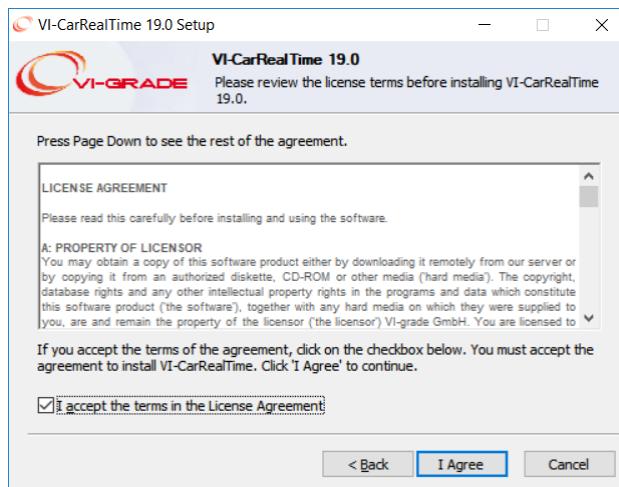
### Product Information

Just click on the “Next” button to proceed.



### License Agreement

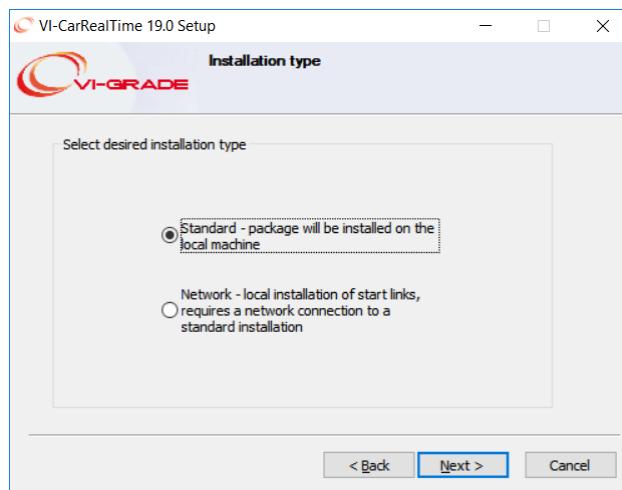
Read the license agreement carefully. The check box will enable the “I Agree” button required to proceed with the application setup.



## Installation Type

Select the type of installation you desire:

- [Standard](#): installs VI-CarRealTime in the target location.
- [Network](#): allows to call VI-CarRealTime installed on a remote machine sharing a standard installation.



### 7.1.3 Standard Installation

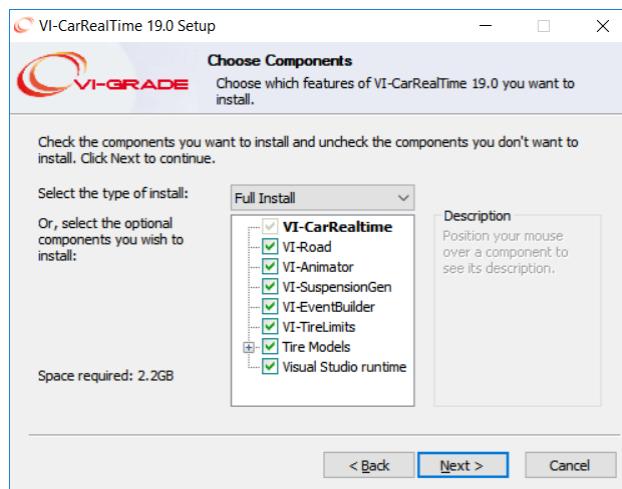
The standard installation is composed by the following sections:

- [Components](#)
- [Installation Folder](#)
- [Installing](#)

## Components

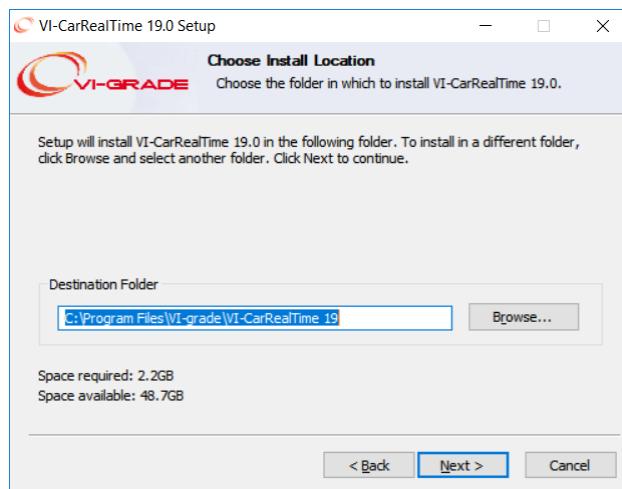
If you have chosen the standard installation type, select the components you want to install.

Click "Next".



## Installation Folder

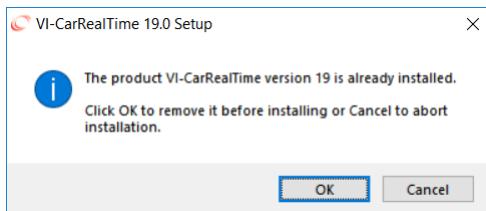
If you have selected the standard installation type choose a desired installation folder where you want VI-CarRealTime files to be copied and click "Next".



Please, notice that the installer will install VI-CarRealTime accessories into directories at the same level of the one you chose. As an example, if you chose to install VI-CarRealTime at *C:\Program Files\VI-grade\VI-CarRealTime 20* then the accessories will be installed at *C:\Program Files\VI-grade\VI-Road 20*, *C:\Program Files\VI-grade\VI-Animator 20* and so on.

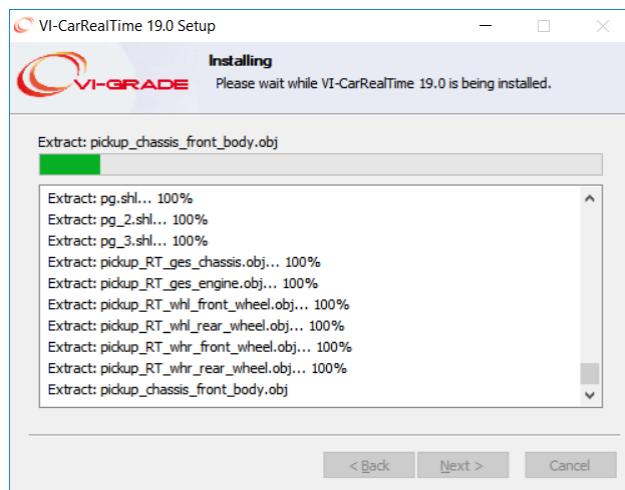
## Installing

If another installation of the *same* version of VI-CarRealTime is present, you will be prompted with the following message:



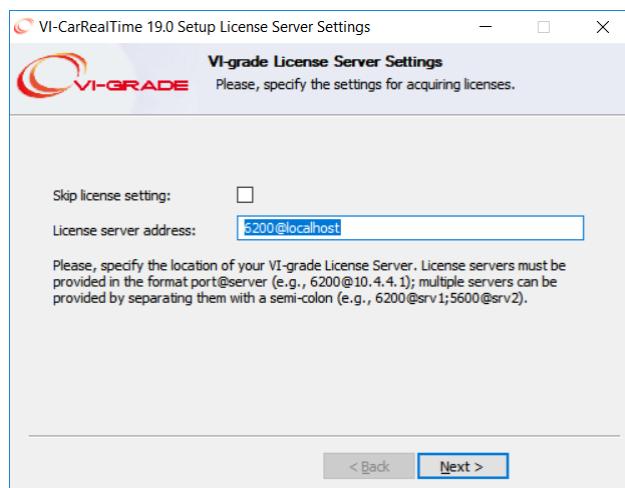
That's because you cannot have multiple installations of VI-CarRealTime featuring the same version on the same machine. If this message appears then, please, accept by pushing the OK button.

Once the installation has started, wait until the process is complete.

*Installation Guide*

At the end of the installation you'll be asked to enter the VI-CarRealTime license.  
You can specify an IP or an alias:

6200@*hostname* (6200@localhost if the license is on the same machine on which you're installing)  
or  
6200@*IPaddress*



Now the files should be copied into the installation directory.



Just select the "Finish" button to complete the installation procedure. You will find a shortcut to VI-CarRealTime on Windows application menu under VI-grade folder.

At the end of the installation procedure, a file named: install-crt-20-0.log will be available in the installation directory.

## 7.1.4 Network Installation

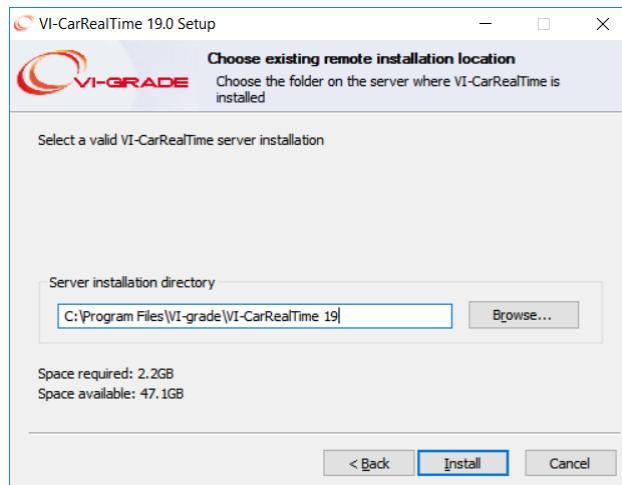
The network installation is composed by the following sections:

- [Installation Folder](#)
- [Installing](#)

**Note:** the network installation mode is designed to allow the execution on the current machine of an installation residing a different host (on which a standard installation has been performed). For most of the cases, the standard installation mode should satisfy your needs.

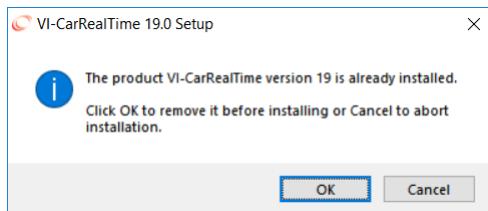
### Installation Folder

If you have selected the network installation type, select the network path to an existing VI-CarRealTime standard installation.



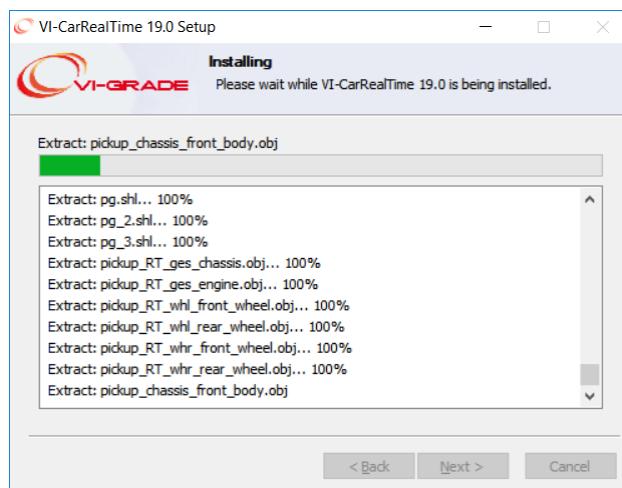
## Installing

If another installation of the *same* version of VI-CarRealTime is present, you will be prompted with the following message:



That's because you cannot have multiple installations of VI-CarRealTime featuring the same version on the same machine. If this message appears then, please, accept by pushing the OK button.

Once the installation has started, wait until the process is complete.

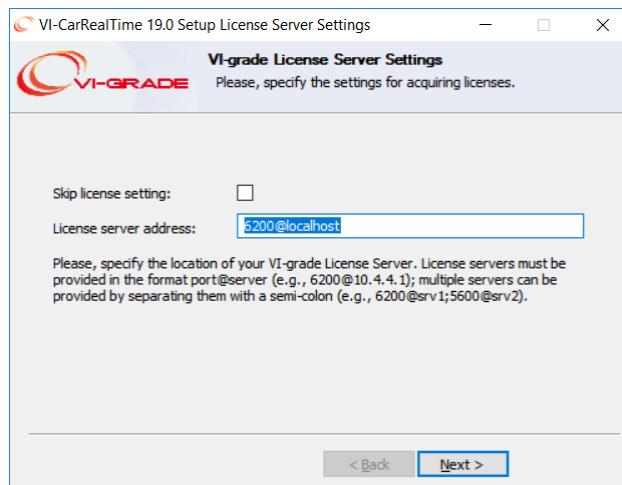


At the end of the installation you'll be asked to enter the VI-CarRealTime license. You can specify an IP or an alias:

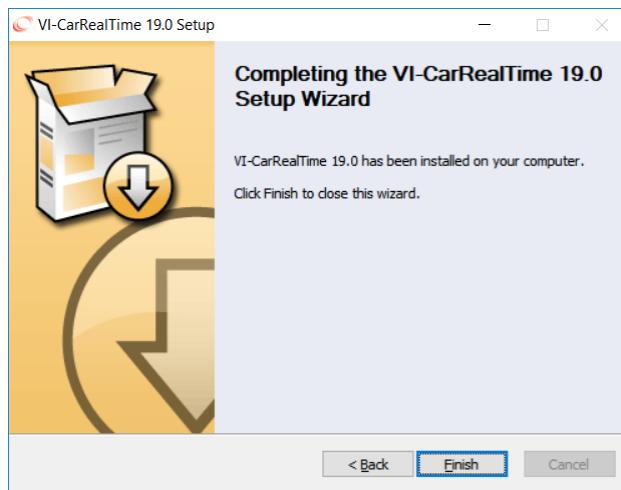
6200@hostname (6200@localhost if the license is on the same machine on which you're installing)

or

6200@IPaddress



Now the files should be copied into the installation directory.



Just select the "Finish" button to complete the installation procedure. You will find a shortcut to VI-CarRealTime on Windows application menu under VI-grade folder.

At the end of the installation procedure, a file named: install-crt-20-0.log will be available in the installation directory.

## 7.1.5 Batch Installation

It is also possible to run the VI-CarRealTime installer in batch mode using the /S option (silent mode). Additional available options are:

**/INSTTYPE:** available options are:

- standard
- network

**/INSTDIR:** sets the installation folder (mandatory)

**/MSCLIC:** for client installation, sets the path for MSC license used for ADAMS/PPT (optional)

**/VIGLIC:** for client installation, sets the path for VI-grade license (optional)

### Examples:

#### Batch standard installation

```
VI_Crt_20_0_x64_Setup.exe /S /INSTTYPE=standard /INSTDIR="C:\Program Files\VI-grade\VI-CarRealTime".
```

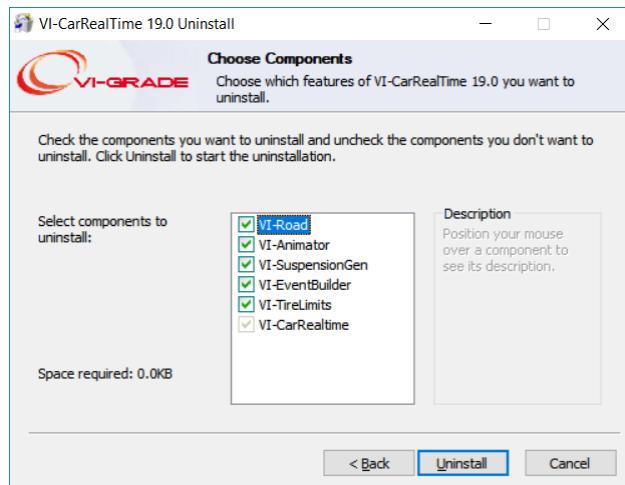
#### Batch network installation

```
VI_Crt_20_0_x64_Setup.exe /S /INSTTYPE=network /INSTDIR="X:\Program Files\VI-grade\VI-CarRealTime" /MSCLIC=27500@licserver /VIGLIC=6200@licserver.
```

## 7.1.6 Uninstalling

The uninstallation of VI-CarRealTime can be performed by using Windows Programs and Features tool or by selecting **uninstall\_vicrt.exe** stored in VI-CarRealTime installation folder. You will then be prompted with the following forms:

## Installation Guide



This is the list of the components that are part of VI-CarRealTime and that can be used even if VI-CarRealTime is not installed. By default, all of them will be uninstalled. The uninstall procedure will look for those components in the directories where they were installed by the VI-CarRealTime installer. Those that cannot be found will be skipped.

## 7.2 Setting up the License

Running VI-grade products always requires the installation of VI-Licensing on the local computer in case of a node locked license or on a computer (license server) in the network in case of a network license.

We recommend to install VI-Licensing prior to installing the application although it is not absolutely required. Please note that VI-grade licensing version 20.0 or newer is required to run this product. You should have received detailed information regarding the installation and operation of the licensing system together with the license file. If not please download the licensing system for your platform from the VI-grade [website support](#) area (registration is required).

No matter what type of license you are running (**node locked license** or **network floating**), during the installation of the client application, you need to define on which machine the license server is running. Every LM-X protected application has a search path for licenses defines as follows:

| Order | Search path |
|-------|-------------|
|       |             |

1 The environment variable VIGRADE\_LICENSE\_PATH.

On Windows we recommend setting the environment variable using the system control panel:

Variable Name: **VIGRADE\_LICENSE\_PATH**

Value: **6200@servermachine**

## Notes:

- for a node-locked license it is sufficient to set `VIGRADE_LICENSE_PATH` value to `6200@localhost` .
  - VI-grade currently issues only license that uses the `COUNT` key and as such requires the operation of a license server also for node-locked licenses.
  - Windows and Linux OS requires different separators when declaring multiple servers on the `VIGRADE_LICENSE_PATH` environment variable:

Windows: VIGRADE\_LICENSE\_PATH=6200@server1 ;6200@server2  
Linux: VIGRADE\_LICENSE\_PATH=6200@server1 ;6200@server2

In order to run simulation with the F-Tire module please make sure to properly configure the connection to a valid F-Tire license server using the Cosin/Tools module available in the Utilities menu of the VI-CarRealTime interface. Please refer to the F-Tire documentation or to the Cosin Scientific Software website ([www.cosin.eu](http://www.cosin.eu)) for problems related to F-Tire.

In order to run simulation with the MF-Swift module, please make sure to properly configure the connection to a valid TASS license server using the environment variable `MADLIC_LICENSE_FILE`. Please refer to the MF-Tyre documentation for more information on MF-Tyre installation.

## 7.3 Troubleshooting

The aim of this topic is to report the most common troubleshooting with licensing.

## 1. Problems with license server

If the software doesn't start up, the first thing to check is that license server has been correctly set up. In particular it can be useful to look at *lmx-serv.log* file stored in VI-Licensing folder in the server machine. Such file logs all main server activities and if something goes wrong during server initialization (read license keys), or during check-out or check-in of a feature key, a message will be written in the log file.

Here the statements that are generally written when server initialization has been successful:

```
[2017-02-17 11:23:25] LM-X License Server v4.6.4 build a9ec77a9 on TPW520 (Win32_x86)
[2017-02-17 11:23:25] Copyright (C) 2002-2014 X-Formation. All rights reserved.
[2017-02-17 11:23:25] Website: http://www.lm-x.com http://www.x-formation.com
[2017-02-17 11:23:25] License server has pid 4800.
[2017-02-17 11:23:25] Serving licenses for vendor VIGRADE.
[2017-02-17 11:23:25]
[2017-02-17 11:23:25] License server using TCP IPv4 port 6200.
[2017-02-17 11:23:25] License server using TCP IPv6 port 6200.
[2017-02-17 11:23:25] License server using UDP IPv4 port 6200.
[2017-02-17 11:23:25] Reading licenses...
[2017-02-17 11:23:25] License file(s):
[2017-02-17 11:23:25] C:\Program Files (x86)\VI-grade\VI-Licensing\license.lic
[2017-02-17 11:23:28] Log file path: C:\Program Files (x86)\VI-grade\VI-Licensing\lmx-serv.log
[2017-02-17 11:23:28] Log to stdout: No
[2017-02-17 11:23:28] Log format: Normal
[2017-02-17 11:23:28] Configuration file path: C:\Program Files (x86)\VI-grade\VI-Licensing\lmx-serv.conf
[2017-02-17 11:23:28] Serving following features:
[2017-02-17 11:23:28] VI_CarRealtime_IFace (v17.0) (100 license(s)) license type: exclusive
```

## Installation Guide

```
[2017-02-17 11:23:28] VI_Driver_Basic_Core (v17.0) (100 license(s)) license type: exclusive
[2017-02-17 11:23:29] VI_Tire_Core (v17.0) (100 license(s)) license type: exclusive
[2017-02-17 11:23:29]
[2017-02-17 11:23:29] To administrate the license server go to page http://SERVER_PC:6200
[2017-02-17 11:23:29] Ready to serve...
```

The work flow is:

1. license server is initialized
2. the port used for TCP and UDP connection is logged: the port used is *TCP\_LISTEN\_PORT* parameter defined in *lmx-serv.cfg* file in VI-Licensing folder
3. lic file is parsed and all feature keys are read: license file used is the one referenced in *LICENSE\_FILE* parameter defined in *lmx-serv.cfg* file
4. once all keys are read, the server is *Ready to serve...*

**Note:** sometimes the following messages might be found in the server log:

```
[2017-02-17 11:23:25] WARNING: Windows firewall blocks TCP port 6200.
[2017-02-17 11:23:25] WARNING: Communication to license server might not work properly.
[2017-02-17 11:23:25] WARNING: Windows firewall blocks UDP port 6200.
[2017-02-17 11:23:25] WARNING: Automatic server discovery might not work properly.
```

Such messages don't always mean that TCP/UDP block is actually locked by Windows firewall, anyway it's recommended to double check that no antivirus or firewall is preventing data flow from such port: the port is used to exchange data between client and server (also for a node-locked license where server and client are in the same machine).

Whenever something goes wrong during server initialization, a message reporting the error will be written in *lmx-serv.log* file. For details on the error please refer to [X-formation](#) site and send an e-mail to [support@vi-grade.com](mailto:support@vi-grade.com) with *lmx-serv.log* file in attachment.

Same concept for check-out and check-in of a feature key; here a proper check-out/in or a key:

```
[2017-02-17 12:43:54] CHECKOUT by USER@TPW520 [::1]: VI_CarRealtime_IFace
[2017-02-17 17:04:46] CHECKIN by USER@TPW520 [::1]: VI_CarRealtime_IFace
```

If something goes wrong during check-out and/or check-in, the error will be written in *lmx-serv.log*.

## 2. Unable to locate a valid license...

If a panel with a message like "Unable to locate a valid license..." shows up when trying to launch one of VI-grade software, it means that the client application hasn't been able to communicate with the server (the machine where VI-Licensing is installed).

So, after checked that on server side everything has been correctly set up, what needs to be done is to check that *VIGRADE\_LICENSE\_PATH* environment variable is set in client machine and that its value is correctly defined. Please refer to Setting up the License topic for all the details about how to set up the environment variable.

If the problem persists, check that no *LMX\_LICENSE\_PATH* variable exists among environment variable pool, since LM-X looks for it as well as for *VIGRADE\_LICENSE\_PATH* one.

Another factor that might lead to such error is due to boost\_interprocess folder. boost\_interprocess is a hidden folder that is created to store a binary file used to exchange information among different processes protected by the license system. If the first time the software has been executed with administrator rights, such binary file is created with just administrator permission so a standard user can't read it. In general, removing the folder and starting again the software as standard user solves the problem. Please refer to [X-formation Interprocess issue](#) topic for all the details.

Further hints:

- check that communication between client and server machine is ok: from a shell dos type the command `ping <server_machine_name>` and check that the packages are correctly exchanged;
- check that an antivirus or a firewall system isn't blocking the communication through the port used to exchange data (6200 by default).
- launch the software with Administrator rights;
- check if the server machine is using another software which relies on the same LM-X licensing and in case make sure that they're not communicating through the same port. A possible solution for such a problem consists in changing the port used by one of the two applications.

If the problem is still open, set `LMX_EXTENDEDLOG` environment variable in client machine (refer to [X-information Enabling extended logging](#) topic for the details) and send an e-mail to [support@vi-grade.com](mailto:support@vi-grade.com) with the log file generated in attachment.

## 7.4 License Toolkit

License toolkit executable (`license_toolkit.exe`) is shipped in `license` folder of VI-CarRealTime installation directory.

To use it, just open a command prompt (`cmd.exe`) and call the executable with the options described below.

### Available command options:

**-check feature\_name**

check for a specific feature availability.

**-expiration feature\_name**

check for remaining time before the selected feature expires.

**Note:** if the time left is less than 1 hour, it will answer that the feature is already expired, but the user will be able to use that feature for all the minutes left.

**-borrow feature\_name time**

borrow a specific feature for the selected time (time in hours). The license is now deployed on the local machine for the given amount of time (and removed from the server for the same amount of time).

The license is **returned automatically** to the server **after the borrowing time** (while the local machine loose the license).

**Note:** if trying to borrow a feature already borrowed, you'll get an error about an invalid input parameter.

**Note:** If your allowed borrowing time (see licensing specs) is less than required (eg. 10h max instead of 24) or you have not the borrowing enabled the answer will be:

```
>>VI-grade licensing ERROR: feature borrowing failed
>>ERROR = The specified borrow period is too long
>>VI_Bike_Core license borrowing failed
```

**-return feature\_name**

to return a specific feature previously borrowed, before the borrowing expiration time.

**-borrowproduct productname time**

borrow the set of features required by product prodname for the selected time (time in hours). The license is now deployed on the local machine for the given amount of time (and removed from the server for the same amount of time).

---

*Installation Guide*

If the borrowing request has succeeded, the features borrowed will be listed as successful.

The license is **returned automatically** to the server **after the borrowing time** (while the local machine loose the license).

**Note:** If the product name provided is wrong, a list of all available product names will be shown.

**Note:** if trying to borrow a product already borrowed, you'll get an error about an invalid input parameter.

**Note:** If your allowed borrowing time (see licensing specs) is less than required (eg. 10h max instead of 24) or you have not the borrowing enabled the answer will be:

```
>>VI-grade licensing ERROR: feature borrowing failed
>>ERROR = The specified borrow period is too long
>>VI_Bike_Core license borrowing failed
```

**-returnproduct productname**

return the set of features related to product name, before the borrowing expiration time.

**Note:** If the product name provided is wrong, a list of all available product names will be shown.

**-featuresfile file\_name**

definition file listing features for each product.

**Note:** the application is case sensitive, so be sure to type correctly the product or the feature name.

# 8 Release Notes

Welcome to the release notes of VI-CarRealTime 20.0. The chapter contains information regarding new features, known issues and update history.

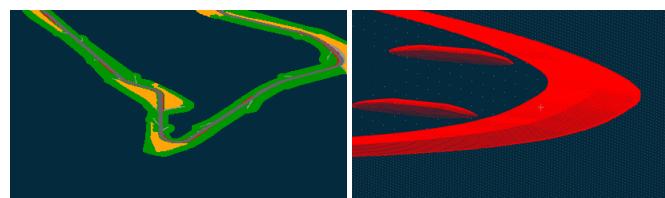
Please send your comments or support requests to [support@vi-grade.com](mailto:support@vi-grade.com).

## 8.1 What's New

### 8.1.1 VI-CarRealTime 20.0

#### Road & Tires:

- The road/tire section of the VI-CarRealTime solver introduces a new data management structure (**CDI**) designed to grant better computational performance.
- A new road model, identified as **GridMesh**, is now included in the VI-Road suite. Such road model is tailored at modelling complete environments with very high surface definition (1 cm) still granting extremely small response time independently from the evaluation position and data resolution. Initialization time is also extremely reduced compared to other road models like the VI-Road - Mesh one. The new model is designed to allow a straight forward connection to scan data. As all other VI-Road variants, the GridMesh grants intellectual property protection through data obfuscation and licensing.



- This new VI-CarRealTime release supports most recent versions of 3rd party tires models:
  - **MF-Tyre/MF-Swift**
    - Support for version 2020.1
    - MF-Swift + Road enveloping + GridMesh is realtime capable
    - Support for National Instrument Hil platform
  - **FTire**
    - Updated interface to version 2020-2
    - Switch to cti loader making FTire installation independent from VI-CarRealTime and VI-DriveSim

- **RIDESuite**
  - Support for RIDE Suite version 1.9 / 2.1
  - Tread wear effects in thermal and dynamic performance
- **CDTire**
  - Direct interface replaced the former user tire integration leading to improved efficiency
- **TameTire**
  - Support for version 6.1

## Modelling enhancements:

- Previous releases, introduced support for multiple configuration of electric drive-train but still relying on external models for the battery component. This VI-CarRealTime release introduces an internal **battery** suitable for different kind of chemistry like:

- Lead-Acid
- Li-Ion
- Ni-Cd
- Ni-Mh

The model parameters are directly accessible in the UI and grant full parametrization of the model response

- Voltage (Constant or Function of SOC)
- Polarization Constant
- Maximum Battery Capacity
- Response Time
- Resistance
- State of Charge
- Exponential Voltage
- Exponential Capacity
- Max Peak Current



- **User sensors** component has been expanded to cover also rotational contribution for displacement, velocity and acceleration.
- **VI-MxMount** component is now supported as bushing type for chassis separated masses:
  - Engine
  - Body on frame

Different modeling variant are supported

- Elastomer - Sum of frequency dependent force component and amplitude dependent force component.
- Hydromount / Hydromount - Two elastomer models (main rubber and chamber rubber) and a damped fluid mass arranged as shown in the scheme.
  
- The trade off between execution speed and modeling accuracy for any VI-CarRealTime model can be adjusted through the new **Variable Model Fidelity** option:



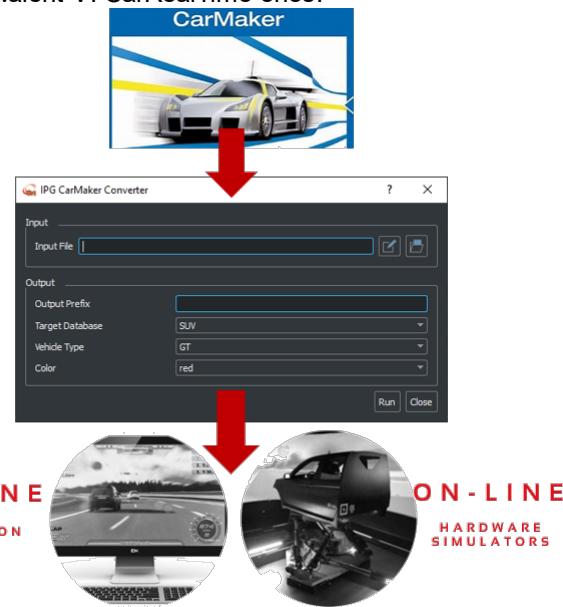
Based on single switch, accessible in the system parameter tree, the user can define which fidelity level to use for a simulation task. The first application of this technology is in the context of the Press Maneuver event: using a less refined, so faster, model for the initial steps of the optimization process, grants an overall reduction of the computation time with results equivalent to the usage of the full model for the entire process.

- The generation of VI-CarRealTime models starting from other vehicle dynamics packages like **CarSim** has been improved in different areas:
  - the import process is now much faster (seconds instead of minutes)
  - in addition to the already supported *par* files, the conversion now can start from the entire model database.
  - Improved data translation for:
    - air spring
    - compliances
    - asymmetric un-sprung mass
    - gear shifting tables
    - differentials inertia
    - clutch
    - aerodynamic



## Release Notes

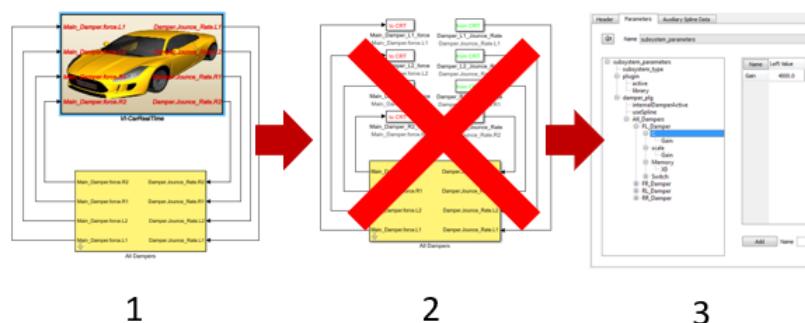
- A new addon module designed to handle **CarMaker** datasets is now available as VI-CarRealTime addon. As for the CarSim Converter, the conversion process is extremely fast and is able to maps the most relevant features of the CarMaker model into the equivalent VI-CarRealTime ones.

**Solver enhancements:**

- The **computational efficiency** of the VI-CarRealTime solver has been improved in many submodules: combined with improvements introduced to the road and tires component, the solution time can be reduced by 20%.
- Kingpin moment** evaluation has been reviewed considering:
  - contribution of force elements acting on the upright
  - secondary contributions like the terms of the wheel inertia tensor perpendicular to the spin axis.
 leading to an even more accurate steering torque feedback computation
- Scaling factor for elements like:
  - dampers
  - antirollbars
  - suspension kinematics and compliance
  - motion ratio
 are now **runtime tunable** in all user scenarios including MATLAB / Simulink and VI-DriveSim.

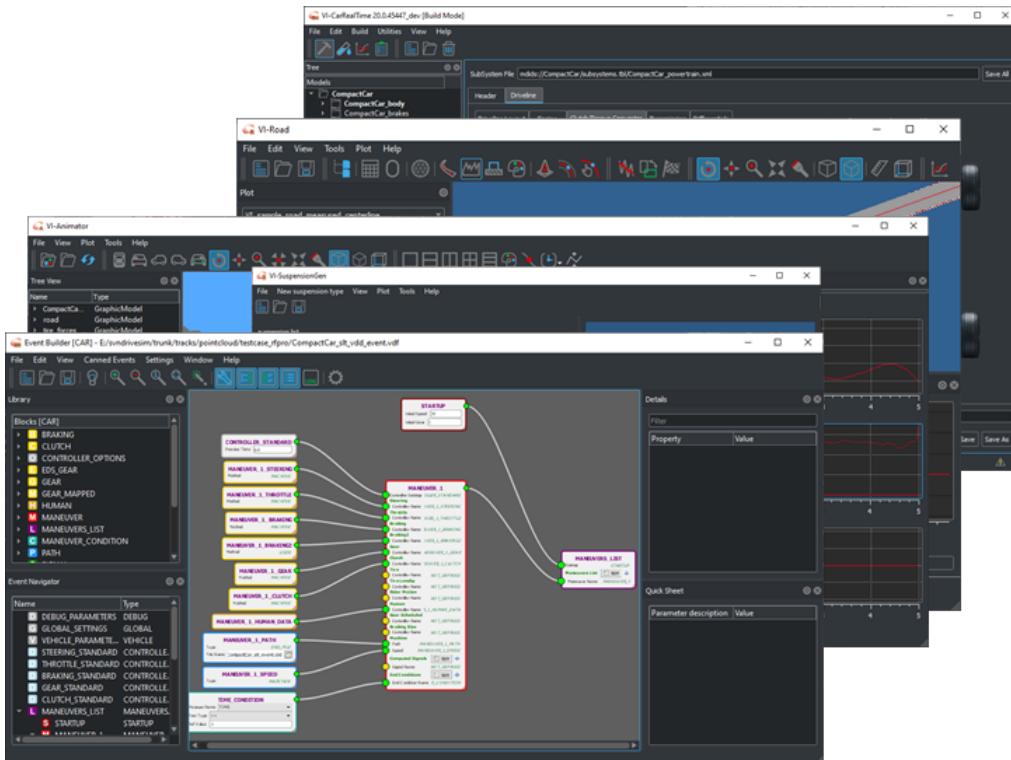
**MATLAB Interface enhancements:**

- The VI-CarRealTime plugins generation from Simulink feature has been redesigned to remove the need of manually creating a specific model based on the interface I/O blocks. With this release, a regular cosimulation model can be directly exported as a solver plugin.
- MATLAB structs are now correctly mapped to the auxiliary subsystem
- Complex models in which the VI-CarRealTime S-function is not at the top level are now supported.
- Support for MinGW compiler



## User Interface enhancements:

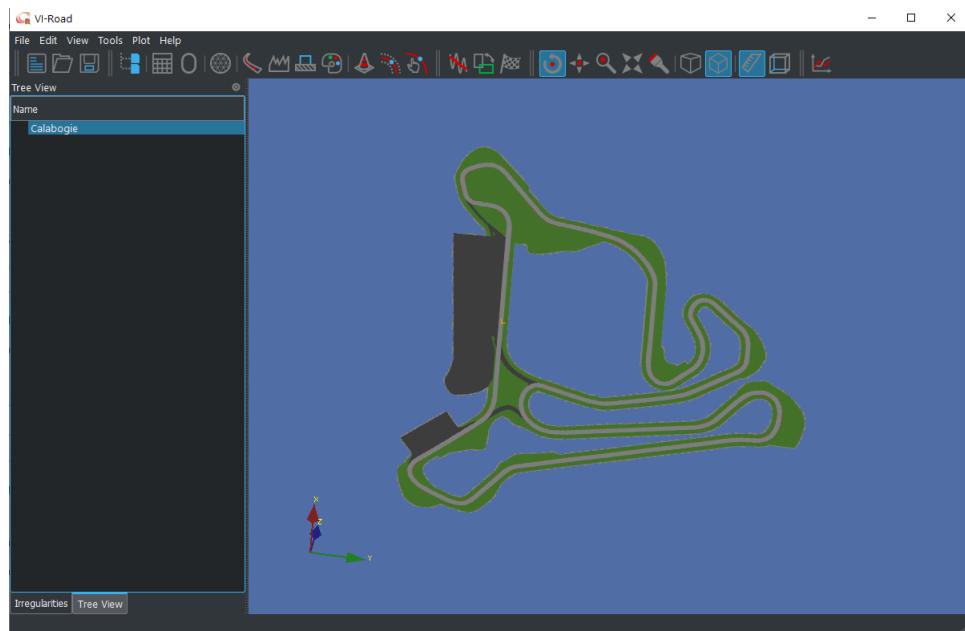
- The complete VI-CarRealTime suite now supports a **dark theme** in addition to the traditional light one both the main UI as well as all ancillary tools like VI-Road, VI-Animator, VI-SuspensionGen, VI-EventBuilder can switch live among the two themes. The background color of all plotting widgets is now customizable and all modules received more consistent icons.



### 8.1.2 What's New in VI-Road

The main focus of VI-Road v20 is the support for the new GridMesh model. On top of the solver support in combination with the most widely used tire models, the VI-Road User interface supports visualization of GridMesh datasets with different resolution levels and all the capability connected to trajectories generation.

## Release Notes



This release introduces also several corrections to the manipulation of openCRG datasets.

Please consult the revision history chapter for the full list of changes.

## 8.2 Licenses

VI-CarRealTime 20.0 requires following set of license keys:

- VI\_CarRealtime\_Core
- VI\_CarRealtime\_IFace
- VI\_Driver\_Basic\_Core
- VI\_Driver\_EventBuilder
- VI\_Road\_Core
- VI\_Road\_Toolkit
- VI\_Tire\_Core
- VI\_Tire\_Toolkit
- VI\_Tire\_TireLimits
- VI\_Animator

The following add-on modules requires a specific license key (red indicates changes respect to previous versions):

- |                                |                                     |
|--------------------------------|-------------------------------------|
| • MaxPerformance               | VI_Driver_Advanced_Core             |
| • PressManeuvers               | VI_CarRealTime_PressManeuvers       |
| • K&C Interface                | VI_CarRealtime_Knc                  |
| • Virtual Test Drive interface | VI_CarRealTime_VirtualTestDrive     |
| • VI-Safety                    | VI_Safety_CRT_IFace                 |
| • Advanced rack-steering model | VI_CarRealTime_AdvancedSteering     |
| • CarSim converter             | VI_CarRealTime_CarSim_Converter     |
| • <b>CarMaker Converter</b>    | VI_CarRealTime_CarMaker_Converter   |
| • FMI Master                   | VI_CarRealtime_FMI                  |
| • MaxPerformance Custom Logics | VI_Driver_Advanced_Optmization_Core |
| • SuspensionGen                | VI_SuspensionGen                    |
| • TameTire Interface           | VI_Tire_TameTire                    |
| • FTire interface              | VI_Tire_FTire                       |
| • MFTyre/MFSwift Interface     | VI_Tire_MFTyre                      |

- **CD Tire** VI\_Tire\_CDTire

VI-CarRealTime can optionally use Adams/PostProcessor, as a postprocessing alternative to VI-Animator. In order to use it the following license key is required in conjunction with the MSC licensing system:

- ADAMS\_Postprocessor

A full **Adams Car** installation is required to utilize VI-CarRealTime plugin. VI-CarRealTime plugin is protected by the license key:

- VI\_CarRealtime\_ADAMS\_IFace

To run Adams Car you need the following keys in conjunction with the MSC licensing system:

- ADAMS\_View
- ADAMS\_Solver
- ADAMS\_Car\_Suspension
- ADAMS\_Vehicle\_Solver
- ADAMS\_Foundation\_Classes
- ADAMS\_TireHandling

Additional (3rd party) license keys are needed for enabling tire models different from VI-Tire.

In order to run simulations with Cosin FTire model, please refer to the documentation stored in the acarrt/cosin subdirectory of the VI-CarRealTime installation.

In order to run simulations with Tass MF-Tyre and MF-Swift tire models please refer to the documentation stored in the acarrt/mftyre\_mfswift or acarrt/mftyre\_mfswift\_v7 subdirectory of the VI-CarRealTime installation.

Please note that starting from v19.0 the FTire and MF-Tyre/MF-Swift interfaces requires a dedicated license.

This product is in part based on incorporated software libraries. Please refer to the [acknowledgments.pdf](#) document, included in the product documentation for a listing of the adopted components and the respective licenses.

## 8.3 3rd Party Compatibility

This table shows the compatibility of the VI-grade suite products with the main 3<sup>rd</sup> party software.

|                             | VI-CarRealTime      | VI-BikeRealTime     | VI-DriveSim          | VI-Driver/VI-Rider<br>for Matlab | VI-Driver<br>for FMI |
|-----------------------------|---------------------|---------------------|----------------------|----------------------------------|----------------------|
| Matlab®                     | from 2015b to 2019b | from 2015b to 2019b | from 2015b to 2019b* | from 2015b to 2019b              |                      |
| SimWorkBench®               | 2018.3<br>2020.1    | 2018.3<br>2020.1    | 2018.3<br>2020.1     |                                  |                      |
| Veristand™ (***)            | 2015sp1             | 2015sp1             |                      |                                  |                      |
| dSPACE® RCP & HIL (**)      | 2018b               | 2018b               |                      | 2018b                            |                      |
| ETAS<br>LABCAR-OPERATOR IP® | 5.4.8               |                     |                      |                                  |                      |
| SCANeR®                     | 1.8r33, 1.9r22      |                     | 1.8r33, 1.9r22       |                                  |                      |
| Prescan®                    | 7.3                 |                     |                      |                                  |                      |
| Virtual Test Drive®         | 1.4                 |                     |                      |                                  |                      |
| SolidThinking Activate      | 2017.1              |                     |                      |                                  |                      |
| Dymola®                     | 2015                |                     |                      |                                  | 2015                 |
| CarSim™                     | 2017.1              |                     |                      |                                  |                      |
| CarMaker™                   | 9.0                 |                     |                      |                                  |                      |
| TameTire                    | 6.1                 |                     | 6.1                  |                                  |                      |
| CDTire                      | 4.2.8               |                     | 4.2.8                |                                  |                      |
| RIDESuite                   | 1.9/2.1             |                     | 1.9/2.1              |                                  |                      |
| FTire                       | 2020.2              |                     | 2020.2               |                                  |                      |

(\*): please refer to SimulationWorkBench documentation for Matlab version compatible with MLToolkit module.

(\*\*): for SCALEXIO targets, only firmware version 4.0.1 is supported.

3<sup>rd</sup> Party Software included in VI-grade products:

|                  | VI-CarRealTime    | VI-BikeRealTime   | VI-DriveSim       | VI-Driver/VI-Rider<br>for Matlab | VI-Driver<br>for FMI |
|------------------|-------------------|-------------------|-------------------|----------------------------------|----------------------|
| MF-Tyre/MF-Swift | 6.2.0.3<br>2020.1 | 6.2.0.3<br>2020.1 | 6.2.0.3<br>2020.1 |                                  |                      |

The following table shows the 3<sup>rd</sup> party compatibility for Adams-based VI-grade product:

|            | VI-Motorcycle | VI-Automotive | VI-Rail | VI-Aircraft | VI-CarRealTime<br>Plug-In | VI-Driver                 |
|------------|---------------|---------------|---------|-------------|---------------------------|---------------------------|
| MSC Adams™ | 2020.0        | 2020.0        | 2019.2  | 2020.0      | 2018.0, 2019.0,<br>2020.0 | 2018.0, 2019.0,<br>2020.0 |
| Matlab®    | *             | *             | *       | *           |                           |                           |

(\*): please refer to Adams documentation for compatibility version.

(\*\*\*) The NI-PXI integration requires Visual C++ 2010 / SDK 7.1 to complete the building procedure successfully. Please refer to the NI-VeriStand documentation for more detail.

The VI-Licensing LMX supported version is **4.9.2** both for Server and for Client.

## 8.4 System Requirements

### Supported Operating Systems

VI-CarRealTime 20.0 is available for the following platforms:

| Platform    | Installer Name            |
|-------------|---------------------------|
| windows x64 | VI_Crt_20_0_x64_Setup.exe |

This installer is compatible with:

- Windows 7 x64
- Windows 10 x64

Please note that this version of VI-CarRealTime is released exclusively for 64 bit OS.

### Hardware Requirements

Minimum hardware capabilities:

- **Processor:** 1.0 gigahertz (GHz) processor
- **RAM:** 2 GB for 64 bit version
- **Hard disk space:** 2.0 GB for full package installation
- **Graphics:** Video card that runs at 1280 x 1024 screen resolution

Recommended hardware capabilities:

- **Processor:** 2.2 gigahertz (GHz) processor
- **RAM:** 4 gigabyte (GB)
- **Hard disk space:** 2.0 GB for full package installation
- **Graphics:** Video card that runs at 1920 x 1080 screen resolution

### Additional Packages

The optional VI-CarRealTime Adams module should be installed separately, based on the desired Adams version:

| Package                | Installer Name                                   | Package Size |
|------------------------|--------------------------------------------------|--------------|
| VI-CarRealTime Plug-in | VI_Crt_plugin_<adams_version>_20_0_x64_Setup.exe | 180 MB       |

Specific overlays are available for supporting the following "hardware in the loop" platforms:

*Release Notes*

| Package                                      | Installer Name                      | Package Size |
|----------------------------------------------|-------------------------------------|--------------|
| dSPACE<br>SCALEXIO r2018b                    | VI_Crt_SCALEXIO_20_0_r18b_Setup.exe | 35 MB        |
| National Instrument<br>Veristand and LabView | VI_Crt_ni_pxi_20_0_x86_Setup.exe    | 107 MB       |
| ETAS LabCar Operator                         | VI_Crt_ETAS_20_0_Setup_x64.exe      | 40 MB        |

**Note:** the Concurrent SimWorkbench is also supported. Please contact VI-grade support to request a specific version of VI-CarRealTime for these environments.

## 8.5 Updating models

Automatic model updates guarantee in most of the cases the forward compatibility of models when switching to newer versions of VI-CarRealTime.

This section describes version specific situations where user actions may be needed to properly use existing models or data files in updated version of the software.

- [Updating to version 20](#)
- [Updating to version 19](#)
- [Updating to version 18](#)

### 8.5.1 Updating to version 20

No specific model updates are required from v19 to v20.

### 8.5.2 Updating to version 19

When updating a powertrain subsystem from v17 to v18, C1 spline map for LSD differential could be created without units for inner torque independent data (assuming it as Newton-meter).

v19 automatically adds the units if missing. You need to update C1 spline map independent data in order to match v18 results.

### 8.5.3 Updating to version 18

VI-CarRealTime v18 powertrain model takes into account the engine inertia even when the clutch model is not active: if your v17 declared an engine inertia with clutch turned off, you will need to set the engine inertia to 0 in v18 in order to match v17 results.

In order to fully support the features implemented in VI-CarRealTime vehicle model during export procedure, VI-CarRealTime plugin for Adams Car may require that Adams Car model templates are updated by introducing specific entities (communicators and/or variables).

Please refer to the vehicle parameter extraction procedure and suspension sequence of analyses documentation for a detailed description of the supported entities.

## 8.6 Changed Behaviour

### 8.6.1 Version 20

#### VI-CarRealTime

##### Vehicle Model

Front suspension scaling factors are applied only to jounce dependent curve (e.g. applying a scaling factor of 2 to the toe curve vs. jounce vs. steer doesn't double the steering ratio, but only the variation of toe with jounce)

## Investigation Mode

The name of the CSV summary file generated at the end of an investigation, has been modified to support investigations with multiple events.

It now contains a suffix with the name of the investigation events.

## Custom Events

With release 20.0 VI-CarRealTime introduces a new Python interpreter. Custom events generated for earlier version of the product must be updated in order to be used with this release.

- Syntax changes due to Python 2.x to Python 3.x update

Some of the changes are related to Python syntax modifications. In most of the cases the needed changes will be limited to a few common functions and keywords. For sure the most common error is due to the change in the Python print statement

The following statement is valid in python 2.x but NOT in Python 3.x :

```
print 'hello world!'
```

The following statement is instead valid in both Python 3.x and in many recent version of python 2, including the one available in earlier VI-CarRealTime versions.

```
print ('hello world!')
```

For any other issue coming from Python 2 to 3 conversion we suggest to refer to one of the many conversion guides available in internet.

- Syntax changes due to VI-CarRealTime Framework update

A few changes are related to modifications in VI-CarRealTime framework. For example in the definition of the event Class, the type of member variables was assigned with specific types, as StringType, IntType, FloatType. These are not supported anymore, but there is a simple workaround that allows to keep using them in VI-CarRealTime 20, maintaining compatibility with earlier versions.

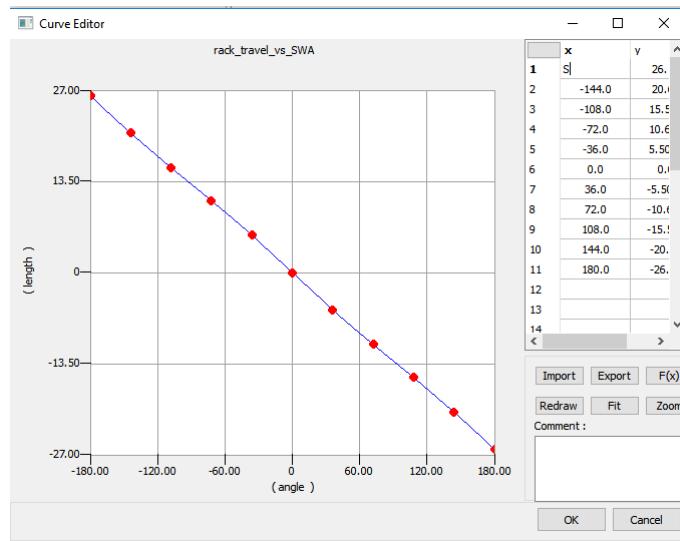
It is enough to add the following code snippet at the beginning of the file containing the class definition:

```
import sys
if sys.version_info[0] > 2:
 ClassType = type
 ListType = list
 TupleType = tuple
 StringType = str
 FloatType = float
 IntType = int
```

## 8.6.2 Version 19

### VI-CarRealTime

With v19, VI-CarRealTime has full support for different rack position layouts. During the export procedure, VI-CarRealTime plugin for Adams Car does not perform any sign adjustment for rack travel vs steering wheel angle spline. Spline will result monotonically decreasing if original model has a rack located backward with respect to kingpin axis.



Depending on rack position, differences on output channels for rack displacement and feedback may appear. It is possible to restore previous behavior by setting to 1 the environment variable `VICRT_PLG_STEER_LEGACY` before running the export procedure.

### KnC Wizard

In v19.1 the export of the compliance from aiding and opposing load cases is performed for the left and right side independently, unlike in v19.0 where the computation was done only for the in phase side and the other side was created symmetric.

It is possible to restore the symmetry of the exported model by setting the symmetry option in the panel of the KnC Wizard.

### VI-SpeedGen

In previous VI-CarRealTime releases the vehicle mass geometry used to initialize speedgen model represented a snapshot of design configuration. Starting from version 19.0 the default configuration represents a snapshot of vehicle configuration after static/setup. This different mass configuration can lead to differences on results of speedgen and maxperformance events. Previous version behavior can be obtained by using the system tree parameter `speedgen->use_design_configuration=1`.

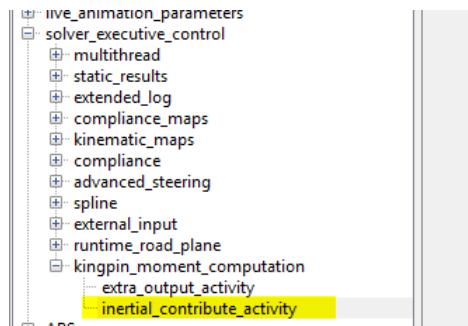
## 8.6.3 Version 18

### VI-CarRealTime

With the introduction of the automatic transmission models, a new output channel (`transmission.gear`) has been added to report the currently engaged gear. With v18 this new channel is sent to VI-Animator in live animation mode instead of channel `driver_demands.gear`, sent in earlier versions. If you have created VI-Animator plot configurations or custom widgets associated to the old channel, please edit them to pick up the `transmission.gear` result component.

## Solver

in v18.2 a new element in the parameter tree has been introduced to allow per model selection of the kingpin moment computation mode.



In previous v18 releases, the selection was possible using an environment variable (VI\_KINGPIN\_INERTIA\_OFF). The default behavior is to include all supported contribution in the kingpin moment computation (1). The environment variable is still supported, but is considered deprecated and will removed in future version

In v18.2, the gear-shifting maps function of throttle are interpreted as depending on the raw throttle input: in previous versions, the effective throttle demand (including the contribution of the engine management system and slip controllers) was used.

## Solver SDK

In v18.2 the behavior of the vehicle relocation C APIs has been modified in order to fix an inconsistent behavior to specified inputs:

- 1) `c_crt_set_sm_XY_position` now forces the estimation of vehicle elevation
- 2) `c_crt_set_sm_position` now relies on elevation provided in input instead of attempting an automatic estimation

if you developed code using one of these APIs you will likely obtain different responses: in order to get the desired behavior, please update your code in order to call `c_crt_set_sm_position` if you want to provide the vehicle initial elevation or `c_crt_set_sm_XY_position` in case you want VI-CarRealTime to compute the elevation.

## VI-Driver

Thanks to some correction applied to reference engine torque input in standstill conditions, VI-CarRealTime will produce smoother throttle and brake signals when vehicle is idling.

## VI-Road

```
the following key in the VDF file is now affecting also open loop throttle signal
while in past version only machine mode was affected.
THROTTLE_CONTROL_ACTIVATION = 'TRUE'
```

```
the consequence is that during a gearshift, the throttle will be released also
when configured in open loop mode.
```

## 8.7 Known Issues

The following limitations have been identified at release time:

- Using silent mode during installation will not display any message in the command window.
- Cross weight may lead to static analysis failures in models including live axle rear suspension.

---

*Release Notes*

- Simulink analyses with huge number of integration steps can cause memory problems (buffer overrun) due to the size of MATLAB workspace output. An immediate workaround is to reduce the number of outputs dumped to MATLAB workspace.
- Simulation statistics and output file formats different from \*.res are only available with active buffer output option (Edit->Preferences...).
- The NI-PXI overlay requires Visual C++ 2010 to complete the building procedure successfully. Please refer to the NI-VeriStand documentation for more details.
- When loading data from a setup file the values stored in components property files (springs, dampers, etc.) are not updated.
- The target speed profile, generated by a VI-MaxPerformance simulation, can be referenced only through a drd file. The parameter System->Properties-->Output Files-->VDF is not supported in case of VI-MaxPerformance event.
- When exporting an MSC Adams/Car model for VI-CarRealTime, Bushings-in-series and Tolerance features of Bushings are not captured by the export and VI-CarRealTime model.
- In the Automatic Model Validation report, for asymmetric vehicles, the curves of track and wheelbase variation versus wheel travel may present an offset with respect to the equivalent Adams curves. Such offset is a constant quantity, added to the track/wheelbase variation curves, which allows VI-CarRealTime suspension model to take into account of wheel centers shifts with respect to the vehicle mid-plane.
- Executing a StaticLapTime or MaxPerformance event using a model with TassInternational MF-Tyre 6.2 tires, several errors are reported in the console: -- ERROR -- TNO TIRE Error opening road data file "DEFAULT\_VITIRE\_FLAT\_ROAD". Such error is misleading because the core is actually using the right road: flat road (default) or the road selected by the user if Use Road Data File checkbox is activated. So such error can be neglected.
- Investigation run with Mode of Simulation set to *matlab\_simulink* will throw errors when computing responses due to a compatibility problem of numpy library with MATLAB.
- Running an investigation which uses a FileDriven event with a VDF generated in a previous MaxPerformance event generates errors due to the fact that VI-Driver can't read/load path DRD file . It happens because DRD file is referenced in the VDF with a relative path: in general DRD file is stored in VI-CarRealTime working directory, while the investigation analyses are run in a specific investigation folder. In general, errors are thrown whenever the investigation uses property files which reference other files with a relative path. Suggested workarounds to overcome the problem:
  - copy all the involved files in the investigation folder;
  - use current VI-CarRealTime directory as investigation folder by simply specifying "." as Output Folder in the Investigation Mode.
- Running a model with the FMI master plugin from a OneDrive directory may lead to failure on the 2nd run. A workaround to prevent the failure is removing the temporary directory in which the FMU files are decompressed.
- If you have Fingerprints created with versions older than VI-CarRealTime 18 and containing Press Maneuver events, you will have to manually update the fields 'Front To Origin Distance' and 'Rear to Origin Distance'. (as an alternative the Fingerprints can be opened in any version between 18.0 and 19.1, and they will be automatically updated).

## 8.8 Revision History

### 8.8.1 Release 20.0

#### Added Capabilities:

| Change ID | Module | Description |
|-----------|--------|-------------|
|-----------|--------|-------------|

|       |               |                                                                                  |
|-------|---------------|----------------------------------------------------------------------------------|
| 18481 | Driver        | Enhance VI-Driver and VI-Road compatibility with MB-SHARC                        |
| 18389 | CarRealTime   | Add documentation on Camber/SVA splines handling with dependent suspension       |
| 18288 | CarRealTime   | Tire Model and Road Format Compatibility Matrix                                  |
| 18275 | SuspensionGen | Remove steering system from rear SolidAxle5Links                                 |
| 18234 | CarRealTime   | Add plugin documentation about ARB deactivation                                  |
| 18075 | CarRealTime   | Remove driver parameters no longer present in the system tree from documentation |
| 17880 | CarRealTime   | Powertrain Layout Enhancements                                                   |
| 17595 | CarRealTime   | Improve car pictures in body panel                                               |
| 17438 | CarRealTime   | Add user VDF option for automatic validation                                     |
| 17418 | CarRealTime   | Add makefile to the pack & go archive                                            |
| 17416 | CarRealTime   | Steering kinematics from vertical motion with fixed steer loadcase               |
| 17381 | CarRealTime   | 3d spline support for Fz compliance                                              |
| 17329 | Road          | Show material ID on Materials frame                                              |
| 17315 | CarRealTime   | Add steering torque input channel                                                |
| 16866 | CarRealTime   | Documenting use_jounce_actuator flag in static loadcase xml                      |
| 15258 | CarRealTime   | Support MFTyre 2020.x on NI_PXI                                                  |
| 15226 | Animator      | Update Icon Design                                                               |
| 15082 | CarRealTime   | Add documentation on methods adopted to export antiroll force                    |
| 15075 | Road          | Remove initial z offset in crg exported from mesh                                |
| 14987 | CarRealTime   | Enhance Preferences Editor                                                       |
| 14722 | CarRealTime   | Support for Adams 2020                                                           |
| 14673 | CarRealTime   | Curve Manager Enhancements                                                       |
| 14669 | CarRealTime   | Make About Dialog consistent with the other products                             |
| 14552 | CarRealTime   | Support FTire 2020.2                                                             |
| 14356 | CarRealTime   | Update Bibliografy in xml file                                                   |
| 14343 | CarRealTime   | Improve FTire usability with VI-Road                                             |
| 14243 | CarRealTime   | Clone models in build mode                                                       |
| 14225 | CarRealTime   | Replace model in entire fingerprint                                              |
| 14059 | CarRealTime   | Support-m module-name when calling python                                        |
| 14018 | CarRealTime   | Improve tire low speed stability with no changes to the property file            |
| 13983 | CarRealTime   | Set default values for ServoGear in SharedModels Rack&Pinion steering            |
| 13932 | CarRealTime   | Update documentation advance steering                                            |
| 13812 | Road          | Background color selection for 3D widget                                         |
| 13350 | CarRealTime   | Add rotational sensors                                                           |
| 13288 | CarRealTime   | Export track damping from Adams Car                                              |
| 13079 | CarRealTime   | Add documentation for missing system parameters                                  |
| 12627 | CarRealTime   | Add flag to ignore braking phase in Fuel Consumption Calculation                 |
| 12595 | CarRealTime   | Add secondary contributes to kingpin moment computation                          |
| 12511 | CarRealTime   | Support body lock option for 7post event                                         |
| 11983 | CarRealTime   | Update CarSim converter                                                          |
| 11976 | CarRealTime   | Certify matlab 2019b                                                             |
| 11344 | CarRealTime   | Create Seven Postrig Tutorial                                                    |
| 11264 | CarRealTime   | Dark Theme                                                                       |

## Release Notes

|       |               |                                                                                       |
|-------|---------------|---------------------------------------------------------------------------------------|
| 11243 | CarRealTime   | King Pin Moment from element acting on upright                                        |
| 11102 | CarRealTime   | MFTyre 2020.1                                                                         |
| 11101 | CarRealTime   | Tametire 6.1                                                                          |
| 10701 | CarRealTime   | Improve correlation for Automatic model validation                                    |
| 10700 | CarRealTime   | Investigation mode improvements                                                       |
| 10408 | CarRealTime   | Automatic deactivation of installation stiffness when stiffness value is non positive |
| 10380 | CarRealTime   | Improve documentation for external driveline with Simulink                            |
| 10353 | CarRealTime   | Optimize solver performance                                                           |
| 10168 | CarRealTime   | Built-in battery model                                                                |
| 9838  | CarRealTime   | CarMaker to VI-CarRealTime converter                                                  |
| 9704  | Road          | GridMesh road model                                                                   |
| 8716  | CarRealTime   | Remove FTire libraries from distribution                                              |
| 8628  | CarRealTime   | Simulink Plugin Export - Enhancements                                                 |
| 7911  | CarRealTime   | Increase model fidelity for ride events                                               |
| 6291  | CarRealTime   | Symmetry control for compliance in K&C Wizard                                         |
| 6092  | CarRealTime   | Direct interface for CD Tire                                                          |
| 5632  | CarRealTime   | Runtime tunable vehicle properties                                                    |
| 4667  | CarRealTime   | Variable fidelity model                                                               |
| 3447  | CarRealTime   | Support for Adams Car CG point location                                               |
| 1309  | SuspensionGen | Export anti-roll bar in dual table format                                             |

**Bugs Corrected:**

| Change ID | Module      | Description                                                                            |
|-----------|-------------|----------------------------------------------------------------------------------------|
| 19046     | CarRealTime | Wrong longitudinal force sign in validation plots                                      |
| 18833     | CarRealTime | Export for body_parts group with flexible bodies fails in force creation               |
| 18736     | Road        | Unwanted icons on default context menu                                                 |
| 18714     | CarRealTime | Hardy Disk Affects torsion bar load output                                             |
| 18616     | CarRealTime | Press maneuver events terminates in case on equation of motion violation               |
| 18573     | Road        | Colors mismatch between materials frame and graphics                                   |
| 18557     | CarRealTime | Path Sensor outputs may contain random data in case of automatic initialization fails. |
| 18411     | CarRealTime | No Differentials options does not deactivate all differentials                         |
| 18382     | CarRealTime | Shared models advanced steering inertia too low                                        |
| 18369     | CarRealTime | Copy and paste does not work on standard tables                                        |
| 18365     | CarRealTime | Incorrect title in several message dialogs                                             |
| 18363     | CarRealTime | Wrong pwr map file exported in obfuscated events                                       |
| 18362     | CarRealTime | VI-CarRealTime Plugin number of compliance analysis are not correctly assigned         |
| 18354     | CarRealTime | Kinematic scaling factor should not affect steering dependency                         |
| 18338     | CarRealTime | SpeedGen can't be launched from command                                                |
| 18328     | CarRealTime | Plugin Export only checks wheel subsystem symmetry to write wheels subsystems          |
| 18213     | CarRealTime | Plugin Export add driveline adds wrong minor role to assembly                          |
| 18012     | CarRealTime | Traceback using maxperf + staticevo when using solver ppt                              |
| 17875     | Road        | Crg road exported from mesh road is flat                                               |

|       |               |                                                                                     |
|-------|---------------|-------------------------------------------------------------------------------------|
| 17791 | CarRealTime   | Investigation Mode Summary Report for multiple events                               |
| 17647 | CarRealTime   | Pickup model event fails to run press maneuver event                                |
| 17586 | Road          | Road outside the viewport                                                           |
| 17528 | CarRealTime   | Throttle Scaling in Cosimulation for electric motor not working                     |
| 17463 | CarRealTime   | Components kingpin moment mismatch (spline vs. no spline)                           |
| 17417 | CarRealTime   | Adams plugin Xdof export does not support variants                                  |
| 17380 | Road          | Different results in corner cutting from drd file or drd in rdf                     |
| 17234 | CarRealTime   | Investigation mode: traceback selecting points as factors                           |
| 17046 | CarRealTime   | Wrong units for influence matrix user output                                        |
| 17032 | CarRealTime   | Plugin validation with adams 2019 uses wrong steering units                         |
| 16867 | CarRealTime   | Remove dots ('.') from VI-CRT user output internal name                             |
| 16029 | Driver        | Combo box list not fully visible                                                    |
| 15267 | Road          | Wrong parameter value in exported crg file                                          |
| 15247 | CarRealTime   | Error in mirroring compliance symmetry                                              |
| 15073 | Road          | Crg rototranslation is not correct                                                  |
| 14923 | Road          | Opencrg generated by mesh is wrong                                                  |
| 14918 | Road          | Continuation line flag lost on save                                                 |
| 14870 | CarRealTime   | Investigation mode candidate and response channel search filters are case sensitive |
| 14858 | CarRealTime   | Speedgen vertical acceleration is always zero                                       |
| 14235 | CarRealTime   | Errors when exporting powertrain that doesn't have phs_powertrain_type              |
| 14210 | CarRealTime   | Export errors when phs_kinematic_flag doesn't exist for a subsystem                 |
| 14167 | CarRealTime   | CarRealTime output.jounce_stop_data.jounce_rear is always 0                         |
| 14144 | CarRealTime   | Events table is useless in review mode                                              |
| 14101 | Road          | No way to open pdf help from VI-Road gui                                            |
| 14100 | SuspensionGen | No way to open pdf help from SuspensionGen gui                                      |
| 14083 | CarRealTime   | User sensor panel is not refreshed in duplicated subsystems                         |
| 14078 | CarRealTime   | Incorrect location for user sensor on unsprung mass parts                           |
| 13894 | CarRealTime   | Shared database 'Pickup' has wrong upshift RPM map                                  |
| 13870 | CarRealTime   | Remove compl_parallel_travel simulation from export analyses                        |
| 13307 | CarRealTime   | CRT demo automatic transmission spline                                              |
| 13179 | CarRealTime   | Plugin Validation Wrong CM displacement request                                     |
| 13098 | CarRealTime   | Steering Rack Force Output Differences                                              |
| 13084 | CarRealTime   | Sevenpostrig output documentation                                                   |
| 13045 | CarRealTime   | Export error with kinematic_flag                                                    |
| 12738 | CarRealTime   | Ouput map doesn't deactivate Advanced Steering Ouputs                               |
| 12673 | CarRealTime   | Transmission efficiency spline documentation                                        |
| 12566 | CarRealTime   | CRT does not execute more than one custom postprocessing script                     |
| 12560 | CarRealTime   | Lap Sensor is triggered before end of DRD path                                      |
| 11902 | CarRealTime   | Skidplate Output Naming Convention                                                  |
| 11596 | CarRealTime   | Adams ppt command not working if path contains spaces                               |
| 11252 | CarRealTime   | FMU failing under OneDrive                                                          |
| 10752 | CarRealTime   | Sedan car steering torque sign inversion around 450 deg                             |
| 9630  | CarRealTime   | Adams Crossover shared model does not perform SPMM compliance analyses              |

*Release Notes*

|      |               |                                                                                     |
|------|---------------|-------------------------------------------------------------------------------------|
| 9583 | CarRealTime   | Active runtime log in Solver Executive Control produce an empty log                 |
| 9552 | SuspensionGen | Anti-roll steering component missing sign                                           |
| 8431 | CarRealTime   | Export ADAMS to CRT - Steering Rack Analysis / Scale Factors                        |
| 8416 | CarRealTime   | VI-CRT Plugin: plugin does not recognise steering compliance communicators activity |
| 7905 | CarRealTime   | Advance Steering variable ratio - Check                                             |
| 4462 | CarRealTime   | UNC paths cause errors in CRT                                                       |
| 3269 | CarRealTime   | Support export of rear wheel steering models                                        |

**8.8.2 Release 19.2****Added Capabilities:**

| Change ID | Module      | Description                                           |
|-----------|-------------|-------------------------------------------------------|
| 14889     | CarRealTime | Outputs for rigid part documentation                  |
| 14540     | Road        | Improve road graphic visualization                    |
| 14138     | CarRealTime | Kinematic mode for steering assist servo gear         |
| 13941     | CarRealTime | Support Adams Car Bumpstop of type external on export |
| 13936     | CarRealTime | Support for license protected obfuscated send files   |

**Bugs Corrected:**

| Change ID | Module      | Description                                                                           |
|-----------|-------------|---------------------------------------------------------------------------------------|
| 15274     | CarRealTime | Torque scaling parameter does not work for motors when using pwr file                 |
| 14888     | CarRealTime | Wrong speed tracking using electric motors at differential                            |
| 14843     | CarRealTime | Body on Frame 7Post Rig Analysis Errors                                               |
| 14827     | CarRealTime | Wrong compliance scaling factor for traction and braking                              |
| 14556     | CarRealTime | Fix wrong formula in frequency bushing documentation                                  |
| 14024     | CarRealTime | VI-CarRealTime plugin does not update already open assemblies                         |
| 13929     | CarRealTime | Automatic Gearbox and Standard clutch configuration generates wrong results           |
| 13777     | CarRealTime | Torque Converter Locking Clutch does not generate constant value (=1) for speed ratio |
| 13720     | CarRealTime | Clutch Slip on MaxPerformance w/Dual Clutch                                           |
| 13679     | CarRealTime | Roll center height computation during turn                                            |
| 13214     | CarRealTime | Potential crash when calling vicrtCreateOstruct                                       |
| 13201     | CarRealTime | Advanced Steering Hardy Disk w/EPS Column                                             |

**8.8.3 Release 19.1****Added Capabilities:**

| Change ID | Module      | Description                                                                             |
|-----------|-------------|-----------------------------------------------------------------------------------------|
| 12921     | CarRealTime | Vtd event - xod path always include local directory - not working wth local path option |
| 12431     | CarRealTime | Add independent integration time step to be used for static analysis only               |
| 11139     | CarRealTime | Add reverse rotation input for electric motor                                           |
| 11065     | CarRealTime | Convert capacity factor to correct units during export                                  |

|       |               |                                                                           |
|-------|---------------|---------------------------------------------------------------------------|
| 11053 | CarRealTime   | Support for CDTire 4.2.8                                                  |
| 10893 | CarRealTime   | Customized cones position in Press Maneuvers event                        |
| 10787 | CarRealTime   | Rack and pinion steering documentation improvements                       |
| 10710 | CarRealTime   | Support dSPACE toolchain 2018b                                            |
| 10699 | Driver        | Support Adams 2019.0                                                      |
| 10629 | CarRealTime   | Support Adams 2019.0                                                      |
| 10622 | CarRealTime   | Allow engine block as differential shaft inner reaction body              |
| 10614 | SuspensionGen | Animation support                                                         |
| 9867  | CarRealTime   | Document Adams Car steering template for rack pinion steering             |
| 9584  | CarRealTime   | Optimize solver performance                                               |
| 9033  | CarRealTime   | Export Bushing Scaling from Adams Car                                     |
| 8988  | CarRealTime   | EMotorTorqueMapFile has no corresponding object in parameters             |
| 8916  | CarRealTime   | Need an option to remove suspension analyses in full vehicle export panel |
| 8846  | CarRealTime   | VI-SpeedGen should account for driveline efficiencies                     |
| 8705  | CarRealTime   | Remove obsolete trailer.mdl from shipped example models                   |
| 8513  | CarRealTime   | Make tir reading case insensitive in CarSim Converter                     |
| 8497  | CarRealTime   | Export tire inertia parameters from CarSim                                |
| 8496  | CarRealTime   | Export torque converter parameters from CarSim                            |
| 8322  | CarRealTime   | Support version 5.4.8 of ETAS LCO                                         |
| 8226  | CarRealTime   | Generate CMD with export settings when exporting a model from Adams Car   |
| 8196  | Road          | Implement function to invert trajectories driving direction               |
| 7940  | CarRealTime   | Show message at the end of suspension export                              |
| 7912  | CarRealTime   | Investigation mode improvements                                           |
| 7877  | CarRealTime   | Add Heave and Roll Damper Inputs                                          |
| 5370  | CarRealTime   | Update OpenCRG to version 1.1.2                                           |
| 3507  | CarRealTime   | Downshifting map with tunable sport/comfort modes                         |
| 3152  | CarRealTime   | Remove Sensors (or bushings) from GUI tables                              |
| 1310  | SuspensionGen | Allow hardpoint selection from data row and viceversa                     |
| 1250  | CarRealTime   | Improve correlation for Automatic model validation                        |

## Bugs Corrected:

| Change ID | Module      | Description                                                              |
|-----------|-------------|--------------------------------------------------------------------------|
| 13047     | Driver      | Simulation not interrupted by end of path event                          |
| 12667     | CarRealTime | VI-SpeedGen Evo doesn't write all the outputs channels                   |
| 12628     | CarRealTime | VI-CRT Chassis Compliance and Damping in v18.2 versus v19.1              |
| 12567     | CarRealTime | API publishSystemFileToDb wrong cfg output                               |
| 12555     | CarRealTime | Rigid Part graphic not copied when publishing a system in a new database |
| 12391     | CarRealTime | User sensor on body part not working                                     |
| 12012     | CarRealTime | Body Rigid Parts are not considered in automatic model validation report |
| 12007     | CarRealTime | Full xml paths when exporting model with Use Existing option             |
| 11921     | CarRealTime | Skidplate point appear and disappear from Skidplate table                |
| 11629     | CarRealTime | Missing kingpin_moment_computation parameters update                     |

## Release Notes

|       |               |                                                                                           |
|-------|---------------|-------------------------------------------------------------------------------------------|
| 11613 | CarRealTime   | Investigation Mode Not Working                                                            |
| 11531 | CarRealTime   | Vehicle Wizard Utils generate wrong gear shifting map                                     |
| 11528 | Driver        | Adams controls simulation with vidriver failure                                           |
| 11506 | CarRealTime   | Wrong Label in RackFeedback Force panel                                                   |
| 11464 | CarRealTime   | Vicrt_startup.py can't include external libs                                              |
| 11422 | CarRealTime   | Traceback in compute combinations with groups (1 element)                                 |
| 11365 | CarRealTime   | Single bumpstop clearance adjustment                                                      |
| 11281 | CarRealTime   | Engine property files obfuscation works only in combination with Overwrite Files          |
| 11239 | CarRealTime   | Incorrect options reported in customization tutorial                                      |
| 10951 | SuspensionGen | SuspensionGen closes writing on a folder where the user has no writing access             |
| 10922 | CarRealTime   | VI-TIRE user tire module skipping the differencing computations in single-thread          |
| 10726 | CarRealTime   | Errors when setting ARB rate using MATLAB api                                             |
| 10597 | CarRealTime   | Wrong offset of steering curves for dependent suspension                                  |
| 10594 | CarRealTime   | Auxiliary Rebound Bumper causes Simulink crash                                            |
| 10580 | CarRealTime   | Simplified main motor does not work                                                       |
| 10414 | CarRealTime   | Crossover Torque Converter Slips                                                          |
| 10398 | Driver        | Adams crash submitting VDF                                                                |
| 10366 | CarRealTime   | No drag force in pickup shared model                                                      |
| 10327 | CarRealTime   | Wrong powertrain and graphics file names during validation                                |
| 10214 | Road          | Crash creating a centerline with path builder tool                                        |
| 10207 | CarRealTime   | Body Compliance: torsion compliance not tunable                                           |
| 10186 | CarRealTime   | CRT Demo external steering model example is out of date                                   |
| 10129 | CarRealTime   | Automatic Model Validation fails when vicrt_19_defaultPrefs.xml is defined in Adams CWD   |
| 10115 | CarRealTime   | Error exporting powertrain data                                                           |
| 10027 | CarRealTime   | Sport/Comfort Gear shifting mode live tuning is not working on simulator                  |
| 9983  | SuspensionGen | Caster angle not computed by mcpherson elastokinematics analysis                          |
| 9787  | CarRealTime   | Sensors appear and disappear from User Sensor table                                       |
| 9785  | CarRealTime   | Steering Wheel Angle end condition is not triggered when the model use rack travel spline |
| 9726  | CarRealTime   | Wrong automatic selection of initial gear                                                 |
| 9636  | CarRealTime   | VI-Speedgen computes wrong speed profile when using electric motors                       |
| 9623  | CarRealTime   | Matlab may crash running a VI-CarRealtime model including a plugin generated by Simulink  |
| 9574  | CarRealTime   | Crt_pluginSOL not included in installer                                                   |
| 9536  | CarRealTime   | Wrong bushing force computation when spline is not symmetric wrt deformation              |
| 9062  | CarRealTime   | Wrong driving torque reactions                                                            |
| 8901  | CarRealTime   | Help: links to anchors in the same page do not work for Investigation mode topic          |
| 8898  | CarRealTime   | Body on frame feature is not compatible with 7 Post events                                |
| 8892  | CarRealTime   | Motor uses property file activation flag even in case of simple model                     |
| 8857  | CarRealTime   | Solver crash when using obfuscated top motor map                                          |
| 8855  | CarRealTime   | Differential efficiency does not change behaviour during throttle release                 |
| 8803  | CarRealTime   | Missing units in AntirollBar Compression Ratio value                                      |
| 8782  | CarRealTime   | Wrong signs in some automatic model validation KnC curves with MBSharc                    |
| 8773  | CarRealTime   | Wrong steering torque sign when using rack-pinion steering                                |
| 8751  | CarRealTime   | Comfort mode shifting tables not set correctly for vehicles exported from Adams           |

|      |             |                                                                                 |
|------|-------------|---------------------------------------------------------------------------------|
| 8739 | CarRealTime | Clutch efficiency is not applied correctly for standard clutch                  |
| 8709 | CarRealTime | Fz Compliance macro errors when exporting MBSharc Model                         |
| 8694 | CarRealTime | Simulink cosimulation crashes when using a send file including trailer model    |
| 8607 | CarRealTime | Traceback when compiling interactively custom solver plug-in                    |
| 8542 | CarRealTime | Carsim converter does not export splines vs Rack                                |
| 8489 | CarRealTime | User defined vehicle location generates a speed gen event not coherent          |
| 8451 | CarRealTime | Solver crash when using v18 automatic transmission                              |
| 8407 | CarRealTime | Wrong documentation on groups required to export engine rods from Adams Car     |
| 8370 | CarRealTime | Main motor torque imported from v18.2 delivers different results                |
| 8338 | CarRealTime | Missing dialog boxes in VI-CarRealTime plug-in for Adams Car older than 2017.2  |
| 8317 | CarRealTime | Export procedure raises errors exporting solid geometry                         |
| 8281 | CarRealTime | Deactivation Runtime input does not work correctly when using dual clutch model |
| 8274 | CarRealTime | Remove "variants" stuff from documentation                                      |
| 8262 | CarRealTime | Missing export_def.py in VI-CarRealTime installation                            |
| 8224 | CarRealTime | Asymmetry issue with rear Installation Stiffness                                |
| 8214 | CarRealTime | Engine Initial Condition may be incorrect                                       |
| 8210 | CarRealTime | Dual motor awd does not update motors and differentials activity                |
| 8195 | CarRealTime | Matlab plugin export failure using expression in vector                         |
| 7729 | CarRealTime | Driveline layout visualization                                                  |
| 4957 | CarRealTime | Investigation mode System Mismatch on empty fingerprint                         |
| 3886 | Driver      | File -> Import Event menu in VI-EventBuilder doesn't work                       |

## 8.8.4 Release 19.0

### Added Capabilities:

| Change ID | Module      | Description                                                                    |
|-----------|-------------|--------------------------------------------------------------------------------|
| 7947      | CarRealTime | Export Advanced steering data from Adams Car                                   |
| 7515      | CarRealTime | Remove need of memory block for simulink interface                             |
| 7198      | CarRealTime | Add compliance threshold to model system tree                                  |
| 6257      | CarRealTime | Exposing Custom Logics documentation                                           |
| 6018      | Animator    | Record movies with full frame rate                                             |
| 5978      | CarRealTime | Export models from Adams for v19                                               |
| 5925      | CarRealTime | Add angle sign picture to susp setup date editor                               |
| 5819      | CarRealTime | Use communicator matching name to detect steering compliance information       |
| 5810      | Driver      | Add Steering Angle Velocity END Condition in VDF                               |
| 5793      | CarRealTime | Export the right EPS map from Adams                                            |
| 5167      | Road        | Allow general road interface to work with variable I/O points                  |
| 5127      | CarRealTime | Add roll center output to standard output set                                  |
| 4922      | CarRealTime | Improved management of masses and auxiliary vertical force in Adams Car Export |
| 4783      | Animator    | Dynamic filter files in AdamsDBfileDialog                                      |
| 4782      | Road        | Enable surf mode on new path selection                                         |
| 4781      | Road        | Enable smoothing function for oval and analytic roads                          |
| 4778      | Road        | Improve continuity of smoothing function for closed tracks                     |

*Release Notes*

|      |               |                                                                          |
|------|---------------|--------------------------------------------------------------------------|
| 4775 | SuspensionGen | Add bushing frame visualization                                          |
| 4774 | SuspensionGen | SuspensionGen orthographic view                                          |
| 4770 | SuspensionGen | Add tierod at lca option for rear mcpherson                              |
| 4769 | SuspensionGen | Option to move mcpherson arb on lca                                      |
| 4768 | SuspensionGen | Support for mcpherson with three links                                   |
| 4767 | SuspensionGen | Function to add and remove antiroll bar                                  |
| 4764 | SuspensionGen | Reference System not clear                                               |
| 4760 | SuspensionGen | Add specific activation flag for each setup function                     |
| 4533 | CarRealTime   | Remove un-used parameters from sample TIR files                          |
| 4515 | CarRealTime   | New SportCar shared model                                                |
| 4421 | CarRealTime   | Support TameTire v5.1                                                    |
| 4419 | CarRealTime   | Support FTire 2019-1                                                     |
| 4399 | CarRealTime   | Engine omega channel should be null when internal engine is disconnected |
| 4398 | Road          | Enable road width field loading a drd as road                            |
| 4356 | CarRealTime   | KnC Wizard support dataset containing NaN                                |
| 3526 | CarRealTime   | Model efficiencies as dissipation of energy                              |
| 3257 | CarRealTime   | Investigation mode                                                       |
| 3250 | CarRealTime   | Expose Scaling factor for suspension elements                            |
| 3145 | CarRealTime   | Enable use of multiple auxiliary subsystems                              |
| 3028 | CarRealTime   | Performance Scaling maps for MaxPerformance                              |
| 2737 | CarRealTime   | Gearbox inertia should be gear dependent                                 |
| 2733 | CarRealTime   | Gear dependent engine torque maps                                        |
| 2729 | CarRealTime   | Add option to define maximum engine speed per gear                       |
| 1402 | CarRealTime   | Increase model fidelity for ride events                                  |
| 1375 | CarRealTime   | "Body on frame" chassis architecture                                     |
| 1372 | CarRealTime   | Solid axle based vehicles                                                |
| 1343 | CarRealTime   | Startup logic behaviour improvements                                     |
| 905  | CarRealTime   | Allow csv file generation for each lap in multi-lap mode                 |
| 756  | Driver        | Path distance as end condition                                           |
| 751  | CarRealTime   | Suspension model for VI-SpeedGen                                         |

**Bugs Corrected:**

| Change ID | Module      | Description                                                                        |
|-----------|-------------|------------------------------------------------------------------------------------|
| 8023      | CarRealTime | Error running user tire sample with single contact point enabled                   |
| 7945      | CarRealTime | Opening up-to-date system - marked as modified                                     |
| 7894      | CarRealTime | Enabling left / right symmetry makes right data being set on the left side         |
| 7813      | CarRealTime | Failures with NI Veristand interface                                               |
| 7684      | CarRealTime | Error in Matlab Exported plugin compilation                                        |
| 7593      | Road        | Wrong road mesh colour                                                             |
| 7592      | Road        | Modified material on a mesh road is not visible on 3D after reloading the rdf file |
| 7414      | CarRealTime | Inerter elements don't apply force to model                                        |
| 7411      | Animator    | Default folder when importing analysisos is wrong                                  |

|      |               |                                                                                                            |
|------|---------------|------------------------------------------------------------------------------------------------------------|
| 7399 | Driver        | Update VDF option chooses wrong shift gear RPM for GSE-based powertrain                                    |
| 6166 | CarRealTime   | Published model still references old XGR file                                                              |
| 5991 | CarRealTime   | 7Post mex function: I/O update generates a res with wrong name                                             |
| 5990 | CarRealTime   | 7Post mex function does not support external plugin                                                        |
| 5976 | CarRealTime   | Problem generating a plugin from with "tunable" parameters                                                 |
| 5949 | CarRealTime   | Matlab crash using 7Post mex                                                                               |
| 5942 | CarRealTime   | "7post testrig end of reference signal reached" warning appears when computing first integration time step |
| 5926 | SuspensionGen | Multilink elastokinematics failure with arb on lca                                                         |
| 5896 | CarRealTime   | Coast_down.xml event is saved with its full path in fingerprint xml                                        |
| 5829 | CarRealTime   | Simulink Plugin Export - Makefile Issue                                                                    |
| 5788 | CarRealTime   | Speedgen fails when vehicle relocation flag is active                                                      |
| 5779 | CarRealTime   | Optimal gear definition fails during straight setup                                                        |
| 5765 | CarRealTime   | Engine Reaction torques get lost selecting an invalid body                                                 |
| 5754 | Driver        | Gear Mapped Module inhibit clutch idle control                                                             |
| 5695 | CarRealTime   | Engine mount request sign mismatch                                                                         |
| 5675 | CarRealTime   | Wrong help labels in powertrain editor                                                                     |

## 8.8.5 Release 18.2

| Version | Change ID | Change                                                                     |
|---------|-----------|----------------------------------------------------------------------------|
| 18.2    | FDB-6022  | Matlab api does not allow appending of aux subsystem to a system instance  |
|         | FDB-6020  | Document how to return material id from user tires                         |
|         | FDB-6016  | Cannot run event with obfuscated model                                     |
|         | FDB-6011  | SCANEr 1.7r37 certification                                                |
|         | FDB-6010  | Wrong contact patch estimation relocating vehicle on external road         |
|         | FDB-6004  | Obsolete tags are not removed from model                                   |
|         | FDB-6002  | Cannot run crg roads on SCALEXIO                                           |
|         | FDB-5986  | Rack-Pinion steering - EPS friction parameters are not parsed correctly    |
|         | FDB-5978  | Vehicle side slip is not properly defined at low speed                     |
|         | FDB-5973  | Cannot initialize simulink model when vfs block is inside a subsystem      |
|         | FDB-5969  | Steering motor_torque output channel is zero when using User Assist torque |
|         | FDB-5953  | Improve export process logging time spent on different stages              |
|         | FDB-5947  | Human driver fails to drive a maneuver that works with robot driver        |
|         | FDB-5946  | Throttle dependent gearshifting maps are hard to set                       |
|         | FDB-5945  | Press Maneuvers for MATLAB crash when launched the second time             |
|         | FDB-5943  | Torque converter splines don't have labels and units                       |
|         | FDB-5941  | Unsupported parts can be selected as bushing attachment in powertran model |
|         | FDB-5929  | Hysteretic Damper conflict with J Damper                                   |
|         | FDB-5928  | Wrong tooltip in trailer body subsystem                                    |
|         | FDB-5924  | Missing source UI file for engine bushings                                 |
|         | FDB-5918  | Wrong labels for frequency bushings in powertrain mount                    |
|         | FDB-5910  | Powertrain outputs wrongly computed when using Inactive clutch             |
|         | FDB-5909  | Print a warning in case of event initial gear greater than maximum gear    |
|         | FDB-5908  | Can't define initial gear greater than 7 for VI-Driver events              |

## Release Notes

|          |                                                                                                                                       |
|----------|---------------------------------------------------------------------------------------------------------------------------------------|
| FDB-5907 | Wrong main motor IC when using motor transmission ratio is different from 1                                                           |
| FDB-5899 | Need a Matlab API to refresh kingpin axis                                                                                             |
| FDB-5897 | Add parameter to skip inertial contribution on kingpin moment computation                                                             |
| FDB-5894 | Missing Payload panel in distributed trailer body subsystems                                                                          |
| FDB-5885 | Add two slalom maneuvers with cones spaced 18 and 36 m                                                                                |
| FDB-5883 | Missing save & restore procedure for trailer                                                                                          |
| FDB-5882 | Static results for trailer analysis not stored in output files                                                                        |
| FDB-5877 | Speedgen fails when the main motor is activated                                                                                       |
| FDB-5873 | Engine part with rods connection may produce instability                                                                              |
| FDB-5869 | Prevent to simulate models not declaring valid kinematics maps function of Rack Displacement in combination with Rack-Pinion steering |
| FDB-5866 | Wrong mass distribution in case of unmapped subsystems                                                                                |
| FDB-5863 | Incorrect response of LSD differential in example driveline.mdl model                                                                 |
| FDB-5861 | Wrong default location for car hitch ball to attach trailer                                                                           |
| FDB-5860 | Trailer example fails to simulate with SedanCar                                                                                       |
| FDB-5851 | Matlab API: error setting rear bumpstop/reboundstop property files                                                                    |
| FDB-5849 | Matlab API: writeSystemStruct doesn't update bumpstop/reboundstop property files                                                      |
| FDB-5844 | Adams Car export failure with old style powertrain template                                                                           |
| FDB-5842 | Unclear message reported during Adams Car export when one or more brake parameters are not found                                      |
| FDB-5841 | Adams Car export fails if cil_tierod_joint communicator is not matched                                                                |
| FDB-5836 | Missing documentation for output channel<br>OUTPUT_FV_Steering_System_DriveSim_Steering_Feedback_Torque                               |
| FDB-5649 | Long model name leads to export failure with unclear messages                                                                         |
| FDB-4853 | Spikes in tire forces with MFTyre running on briges                                                                                   |
| FDB-4515 | Trailer is unable to retrieve user vehicle location                                                                                   |

## 8.8.6 Release 18.1

| Version | Change ID | Change                                                                                       |
|---------|-----------|----------------------------------------------------------------------------------------------|
| 18.1    | FDB-5811  | Improve user tire example with non-linear forces implementation                              |
|         | FDB-5796  | Rack and pinion steering may incorrectly partition upper column inertia                      |
|         | FDB-5768  | Main engine ratio is not propagated to VI-SpeedGen                                           |
|         | FDB-5766  | Crash writing simulation final statistics with small circular buffer                         |
|         | FDB-5765  | Wrong force at wheel computation using 4 wheel steering models                               |
|         | FDB-5763  | Open differential dialog shows a wrong GUI                                                   |
|         | FDB-5760  | User tires cannot distinguish between speedgen and crt invocation                            |
|         | FDB-5756  | Torque Map Property File is incorrectly required with Simple Motor Type                      |
|         | FDB-5754  | Cannot run fire with viroad rdf on linux                                                     |
|         | FDB-5753  | User event vdf contains a time based end condition                                           |
|         | FDB-5741  | Speedgen solver crashes when using MaxPerformance with TireLimits                            |
|         | FDB-5739  | Apply flying car plane computation mode to user sensors                                      |
|         | FDB-5738  | Options in Mode of Simulation menu for events vanish                                         |
|         | FDB-5736  | GUI: driveline layout broken when no Motor is selected                                       |
|         | FDB-5721  | Additional rack displacement and steering deactivation extra input conflicts standard inputs |
|         | FDB-5719  | Wrong front and rear unsprung inertia with Adams/Car Plugin                                  |

|          |                                                                                       |
|----------|---------------------------------------------------------------------------------------|
| FDB-5718 | Various documentation errors                                                          |
| FDB-5717 | Ride Height map with load - experimental mode                                         |
| FDB-5716 | Problems running coast down event with dual clutch transmission                       |
| FDB-5709 | SpeedGen fails when using wheel motors in powertrain subsystem                        |
| FDB-5703 | GUI hangs when a wrong aerodynamic property file path is specified                    |
| FDB-5700 | External engine co-simulation failure during 7Post analysis                           |
| FDB-5699 | Wrong xform computation for singular conditions                                       |
| FDB-5692 | Suspension frequency dependent longitudinal DOF                                       |
| FDB-5686 | FMI: External Driveline tutorial is not aligned with distributed files                |
| FDB-5685 | Multi thread unpredictable behavior                                                   |
| FDB-5678 | Deactivated ARBs are reactivated by plugin during export                              |
| FDB-5677 | Reboundstop property file curve are misinterpreted                                    |
| FDB-5671 | Distributed solverPlugin_template.cpp example doesn't provide all available callbacks |
| FDB-5670 | Engine torque reaction not updated properly                                           |
| FDB-5669 | Fz Compliance are calculated in a wrong way                                           |
| FDB-5660 | Need a command line switch to override output prefix                                  |
| FDB-5657 | Closing RunWindow using close button(X) may leave the gui locked                      |
| FDB-5655 | Support for Adams 2017.2                                                              |
| FDB-5652 | Improve bumpstop management in model export                                           |
| FDB-5651 | Default tire testrig rt integrator is euler                                           |
| FDB-5613 | Save As button for Property Files do not update GUI                                   |
| FDB-5573 | Channels not properly updated in GUI                                                  |
| FDB-5572 | CRT/TireLimits pre-processing error using MFTYRE 6.x                                  |
| FDB-5556 | Errors occurs loading analysis with animation from CRT in Adams PPT                   |
| FDB-5551 | Expose bumper curve in gui as done for damper                                         |
| FDB-5470 | Impossible to delete a User Sensor or modify sensor type                              |
| FDB-5468 | Gearbox image doesn't disappear when the component has been deactivated               |
| FDB-5451 | VI-Road RDF loaded multiple time with FTIRE                                           |
| FDB-5428 | RH maps computation depends on vehicle user location parameter                        |
| FDB-5406 | Expose tire contact patch on-road/out-of-road output                                  |
| FDB-5063 | Support execution maxpeformance and pressmaneuvers solvers from command line          |
| FDB-4755 | Implement a simulink target to compile Simulink model as VI-Crt plugins               |
| FDB-4736 | VI-CRT batch launcher crashes when short path names are not available                 |
| FDB-4697 | Runtime batch for dll creation are not compatible with windows compiler 14.0 (VS2015) |
| FDB-4610 | Output channel filter for Active Only                                                 |
| FDB-3818 | Implement reload button for output map editor                                         |

## 8.8.7 Release 18.0

| Version | Change ID | Change                                                                       |
|---------|-----------|------------------------------------------------------------------------------|
| 18.0    | FDB-5645  | Wrong engine speed initialization with user tire                             |
|         | FDB-5590  | Solver crash during straight setup analyses when integration step is not 1ms |
|         | FDB-5589  | setSystemParameterTreeValue doesn't manage multiple parameters types         |
|         | FDB-5565  | Cannot register auxiliary output from more than 1 plugin                     |

## Release Notes

|          |                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------|
| FDB-5561 | Wrong initialisation with external engine and user defined location                             |
| FDB-5558 | Speedgen produce wrong engine torque during braking in output channel                           |
| FDB-5527 | Wrong CG location in HTML validation report when engine part is active                          |
| FDB-5515 | Crash with large RDF on XPC target                                                              |
| FDB-5509 | Assembly/design condition in report file is wrong computed when using external engine           |
| FDB-5499 | Possible static failures with automatic z location                                              |
| FDB-5480 | Add new output channels reporting the tire to road closest point even when the tire is flying.  |
| FDB-5465 | Wrong torque output from engine mount for large deformation                                     |
| FDB-5458 | SpeedGen input generator (spg_igen) does not report error messages                              |
| FDB-5455 | Output step field disappear after a VI-DriveSim event execution                                 |
| FDB-5447 | Rear suspension dependency input does not work in simulink                                      |
| FDB-5433 | Traceback executing events referencing a file from a non registered database                    |
| FDB-5423 | Missing aerodynamic forces in simulink events                                                   |
| FDB-5415 | Improve documentation about Side view angle                                                     |
| FDB-5399 | Simulink simulation fails after second consecutive execution                                    |
| FDB-5392 | Longitudinal suspension DOF misbehavior (16.2 vs. 17.2)                                         |
| FDB-5389 | MF-Tyre/MF-Swift license not returned when static equilibrium analysis fails                    |
| FDB-5388 | license error with MF-Tyre/MF-Swift tire property file and mode 125                             |
| FDB-5387 | Output prefix field cannot be a full path                                                       |
| FDB-5376 | Custom events can't be loaded in vicrt GUI                                                      |
| FDB-5339 | Vicrt-plugin: Number Of Analyses parameter in auxiliary roll stiffness setup panel is neglected |
| FDB-5338 | Create a tutorial for using an external steering system                                         |
| FDB-5316 | Add low speed threshold to internal abs controller                                              |
| FDB-5315 | Crt plugin: set vicrt integration step in automatic model validation panel                      |
| FDB-5309 | First order compliance improvements                                                             |
| FDB-5306 | Matlab API: error when using frontSuspensionMainSpringFileSetSpline2D function                  |
| FDB-5302 | Document output prefix in mxp simulation with Matlab                                            |
| FDB-5301 | Wrong units in distributed 'aero_advanced.aer' file                                             |
| FDB-5300 | Wrong Cz values using advanced aerodynamic module                                               |
| FDB-5283 | Example Databases are no more shipped with vicrt installer                                      |
| FDB-5282 | Port to adams 2017                                                                              |
| FDB-5281 | VI-Drivesim Files Path field is not used as target path for send creation                       |
| FDB-5278 | Possible crash when using a plugin dll in Simulink environment                                  |
| FDB-5274 | Missing documentation for standard aerodynamic                                                  |
| FDB-5270 | Steering compliance lock should be deactivated during steering feedback computation             |
| FDB-5268 | Skidplate Forces computation on 3d road with high height/bank variation                         |
| FDB-5253 | VI-Driver feedforward problems using external aerodynamic                                       |
| FDB-5251 | Possible crashes during 7Post analysis                                                          |
| FDB-5243 | Crt plugin: mismatched powertrain splines when pvs_max_gears differs from model gear number     |
| FDB-5242 | Add license troubleshooting topic in documentation                                              |
| FDB-5237 | Automatic computation of kingpin location/orientation splines                                   |
| FDB-5236 | PressManeuvers: wheel/cone base intersection not detected                                       |
| FDB-5232 | Isolc unable to retrieve tire width with MF-Tyre                                                |

|          |                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------|
| FDB-5231 | Vehicle wizard errors when defaultPrefs.xml is not in the working directory                         |
| FDB-5226 | TireLimits: Graph scaling for tire ellipse                                                          |
| FDB-5225 | "Setup on flat road" flag not working when PSD or user location are active                          |
| FDB-5221 | Error when gear number is less than 4 in Gear Ratio table                                           |
| FDB-5208 | Incorrect negative engine torque in STL event                                                       |
| FDB-5207 | Engine rpm exceed the MaxRpmLimit during STL event when Lazy=0                                      |
| FDB-5202 | GUI: changing rdf file doesn't make fingerprint bold                                                |
| FDB-5192 | Include driveshaft reaction torques to crt model                                                    |
| FDB-5185 | No effects of Initial Gear flag in MaxPerformance & SpeedGen events                                 |
| FDB-5184 | Manage external inputs with Rk2 solver                                                              |
| FDB-5179 | Documenting external road                                                                           |
| FDB-5176 | Speedgen brake bias doesn't match the actual torque bias                                            |
| FDB-5170 | Database publish doesn't work when model has files that are not directly in a database              |
| FDB-5160 | Rpm limiter doesn't work with lazy shift enabled                                                    |
| FDB-5155 | Models with compliance symmetry management                                                          |
| FDB-5143 | Single damper element doesn't provide force at wheel                                                |
| FDB-5131 | Missing contribute on kingpin moment computation                                                    |
| FDB-5128 | Max engine brake torque scaled by driveline efficiency                                              |
| FDB-5091 | Acceleration is greater if the negative engine torque is increased (at big values)                  |
| FDB-5090 | Carsim 9.0 models failing export                                                                    |
| FDB-5087 | SCANEr interface file does not compute correctly tire radius and steering ratio                     |
| FDB-5075 | Problems exporting a model with a compliant Steering                                                |
| FDB-5073 | Automatically add example vehicle models to the database configuration during installation          |
| FDB-5069 | Error loading cfg file with path containing DATABASE string                                         |
| FDB-5067 | Vimaxperformance command line doesn't match other crt solver one                                    |
| FDB-5051 | Add output channels for driver demands.                                                             |
| FDB-5048 | Upgrade openCRG lib to version 1.1                                                                  |
| FDB-5043 | TNO tires are not compatible with external road                                                     |
| FDB-4998 | Crash using extra input for suspension testrig event                                                |
| FDB-4997 | Expose road material id among tire output                                                           |
| FDB-4988 | K&C interface: track and wheelbase variation offset                                                 |
| FDB-4980 | Analysis failure when selecting output step > 0.01                                                  |
| FDB-4966 | CRT user sensors should all be defined in vehicle reference frame                                   |
| FDB-4940 | Speedgen: wrong tire normal force calculation on banked road                                        |
| FDB-4930 | Independent integration time step for engine part                                                   |
| FDB-4928 | Add driver longitudinal target input                                                                |
| FDB-4853 | MFTyre contact patch computation failure on meshed bridge                                           |
| FDB-4792 | Add STEERING DAMPING parameter (to run steer release maneuver with the std steering subsystem)      |
| FDB-4776 | SpeedGen 2D ride height maps                                                                        |
| FDB-4763 | Rack and pinion steering redesign for consistency with VI-DriveSim                                  |
| FDB-4727 | Integrate akima spline on vinct core                                                                |
| FDB-4677 | SpeedGen calculates a negative lap time using a low initial speed (<0.1m/s & >0m/s) in the XSG file |
| FDB-4659 | Port to dspace scalexio (release 2016)                                                              |

## Release Notes

|          |                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------|
| FDB-4644 | Multi-thread support for USER tires                                                                 |
| FDB-4642 | Single-road support for MF-Tyre tire                                                                |
| FDB-4640 | Support for multi-thread pacejka tires                                                              |
| FDB-4598 | VI-CRT K&C Interface revamping                                                                      |
| FDB-4569 | Nam file is not updated if already exists                                                           |
| FDB-4567 | Update system output_map xml file                                                                   |
| FDB-4561 | Units not shown in differential curve editor                                                        |
| FDB-4531 | MXP and VDD giving different results                                                                |
| FDB-4507 | FTIRE contact patch force output strongly affected by multithread mode                              |
| FDB-4463 | Possible lack of concurrence between rack travel vs steer wheel angle and steering feedback splines |
| FDB-4308 | Efficiency map for transmission losses                                                              |
| FDB-4303 | CRT log file is ill formatted                                                                       |
| FDB-4292 | Adv steering messages not included in log file                                                      |
| FDB-4264 | Could not submit a new event because gui is locked                                                  |
| FDB-4240 | Generalize compliance data set for k&C import                                                       |
| FDB-4224 | Ride height                                                                                         |
| FDB-4074 | Add parameters to VI-Driver to control delay between consecutive gearshifting                       |
| FDB-3994 | K&C interface compliance loadcases                                                                  |
| FDB-3964 | Suspension Compliance - missing or ill defined                                                      |
| FDB-3837 | Missing center differential outputs                                                                 |
| FDB-3681 | Efficiency scaling for engine and gearbox                                                           |
| FDB-2633 | Automatic plot generation using tire testrig CRT                                                    |
| FDB-2576 | Improve multiple road initialization in CRT                                                         |

**8.8.8 Release 17.3**

| Version | Change ID | Change                                                                     |
|---------|-----------|----------------------------------------------------------------------------|
| 17.3    | FDB-5319  | VI-DriveSim event does not produce SCANEr interface file when requested    |
|         | FDB-5310  | Support for ETAS HIL platform                                              |
|         | FDB-5278  | Random crashes using VI-CarRealTime S/function in combination with plugins |
|         | FDB-5010  | Matlab API are not compatible with Matlab 2016b                            |

**8.8.9 Release 17.2**

| Version | Change ID | Change                                                                             |
|---------|-----------|------------------------------------------------------------------------------------|
| 17.2    | FDB-5191  | Kingpin moment computation includes driving torques reacting on chassis            |
|         | FDB-5177  | Right side Compliance vs Fx forces not computed during suspension testrig analysis |
|         | FDB-5161  | Port Adams Car interface to Adams 2016                                             |
|         | FDB-5159  | Matlab API documentation is not fully updated to v17.x data structure              |
|         | FDB-5158  | Add a matlab API to Get/Set damper 3D spline data                                  |
|         | FDB-5149  | Static analysis failure after vehicle relocation                                   |
|         | FDB-5137  | Add path_distance channel to lap_sensor outputs                                    |
|         | FDB-5124  | Implement possibility to generate and execute fully obfuscated models              |
|         | FDB-5112  | SpeedGen startup strategy fails in case of non monotonic powertrain map            |
|         | FDB-5106  | Matlab API: error when using frontTireSetPropertyFile function                     |

|          |                                                                                  |
|----------|----------------------------------------------------------------------------------|
| FDB-5078 | Fx compliance doesn't switch with external braking model                         |
| FDB-5066 | No error flag returned by pressmaneuver executable                               |
| FDB-5062 | Binary file encryption error undetected                                          |
| FDB-5030 | Vehicle relocation api issue                                                     |
| FDB-5026 | "Missing Input Errors" message related tire force graphics with 7Post simulation |
| FDB-5025 | Missing spring and damper property file causes errors                            |
| FDB-5024 | Wrong lower cardan ratio output channel                                          |
| FDB-5017 | Add an Harware specification page to documentation                               |
| FDB-5005 | Report file does not contain design condition                                    |
| FDB-5003 | Automatic z location option is always active                                     |
| FDB-5000 | MxP Matlab: license not released after CTRL+C                                    |
| FDB-4992 | SCANeR interface file inherits the system file name instead of the event name    |
| FDB-4991 | Steer release maneuver in Simulink not supported for base steering system        |
| FDB-4967 | Matlab API performance problem                                                   |
| FDB-4948 | 3D damper spline not shown in Curve Editor                                       |
| FDB-4927 | Add Tire Limits capability to DriveSim Event                                     |
| FDB-4839 | Rep file misses Assembly Conditions when no setup is performed                   |
| FDB-4809 | Wrong contact patch computation in suspension testrig                            |
| FDB-3822 | Crash writing files when working dir is wrong                                    |
| FDB-3061 | Solver Settings GUI fields corrupted at startup                                  |

## 8.8.10 Release 17.1

| Version | Change ID | Change                                                                            |
|---------|-----------|-----------------------------------------------------------------------------------|
| 17.1    | FDB-4965  | MaxPerformance event predicts 0s lap time when initial speed is 0                 |
|         | FDB-4963  | Aero_forces_rear_sideforce output always equals to aero_forces_rear_downforce     |
|         | FDB-4953  | Add solver status output channel                                                  |
|         | FDB-4950  | Lsd c1 table not read from 16.2 setup file                                        |
|         | FDB-4941  | Error in path compensation output computation during static analysis              |
|         | FDB-4938  | Publishing operation breaks .obj references on source body subsystem              |
|         | FDB-4929  | Static equilibrium for models with engine part lasts forever                      |
|         | FDB-4926  | Misaligned lap sensor and lap time triggers                                       |
|         | FDB-4924  | Missing example dcd file                                                          |
|         | FDB-4922  | Matlab API --> error in creating Struct if Auxillary subsystem block is present   |
|         | FDB-4920  | Body subsystem is not marked as modified when ride height maps are recomputed     |
|         | FDB-4916  | VI-CarRealTime export generates fixed analysis names                              |
|         | FDB-4912  | Possible wrong average wheel travel in auxiliary roll force with asymmetric range |
|         | FDB-4908  | Possible Vehicle model export failure due to FORTRAN solver incompatibility       |
|         | FDB-4905  | Missing example file for engine part subsystem                                    |
|         | FDB-4899  | Left over file handles in matlab environment may lead to simulation failure       |
|         | FDB-4898  | Add output channels for road normal                                               |
|         | FDB-4896  | Missing user outputs deactivation using output map or env var VI_XFORM_REMOVE=1   |
|         | FDB-4887  | Tirelimits: spikes in lon/lat margin channels                                     |
|         | FDB-4886  | Speedgen: randomic crash                                                          |

## Release Notes

|          |                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------|
| FDB-4885 | GUI widget for vehicle setup is not properly updated for central elements                           |
| FDB-4880 | OpenCRG and OpenDRIVE sensor indexing problems                                                      |
| FDB-4879 | Improve documentation for Suspension Compliances                                                    |
| FDB-4875 | Skidplate computation may lead incorrect results when more than one wheel is flying                 |
| FDB-4874 | Steering.steer_at_spindle input channels don't work                                                 |
| FDB-4869 | Wind Effect in .aer files for Static Laptime Event neglects vehicle yaw                             |
| FDB-4854 | GUI doesn't open if vicrt_17_defaultPrefs.xml has a wrong working dir                               |
| FDB-4852 | Support multiple dspace overlays on the same installation                                           |
| FDB-4845 | Wrong current system focus refreshing a subsystem                                                   |
| FDB-4843 | Servo Gear Differential equation deos not work with internal EPS model                              |
| FDB-4840 | Aerodynamic modifiers panel not updated at startup                                                  |
| FDB-4838 | MaxPerformance event may be terminated prematurely when the vehicle is relocated                    |
| FDB-4832 | Missing vehicle configuration modified status                                                       |
| FDB-4829 | FTIRE custom installation failure when COSIN_PREFIX includes spaces                                 |
| FDB-4825 | Speedgen post processing may alter speed profile too much                                           |
| FDB-4820 | Susbsystem reload doesn't work properly                                                             |
| FDB-4819 | Failure on speedgen initialization due to engine map                                                |
| FDB-4816 | Speedgen optimal gear is wrong at low rpm                                                           |
| FDB-4815 | Suspension testrig failure using VI-Steering                                                        |
| FDB-4812 | User sensors calc frequency is bound to output rate                                                 |
| FDB-4810 | Driving Machine File Editor not working                                                             |
| FDB-4808 | Save As subsystem replaces the subsystems of all the systems                                        |
| FDB-4807 | VI-TireLimits GUI shows wrong camber slider range                                                   |
| FDB-4804 | Incorrect default longitudinal slip step for runtime TireLimits computation                         |
| FDB-4803 | Event names are case sensitive                                                                      |
| FDB-4800 | Can't edit compliance vs fz map with GUI                                                            |
| FDB-4798 | ISO standard typo in GUI                                                                            |
| FDB-4796 | PressManeuvers Matlab crashes after stop with CTRL+C                                                |
| FDB-4791 | Compliance vs fz data are neglected in overall compliance computation                               |
| FDB-4790 | Add support for air spring                                                                          |
| FDB-4786 | VI-CarRealTime crash using TireLimits from simulink                                                 |
| FDB-4782 | Tierod force computation is skipped when spline rack_displacement_vs_steering_wheel is not defined  |
| FDB-4781 | MaxPerfMatlab ignores the output prefix writing the result file                                     |
| FDB-4767 | Add support for dSPACE release 2014-B and 2015-B                                                    |
| FDB-4760 | Vehicle center of gravity height depends on design wheel center location                            |
| FDB-4759 | Incorrect Traction control computation when wheel center design position doesn't match wheel radius |
| FDB-4758 | Can't export Adams Car models including linear springs                                              |
| FDB-4750 | Missing checkin of road core license in speedgen                                                    |
| FDB-4742 | Support for partially obfuscated mesh roads                                                         |
| FDB-4740 | Renaming in tree view is not triggered by two consecutive clicks                                    |
| FDB-4737 | Support export from VI-Automotive of skidplate component                                            |
| FDB-4735 | Update FMI Examples distributed with CRT                                                            |
| FDB-4734 | Missing win32 dll in double wishbone FMI distributed in vicrt                                       |

|          |                                                                                                                    |
|----------|--------------------------------------------------------------------------------------------------------------------|
| FDB-4733 | Change FMU unzip directory path generation                                                                         |
| FDB-4729 | Carsim importer neglects engine inertia and initial toe/camber                                                     |
| FDB-4728 | K&C wizard incorrect definition of toe map for longitudinal compliance                                             |
| FDB-4726 | Vehicle setup may fail on 3d road with vehicle relocation                                                          |
| FDB-4725 | Longitudinal slip problems on tire testing using PAC tires                                                         |
| FDB-4724 | LSD preload should not be included in the torque reacting to C1                                                    |
| FDB-4721 | OUTBOUND_SAFE option could not work correctly                                                                      |
| FDB-4695 | Incorrect wheel setup (toe/camber) when reference is set to ground                                                 |
| FDB-4681 | Servo Steering Map doesn't neglect when Steering Feedback Map is active                                            |
| FDB-4679 | Live animation is interrupted when a MaxPerformance attempt leads to an equation of motion failure                 |
| FDB-4678 | Speedgen parameters --> check the correct value to define the MIN VELOCITY in the XSG file                         |
| FDB-4675 | SpeedGen crashes using an initial speed = 0.0m/s                                                                   |
| FDB-4674 | Could not set initial speed value = 0m/s in SpeedGen simulation                                                    |
| FDB-4654 | Adjustable bumpstop does not support negative clearance                                                            |
| FDB-4643 | OpenCRG roads are loaded multiple times                                                                            |
| FDB-4631 | Gravity option for acceleration sensors                                                                            |
| FDB-4630 | CRT does not exit immediately after tire initialization error                                                      |
| FDB-4629 | Component property files can't be saved                                                                            |
| FDB-4627 | Pressmaneuver event should not use absolute path referencing VDF file                                              |
| FDB-4616 | Clutch torque is not 0 when gear is neutral                                                                        |
| FDB-4613 | Vehicle setup doesn't work with individual roads                                                                   |
| FDB-4607 | Certify compatibility with veristand 2015                                                                          |
| FDB-4606 | Update DoubleLaneChange and ObstacleAvoidance according to ISO-3888-1/2                                            |
| FDB-4602 | Fuel consuption does not allow cutoff conditions                                                                   |
| FDB-4592 | User input channels disappear from simulink interface after cancel button                                          |
| FDB-4588 | Missing end condition in VDF file generated by PressManeuvers for Matlab                                           |
| FDB-4587 | Linker errors while compiling solver plugin libraries                                                              |
| FDB-4579 | Antrollbar forces imported from K&C data may be wrong when active and inactive antrollbar loadcases are provided   |
| FDB-4564 | Application crash running PressManeuvers with circular buffer                                                      |
| FDB-4563 | Memory leak running PressManeuvers with experimental clutch                                                        |
| FDB-4551 | Add upper speed limit field to PressManeuvers event                                                                |
| FDB-4550 | Allow storage of all feasible run for PressManeuvers event                                                         |
| FDB-4539 | Full vehicle export using "No Suspension and Steering" option fails when subsystems don't exist in target database |
| FDB-4536 | Vi-driver closed loop event failure when vehicle is moved away from origin                                         |
| FDB-4529 | Final drive ratio field should not appear in transmission editor                                                   |
| FDB-4505 | Road normal from crg roads with pac2002 is always assumed to be vertical                                           |
| FDB-4326 | Create a comprehensive library of example models                                                                   |
| FDB-4147 | Support import of CarSIM 9.0 models                                                                                |
| FDB-2493 | Bumpstop adjustment failure when no additional adjustments are active                                              |
| FDB-1682 | Advanced aero forces: support for wind effect                                                                      |

## 8.8.11 Release 17.O

| Version | Change ID | Change                                                                                   |
|---------|-----------|------------------------------------------------------------------------------------------|
| 17.0    | FDB-4519  | Throttle demand should be 0 when steering input is applied in J-Turn and Fishhook events |
|         | FDB-4518  | Implement ramp steer event                                                               |
|         | FDB-4513  | Lap sensor does not detect 1st lap when vehicle is started away from the origin          |
|         | FDB-4512  | Log communicators and flags found in the Adsm Car model affecting the export process.    |
|         | FDB-4511  | VI-CarRealTime interface fails to start when using "run as administrator" option         |
|         | FDB-4503  | Report file contains wrong cog position when using external road                         |
|         | FDB-4486  | Internal TCS should not work when vehicle is in null gear or in clutch open conditions   |
|         | FDB-4485  | Mismatched LSD differential implementation on matlab example models                      |
|         | FDB-4469  | Vehicle dimensions are stored in press maneuvers gui                                     |
|         | FDB-4439  | Error reading drd track file from mesh rdf                                               |
|         | FDB-4438  | Auxiliary longitudinal dof - not consistent hub acceleration output channels             |
|         | FDB-4437  | Missing initialization of low wheel omega filter                                         |
|         | FDB-4415  | Unrealistic default parameter settings for canned sine steer event                       |
|         | FDB-4413  | Driver steering ratio input is not correct when using vi-steering module                 |
|         | FDB-4412  | Rack and pinion steering should support positive spline for EPS electric motor current   |
|         | FDB-4411  | Engine rpm not initialized with vi_crt_demo model and settle setup mode                  |
|         | FDB-4393  | Understeer_gradient output channel is noisy for the first few samples                    |
|         | FDB-4387  | Separate max and min saturation for rack and pinion column friction model                |
|         | FDB-4358  | Extend rack and pinion steering friction model with hyperbolic formulation               |
|         | FDB-4357  | Inertia of all steering column parts should be editable from the gui                     |
|         | FDB-4356  | Plugin library name definition for rack and pinion steering is not required              |
|         | FDB-4355  | Add rigid switch in rack and pinion steering to stiffnesses instead of using 0           |
|         | FDB-4350  | MATLAB/Simulink cosimulation with FTire module fails                                     |
|         | FDB-4330  | MaxPerformance files does not support database references                                |
|         | FDB-4320  | Documentation for differential torque in Limited Slip Differential is wrong              |
|         | FDB-4316  | Support request as output file format                                                    |
|         | FDB-4315  | Integrate external road                                                                  |
|         | FDB-4305  | K&C Interface - Missing left side toe compliance for single loadcases                    |
|         | FDB-4301  | Model imported from CarSim may miss spring data                                          |
|         | FDB-4297  | Clutch model for crt demo should be made active by default                               |
|         | FDB-4294  | Antirollbar activation inputs do not work when ARB is defined as dual table              |
|         | FDB-4291  | VI-CarRealTime Adams Car plugin exports incorrect names for displacements request        |
|         | FDB-4284  | Support for Matlab 2014                                                                  |
|         | FDB-4274  | Error in sprung mass CG computation in K&C interface                                     |
|         | FDB-4271  | Implement api call to extract current simulation time step                               |
|         | FDB-4270  | Add road friction to std output channels                                                 |
|         | FDB-4246  | Potential crash when calculation is interrupted pressing CTRL+C                          |
|         | FDB-4244  | Python.exe crash running VI-CarRealTime interface with IST timezone                      |
|         | FDB-4242  | Tire user interface enhancements should support save/restore                             |
|         | FDB-4239  | No output files generated when working dir is not existing                               |
|         | FDB-4234  | Wrong contact patch computation in case of 7Post events                                  |

|          |                                                                                                      |
|----------|------------------------------------------------------------------------------------------------------|
| FDB-4228 | Detroit locker toggle button incorrect behaviour                                                     |
| FDB-4227 | VI-CarRealTime live animation channels are fixed                                                     |
| FDB-4221 | VI-Safety events are not compatible with external engine model                                       |
| FDB-4218 | Could not run custom cosin tools using cosin_prefix environment variable                             |
| FDB-4216 | SevenPostrig event does not work with advanced steering                                              |
| FDB-4215 | Advanced steering model with hydraulic eps does not restore all states properly                      |
| FDB-4206 | CarSIM importer fails to import model with overall ride+tire stiffness                               |
| FDB-4199 | Tire error message unclear                                                                           |
| FDB-4198 | VI-Speedgen may retrieve wrong friction value from road file                                         |
| FDB-4192 | Obstacle avoidance simulates too long                                                                |
| FDB-4184 | Advanced steering may produce wrong results when steer to rack spline has negative slope             |
| FDB-4178 | Could not run FTIRE simulation once custom COSIN_PREFIX is defined                                   |
| FDB-4176 | Misleading error msg from tno tyre model                                                             |
| FDB-4174 | Add solver input for missing activation modules                                                      |
| FDB-4171 | VI-CarRealTime Suspension testrig analysis does not support solver plugins                           |
| FDB-4170 | VI-CarRealTime Suspension testrig analysis does not support vehicle setup                            |
| FDB-4169 | Static analysis failure with setup and uneven road                                                   |
| FDB-4146 | K&c interface: longitudinal opposing loadcases report                                                |
| FDB-4140 | Wrong folder is proposed browsing for an auxiliary subsystem                                         |
| FDB-4134 | Missing documentation for MaxPerformance Tire LImits Predictor flag                                  |
| FDB-4125 | Support user defined vehicle initial location/orientation                                            |
| FDB-4124 | Suspension compliance maps are not mirrored in case of left/right symmetry                           |
| FDB-4120 | Memory leak in simulink s-function                                                                   |
| FDB-4119 | 2d auxiliay anti roll map exported by v16.2 differs from v16.0                                       |
| FDB-4116 | Unexpected vehicle response when changing SteeringGearRatioScale                                     |
| FDB-4091 | Automatic validation: straight acceleration shifting time mismatch vs adams                          |
| FDB-4082 | Incorrect VI-Driver response to std_tire_ref marker shift                                            |
| FDB-4077 | License check failure on PXI platform with multiple network cards                                    |
| FDB-4076 | VI-Animator launched from utilities menu does not inherit current CRT configuration                  |
| FDB-4072 | Spring motion ratio does not depend on steering                                                      |
| FDB-4067 | Steering wheel velocity output returned to simulink is when the rack input is used                   |
| FDB-4061 | Missing Matlab api for auxiliary roll maps                                                           |
| FDB-4060 | File Select erroneously returns "None" file when pressing cancel button                              |
| FDB-4056 | Update logic to detect existing accessories like VI-Road and VI-Animator allowing manual selection.  |
| FDB-4029 | K&C license is not returned when data import fails                                                   |
| FDB-4011 | Simulink output for steering torque is wrong when an external steering model is used                 |
| FDB-4004 | Road graphic is wrongly positioned in VI-Animator when std_tire_ref is shifted from default location |
| FDB-4003 | Aerodynamic auxiliary front sensor is connected to rear chassis when body compliance is active       |
| FDB-4001 | Export problems when model contains custom steering displacements request                            |
| FDB-3997 | Implement Press Maneuver event for Matlab                                                            |
| FDB-3995 | Images in K&C interface compliance documentation are wrong                                           |
| FDB-3991 | MaxPerformance events stops complaining about no differences in computed speed profile               |
| FDB-3990 | MaxPerformance analysis log file report incorrect scaling factor                                     |

## Release Notes

|          |                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------|
| FDB-3984 | Misleading warnings about extrapolation at time=0 with spc file                                             |
| FDB-3983 | User sensor output for 7Post analysis in simulink is 1 step delayed                                         |
| FDB-3975 | Road Data file with friction table shorter than road length can produce 0 friction                          |
| FDB-3963 | Add external inputs to support manuever based control channel switching                                     |
| FDB-3866 | Un-installer doesn't remove VI-CarRealTime accessories                                                      |
| FDB-3829 | Integrate matlab resreader tool                                                                             |
| FDB-3792 | Upgrade FMI master to version 2.0                                                                           |
| FDB-3741 | Wrong VI-SpeedGen limit for braking phase stored in result file                                             |
| FDB-3704 | Upgrade VI-CarRealTime GUI architecture                                                                     |
| FDB-3700 | External suspension support through fmi                                                                     |
| FDB-3698 | Minor problems converting XML event to VDF                                                                  |
| FDB-3683 | VI-SpeedGen and Road are initialized at each MaxPerformance iteration                                       |
| FDB-3682 | VI-Road call from utilities menu doesn't set current working dir                                            |
| FDB-3674 | Support for Adams 2015                                                                                      |
| FDB-3669 | Improve custom tire documentation                                                                           |
| FDB-3649 | Small differences between res file generated by MaxPerformance and FileDriven event                         |
| FDB-3639 | Road graphics not updated after rdf modification                                                            |
| FDB-3584 | CTRL+C does not stop execution gracefully with FTIRE                                                        |
| FDB-3508 | VI-Driver end_condition triggering information are not reported in log file                                 |
| FDB-3406 | SevenPostrig live animation doesn't work                                                                    |
| FDB-3397 | Windows 8.1 start menu: unable to distinguish documentation links                                           |
| FDB-3383 | VI-CarRealTime execution failure when VICRT_INST_DIR environment variable points to a previous installation |
| FDB-3293 | Could not change output prefix in Maxperformance for Matlab simulation                                      |
| FDB-3263 | Strong influence of maxlongslip coefficients on VI-SpeedGen predicted max speed                             |
| FDB-3259 | VI-CarRealTime preferences file are not version dependent                                                   |
| FDB-3247 | Implement skidplate component                                                                               |
| FDB-3119 | Defect of online-help in Kinematics/Steer angle topic                                                       |
| FDB-2984 | Enable definition of custom output channels from solver plugin                                              |
| FDB-2906 | Support NI-VeriStand 2014                                                                                   |
| FDB-2880 | Sprung mass in autogenerated suspension assembly is incorrect                                               |
| FDB-2536 | Rolling resistance returned by tir file marked as 'MF-Tyre' is wrong                                        |
| FDB-2464 | Support for user defined event chain (Custom events)                                                        |
| FDB-2158 | Send file can be loaded as model                                                                            |
| FDB-2157 | File not found error message running solver job                                                             |
| FDB-2129 | Conflicts with startup scripts using multiple vi-carrealtime version                                        |
| FDB-2046 | VI-CarRealTime --> velocity profile source for output vdf                                                   |
| FDB-2044 | Build mode may become not accessible                                                                        |
| FDB-790  | Raise a warning when installation stiffness is smaller than the spring stiffness                            |
| FDB-128  | License Expiration Warning                                                                                  |

## 8.8.12 Release 16.2

| Version | Change ID | Change                                                     |
|---------|-----------|------------------------------------------------------------|
| 16.2    | FDB-3888  | Extended search path for vicrt_cdb.cfg to send file folder |

|          |                                                                              |
|----------|------------------------------------------------------------------------------|
| FDB-3886 | Wrong scaling factor exporting spline based differentials from VI-Automotive |
| FDB-3876 | MESH road data file memory reallocation issues                               |
| FDB-3869 | Powertrain map interpolation fails                                           |
| FDB-3868 | Crash with negative motion ratio and setup                                   |
| FDB-3859 | Error when deactivating ARB subsystem                                        |
| FDB-3842 | Trailer analysis depends on std tire ref position                            |
| FDB-3836 | First 2 loadcases are missing in suspension test-rig res file                |
| FDB-3834 | Hard-coded aligning torque values in steering.lcf                            |
| FDB-3819 | Smoothing time not working for SURFMESH when CP mode = 1                     |
| FDB-3817 | Python fitting utils does not compute correct polynomial coefficients        |
| FDB-3814 | User VDF file for VIDriveSim event                                           |
| FDB-3804 | Suspension testrig angles setup                                              |
| FDB-3773 | Report file inertial attributes depends on std_tire_ref Z definition         |
| FDB-3772 | Wrong aerodynamic sensor position when internal aero is not active           |
| FDB-3759 | Automatic model validation bad curve matching with VI-Automotive models      |
| FDB-3743 | Suspension validation settings                                               |
| FDB-3742 | Loadcase for auxiliary compliance has wrong limits for pre-analyses          |
| FDB-3729 | Chassis dof offset                                                           |
| FDB-3726 | VI-Automotive models are wrongly exported                                    |
| FDB-3725 | Errors at the end of VI-Automotive models export procedure                   |
| FDB-3717 | Torsion stiffness limit in auxiliary subsystem is not mapped                 |
| FDB-3711 | Application remains hanged up randomly                                       |
| FDB-3707 | User tire tutorial                                                           |
| FDB-3662 | VI-DriveSim Event Wrong VDF path into _send.xml                              |
| FDB-3629 | Improving model export capability                                            |
| FDB-3585 | Multithread computation is not deterministic with curved RGR                 |
| FDB-3264 | Export VI-Automotive models keeping adjustments active                       |
| FDB-2813 | Random license issue running Maximum Performance event                       |

## 8.8.13 Release 16.1

| Versio n | Change ID | Change                                                                   |
|----------|-----------|--------------------------------------------------------------------------|
| 16.1     | -         | this version includes upgrades for compatibility with VI-Automotive 16.0 |

## 8.8.14 Release 16.0

| Versio n | Change ID | Change                                                                                |
|----------|-----------|---------------------------------------------------------------------------------------|
| 16.0     | FDB-3637  | Problems solving static equilibrium using advanced aeromap with sideslip dependency   |
|          | FDB-3633  | External engine solver is not correctly deallocated when performing simulink analyses |
|          | FDB-3628  | Abort time for trailer simulation is hardcoded to 100s                                |
|          | FDB-3622  | Suspension compliance is wrongly computed using external tire input from simulink     |
|          | FDB-3614  | Center differential does not behave according to the LSD specification                |
|          | FDB-3569  | Force graphics are wrong when std_tire_ref is modified                                |
|          | FDB-3539  | Incorrect processing of steering data for K&C import in case of redundant input data  |

## Release Notes

|          |                                                                                       |
|----------|---------------------------------------------------------------------------------------|
| FDB-3528 | Rack forces may be inaccurate for high jounce                                         |
| FDB-3521 | Prevent errors when models does not include body geometry                             |
| FDB-3505 | Trailer model 64bit does not run properly                                             |
| FDB-3494 | Improve stability of external powertrain example subroutine                           |
| FDB-3469 | Support for variable ratio in advanced steering EPS                                   |
| FDB-3452 | Problems running max performance event with external powertrain                       |
| FDB-3443 | Some parameters stored in the system tree may not be exported properly from Adams Car |
| FDB-3442 | Wrong units in steering angle channel in model validation report                      |
| FDB-3426 | Improve export process for auxiliary vertical stiffness computation                   |
| FDB-3423 | Prompt a warning for circular buffer activation mode                                  |
| FDB-3422 | Static Load Editor ignores changes                                                    |
| FDB-3413 | Incorrect conversion of pwr_scaling_factor parameter from v14                         |
| FDB-3356 | Problems running VI-CarRealTime on Matlab 2013b win64                                 |
| FDB-3390 | Wrong computed CG with SportsCar models                                               |
| FDB-3323 | Implementing an import procedure for CarSim models                                    |
| FDB-3314 | Static on 3d road could fail for models with inactive setup                           |
| FDB-3313 | Adams Car export doesn't write system xml                                             |
| FDB-3304 | MaxPerformance event GUI does not prevent access to initial gear field                |
| FDB-3302 | Export error for cos_ARB_force                                                        |
| FDB-3295 | Speedgen event disregards pwr_scaling_factor parameter                                |
| FDB-3274 | New Matlab library for accessing vehicle data                                         |
| FDB-3250 | Wrong computed tierod forces                                                          |
| FDB-3226 | Misuse event should stop when limit roll is reached                                   |
| FDB-3225 | Auxiliary subsystem status not properly saved.                                        |
| FDB-3217 | Simulink interface documentation reports incorrect channel names                      |
| FDB-3204 | VDF converter does not map yaw rate controller activation                             |
| FDB-3187 | Automatic validation report errors for SportsCar models                               |
| FDB-3182 | Automatic model validation errors                                                     |
| FDB-3165 | Automatic model validation fails when results file are disabled                       |
| FDB-3157 | VDF parameters definition in VI-DriveSim event                                        |
| FDB-3153 | Leftover files when uninstalling crt_plugin                                           |
| FDB-3135 | Could not run VI-driver event with std_tire_ref.psi =0                                |
| FDB-3134 | Road graphic is wrongly positioned when std_tire_ref data are changed                 |
| FDB-3124 | Roundoff error in two consecutive setup files savings                                 |
| FDB-3121 | Export errors when RES file is not selected in the model                              |
| FDB-3114 | Active flag not working for Static Loadcase editor                                    |
| FDB-3104 | Edit button not working for Static Loadcase File                                      |
| FDB-3098 | Wrong model inputs description                                                        |
| FDB-3089 | Road data save & restore before static phase                                          |
| FDB-3086 | Wrong definition in bushing torque expression                                         |
| FDB-3077 | Ride height maps not updated                                                          |
| FDB-3076 | TireTestrig does not save results in Matlab format                                    |
| FDB-3074 | TNO tire initialization fails with overlay 64 bit                                     |

|          |                                                                                 |
|----------|---------------------------------------------------------------------------------|
| FDB-3072 | MaxPerformance USE_TIRELIMITS flag always set to FALSE in mxp file              |
| FDB-3071 | Missing obj files when a system is saved with a different name                  |
| FDB-3059 | Updating Vehicle_Understeer_Stability_Factor definition                         |
| FDB-3057 | STI user tire example                                                           |
| FDB-3053 | User tire crt tutorial                                                          |
| FDB-3048 | vcredist_x64 not installed                                                      |
| FDB-3046 | Wrong compliance and auxiliary stiffness in Adams Car exported model            |
| FDB-3038 | Missing aerodynamic inputs for MATALB Simulink standard block                   |
| FDB-3036 | Plugin changes for MB-SHARC support                                             |
| FDB-3035 | Add throttle/brake to VI-Animator startup script                                |
| FDB-3034 | A/Car plugin export failure                                                     |
| FDB-3032 | 7post results file are missing                                                  |
| FDB-3031 | Performance issues on dSPACE                                                    |
| FDB-3017 | Support usage of Adams Ftire license for basic FTire computation mode           |
| FDB-2991 | Support for Adams 2013.2                                                        |
| FDB-2980 | Tire Testrig issue with call to VI-Animator                                     |
| FDB-2976 | Support for different layout of servo assistance force                          |
| FDB-2975 | Implement steering release event                                                |
| FDB-2974 | Improve fuel consumption capabilities                                           |
| FDB-2970 | Support cosimulation with FMU component                                         |
| FDB-2967 | Output as bus on Simulink interface does not work correctly                     |
| FDB-2964 | Vi-drivesim file event is always saved on root                                  |
| FDB-2962 | Phantom load using asymmetric antirollbar                                       |
| FDB-2961 | Cross weight setup failure on 7Post event                                       |
| FDB-2958 | Curb sled simulation failure with 64bit overlay installed                       |
| FDB-2948 | set SVM_GLOBAL_OUT_XML_FLG to 0.0 seems not work                                |
| FDB-2928 | Inconsistent rpm in speedgen                                                    |
| FDB-2927 | vicrt_passive.mdl model incompatibility with MATLAB 2012                        |
| FDB-2908 | Improve the definition of the graphic files                                     |
| FDB-2907 | Support for MF-Tyre 6.2                                                         |
| FDB-2898 | Introducing Skidpad setup                                                       |
| FDB-2897 | Different results if model has adjustments                                      |
| FDB-2890 | ISOLC tolerance error when using models that required setup                     |
| FDB-2886 | Support for FTire 2014-1                                                        |
| FDB-2861 | Adding a simple aux subsystem template for res file extension from simulink     |
| FDB-2858 | Problems with roads DB memory management                                        |
| FDB-2827 | Straight setup mode fails with external engine body                             |
| FDB-2815 | Speedgen is not passing computation and simulation modes to custom aerodynamics |
| FDB-2749 | Using flyingLap Mxp final maneuver and the same dynamic are different           |
| FDB-2704 | Moving ISOLC under mxp2012 framework - reduce computation time                  |
| FDB-2668 | Change working dir path to relative in send file                                |
| FDB-2615 | PAC2002 error management                                                        |
| FDB-2571 | Running multiple events may leads to unpredictable results                      |

*Release Notes*

|          |                                                                                             |
|----------|---------------------------------------------------------------------------------------------|
| FDB-2518 | Print a message while performing automatic road graphic generation                          |
| FDB-2312 | Add global inertia properties in report file                                                |
| FDB-2308 | Adding Ride height outputs to Static LapTime event                                          |
| FDB-2307 | Seven-post event including free up wheels option                                            |
| FDB-2306 | Control the significant digits written to xml files                                         |
| FDB-2270 | Improving error message for StaticLapTime Event                                             |
| FDB-2223 | Changing the format of the error message when no input file is specified (VDD or DM events) |
| FDB-2115 | Road Database Cleanup Disabled                                                              |
| FDB-1199 | Unexpected third spring behaviour                                                           |
| FDB-1132 | Could not remove custom plot pages in TireLimits application                                |



[www.vi-grade.com](http://www.vi-grade.com)  
email: [info@vi-grade.com](mailto:info@vi-grade.com)