755 Phoenix Drive, Ann Arbor MI, 48108, USA
Phone: 734 668-2930 • Fax: 734 668-2877 • Email: info@carsim.com
carsim.com

Simulations with Multiple Vehicles

Michael Sayers, Ph.D., Chief Technology Officer

ADAS scenarios often involve multiple vehicles and their interactions. If a scenario requires multiple vehicles, CarSim and TruckSim support several options to include the vehicles in a simulation. One of the options, introduced in version 2020.1, is to include multiple vehicles in a simulation handled by a single VS Math Model.

Classic Simulation with One Vehicle

CarSim and TruckSim are software tools for simulating the dynamic behavior of passenger vehicles, light-duty trucks, and heavy vehicles. They use 3D multibody dynamics models to accurately reproduce the physics of the vehicle in response to controls from the driver and/or automation: steering, throttle, braking, and gear shifting. Environmental conditions include a 3D ground/road surface as well as aerodynamic and wind effects.

The Windows versions of CarSim and TruckSim each include a VS Browser for running the models, setting parameters and tables, and viewing results. The VS Math Models are constructed with modules and supporting functions in VS Solver libraries — dynamic libraries that are loaded by the VS Browser or possibly third-party software — that run the simulation by continuously solving the equations in the math models. Results are viewed with VS Visualizer, a tool for providing high-quality animation of simulation results corresponding to engineering plots of hundreds (or sometimes thousands) of variables from the math models.

CarSim and TruckSim were originally developed in the 1990s to reproduce vehicle dynamics as obtained in proving ground testing and by measurements made on roads and highways. In addition to showing the motions of the vehicle with virtual video cameras, VS Visualizer shows the road surface and indicators such as force arrows to help visualize the dynamics of the vehicle during the simulation.

In most classic scenarios, there is a single vehicle represented with enough detail to reproduce the behavior of a physical vehicle undergoing the same test conditions. In the current architecture, a single VS Solver library has a collection of modules that are assembled for each simulation as needed to construct a VS Math Model for a vehicle of interest. Here are some of the modules supported by CarSim and TruckSim:

- Several types of sprung masses for the vehicle lead unit. These also have all of the driver control systems associated with the vehicle: steering, braking, powertrain, and closed-loop controllers.
- Several types of suspensions. These may be steered or unsteered, solid axle or generic/independent.
- Semitrailer sprung masses, with a hitch to connect to the vehicle unit in front. TruckSim supports trains of semitrailers; CarSim supports a single trailer.

In versions 2020.0 and older, each VS Math Model always had a single vehicle with a single lead unit and associated controls. The number of suspensions and trailers were assembled as needed by the model layout specified in the Browser GUI. TruckSim supports a large number of trailers and suspensions; CarSim requires an optional license to include trailers.

ADAS Simulation with One Full Vehicle Model

CarSim and TruckSim include options to add features to the basic vehicle model. Some features extend the vehicle, such as payloads, controllers (ABS, ESC, Speed Control), motion sensors, and user-defined forces and moments. Other features extend the simulation environment by adding moving objects, paths and road surfaces, and ADAS sensors that can detect the moving objects.

The moving objects are often used to enhance the visualization by showing other vehicle shapes, or points of interest on a target path, or other "actors" in a scene such as pedestrians and bicyclists. These optional components are added as needed. Some are either installed or not (e.g., speed controller), while other support multiple instances (e.g., payloads, sensors, moving objects). The ADAS Sensors require an optional licenses in order to be installed; all other optional components are included in a basic license.

Most ADAS scenarios include multiple moving objects to represent things that can be detected by an ADAS Sensor. These include signs, traffic signals, other vehicles, pedestrians, animals, and bicyclists.

Each ADAS sensor can potentially detect every moving object that is included in the simulation. Each sensor-object detection combination is represented with 24 output variables that can be exported to third-party software or used in VS Commands to provide ADAS interventions.

In a typical ADAS simulation, there is a single "ego vehicle" represented by the full multibody model assembled in the VS Math Model. Other actors are represented with moving objects. The moving object appears in VS Visualizer with whatever 3D asset is assigned to it, which might be a vehicle shape, or an animated person, animal, or bicyclist. However, within the VS Math Model, the object has a simple fixed shape, such as a circle, rectangle, or polygon. Options for moving the object include:

- Specifying the X, Y, Z coordinates and Yaw, Pitch, and Roll angles with imports from external software or with VS Commands.
- Specifying location and heading angle using an existing VS Path.
- Specifying elevation and orientation (pitch and roll) using an existing VS Road surface.

- Specifying speed with imports or VS Commands, and using numerical integration to get position (either path location or global X and Y).
- Specifying acceleration with imports or VS Commands, and using numerical integration to get velocity and position.
- Specifying Yaw/heading relative to a path.
- Using an "offtracking" option to automatically add a differential equation so the object is oriented as if a rear axle center always follows a front axle center, as seen in the low-speed tracking behavior of a vehicle.

The above options have been used in many examples to represent other vehicles in a simulation. For example, the simulation shown in Figure 1 has a single full ego vehicle model (blue vehicle with yellow force arrows). All of the other vehicle shapes shown are attached to moving objects under acceleration control, with offtracking. For example, the white sedan behind the ego vehicle shows realistic offtracking of the rear wheels being close to the curb while making a 90° turn.

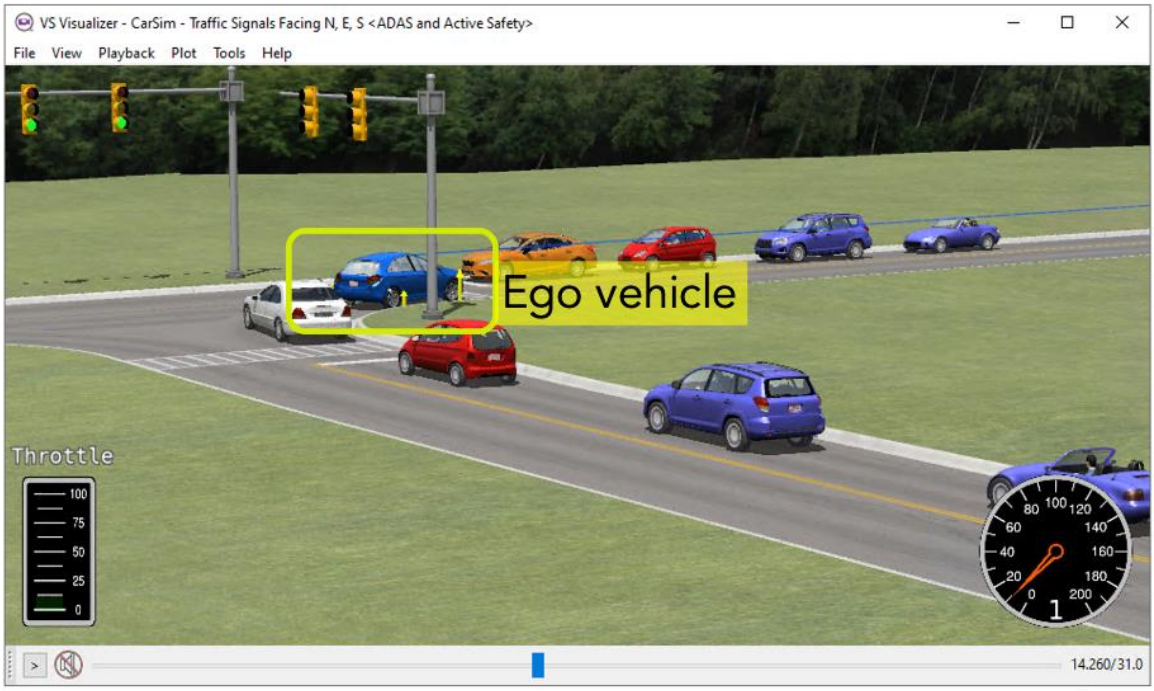

Figure 1. Simulation with one ego vehicle (full model) and moving object traffic vehicles.

This method works well for some situations, but there are limits. There are no wheel motions or suspension deflections. Movements based on defined acceleration follow a basic physics model, but cannot handle inputs such as throttle, steering, braking, and interactions with 3D ground features such as curbs.

The computational speed depends on how many ADAS sensors and targets are defined. In most cases, the extra time needed to run a simulation with the targets and sensor much less than the time needed for the vehicle by itself. That is, the simulation might take 30% longer to run, but would still be much faster than the time that would be needed of the moving objects were replaced with full vehicle models.

Running Multiple VS Math Models in Parallel

A VS Solver is a dynamically loaded library (DLL) file (on Windows, it is a .dll file, on Linux an .so file). This architecture allows the VS Solver to be used by programs other than the VS Browser. The VS Math Model can be constructed and run from MATLAB Simulink using an S-Function that acts as a "wrapper" that loads the solver DLL; it can be run from LabVIEW the same way; it can be run from a Functional Mockup Interface (FMI) environment if included in a Functional Mockup Unit (FMU) file. These options are supported by any CarSim or TruckSim Windows package.

Multiple vehicles can be simulated under the control of Simulink, LabVIEW, FMI, and other third-party simulation tools by loading copies of the original VS Solver file. This approach has been supported for at least 15 years (in Simulink), and was simplified in the 2020.0 version with the introduction of the **Parallel VS Math Models** tool in the VS Browser (originally named "Multiple Vehicles" and renamed for 2020.1 and again for 2021.0).

For example, Figure 2 shows two CarSim vehicles running together under Simulink, as set up using the **Parallel VS Math Models** library tool. The two blue links in the GUI screen (1 and 2) are connected to **Run Control** datasets that set up independent run conditions for each vehicle. Both **Run Control** datasets identify the same Simulink model, which has corresponding blocks for each vehicle (3 and 4).

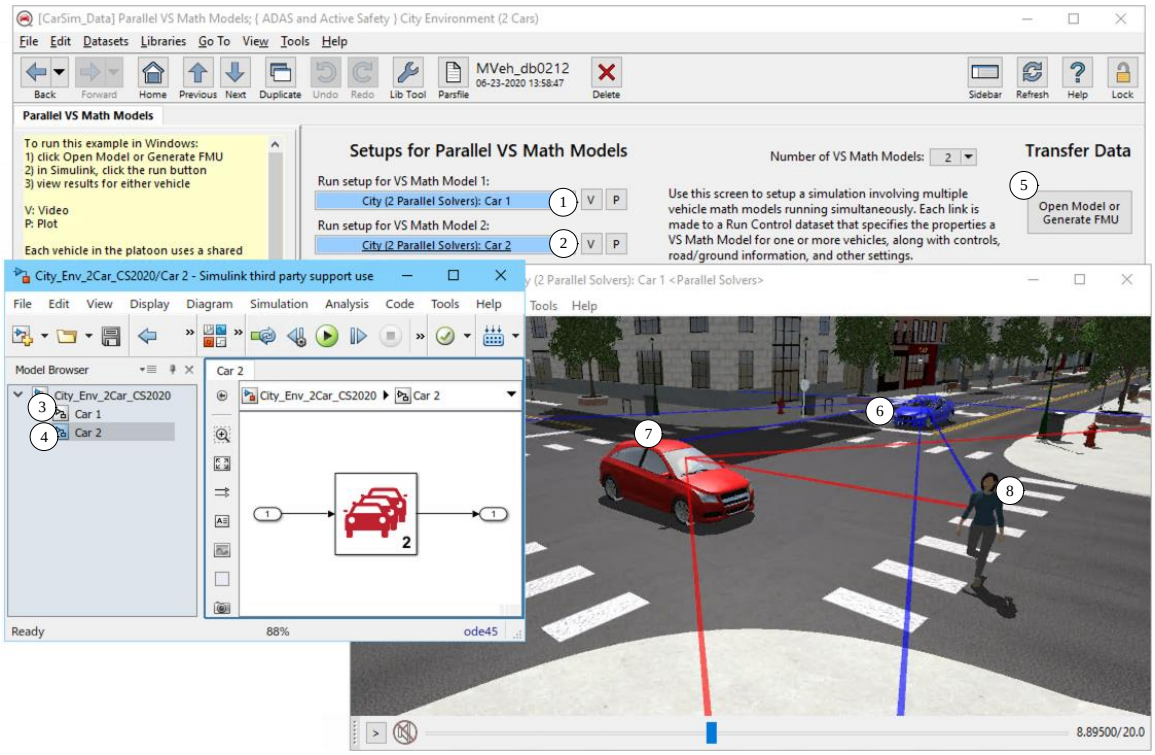

Figure 2. Two CarSim vehicle models running together under Simulink.

The CarSim **Parallel VS Math Models** screen has a button to **Open Model or Generate FMU** (5). When it is clicked, the Browser does several things:

1. If there are two or more links to **Run Control** datasets, it copies the solver .dll file and renames the copies, such that each vehicle will run from a unique solver file.
2. It visits each **Run Control** screen and updates the two files that are needed to start the simulation (the *Simfile* and the *All Parsfile*).
3. It opens the Simulink model and uses Windows COM to transfer simulation settings, such as time step.

The simulation can then be run from Simulink. Results are viewed by clicking the **V** or **P** buttons on the **Parallel VS Math Models** screen (1 or 2) to view the results. In this example, Car 1 is the blue vehicle 6, and car 2 is the red vehicle 7.

In order to run a simulation such as this, there are several details that must be set up manually:

1. If the vehicles are going to detect each other, each **Run Control** setup must include an ADAS sensor for the vehicle, plus a moving object to serve as a target that follows the motions of the other vehicle. For example, the setup for the blue vehicle includes a target whose location will be updated each time step to match the location of the red vehicle.
2. Each **Run Control** setup must define export variables that include the six coordinates and angles that define the instant position of the ego vehicle sprung mass. For example, the setup for the red vehicle provide variables that are exported to Simulink and will be imported by the blue vehicle to locate the target moving object.
3. Each **Run Control** setup must define import variables that include the six coordinates and angles that define the instant position of target. For example, the blue vehicle imports six variables to locate the target to match the sprung mass of the red vehicle.
4. Each **Run Control** setup must define the same terrain, either with the same VS Terrain file or with the same set of road surfaces.
5. If there are other targets that can be sensed by an ADAS sensor, they should be included in each **Run Control** setup. In the example, the buildings on the intersection are detected by both vehicles; that was achieved by including the same target objects in both setups. Also, the walking pedestrian 8 was defined in exactly the same way in both setups, such that both vehicle sensors can detect the same pedestrian.

This approach can be extended to include more vehicles. If there are more vehicles, then each run setup must include targets for all of the external vehicles as described in items 1 – 3 above.

The time needed to run a simulation this way is the time needed to run one vehicle plus targets by itself, plus the times needed to run each other vehicle (plus targets), plus some overhead in running with the third party tool. For example, the time needed to run a simulation with two vehicles is greater than twice the time need to run with one vehicle.

A standard CarSim or TruckSim license supports the use of two VS Math Models running in parallel (each from a separate VS Solver library); if three or more VS Solver libraries are used, additional Parallel Solver licenses are required.

Running Multiple Vehicles from One VS Math Model

As noted earlier, the CarSim and TruckSim math models are modular, with modules for vehicle parts such as sprung masses, suspensions, and control systems. Going back to version 2018.0, the solvers will assemble a vehicle model based on layout information provided by the input *Simfile* and *All Parsfile*. Starting with version 2020.1, the modular approach was extended to control systems such as steering, braking, powertrain, and closed-loop controllers such as the driver model path follower and speed controller. This allows the solver to build a model that includes multiple vehicles, each with independent control options.

For example, Figure 3 shows a simulation similar to the one shown in Figure 2 that used Simulink. In this case, no third-party software is used. The simulation is made completely from within CarSim, and includes four vehicles.

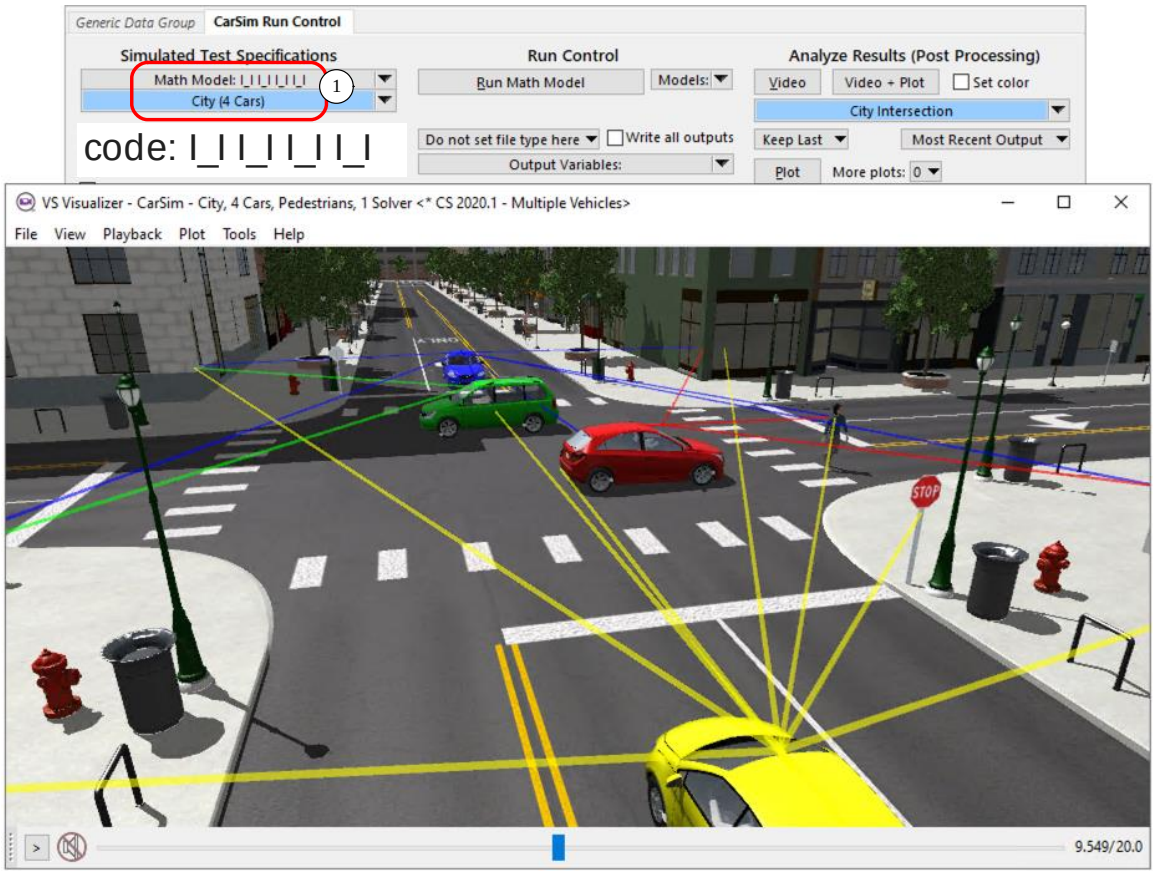

Figure 3. Four vehicles interacting in CarSim 2020.1 without external software.

This is an advanced setup that requires the use of Generic libraries. Instead of linking to a vehicle dataset in the top-left link (1), this **Run Control** dataset links to a **Generic Data Group** dataset that specifies a vehicle code `I_I I_I I_I I_I`. The spaces in the code indicates that this setup is made for four vehicles, each with configuration `I_I`.

Figure 4 shows the generic dataset that specifies the model layout code (1), and then four potential groups of settings, one for each vehicle. The first group has a link to the first vehicle (2) and a link to a **Procedures** dataset (3). The remaining three vehicles are each set up with three controls: a link

to the vehicle and its sensor (4, 7, and 10), information in a miscellaneous yellow field specifying where the vehicle starts and which path it follows (5, 8, and 11), and a link to a **Procedures** dataset with control options controls (6, 9, and 12).

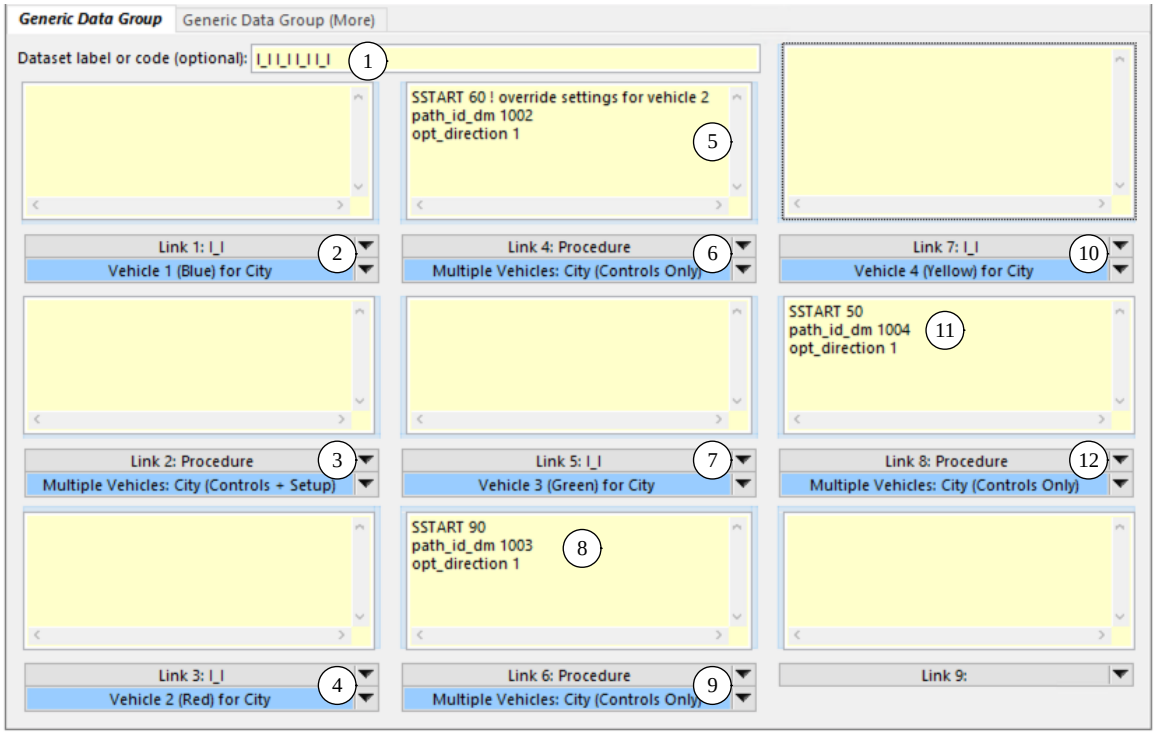

Figure 4. Generic Data Group to set up simulation with four vehicles.

All four of the vehicle links go to datasets in the **Generic Data Group (More)** library. The settings for the first vehicle include both information specific to that vehicle, plus information used by all vehicles in the simulation (Figure 5).

This dataset shows the vehicle code for the selected vehicle 1. (This is done so the code appears wherever a link is made to the dataset, e.g., 2 in Figure 4). The first miscellaneous yellow field 2 has a command to set the color for the vehicle (for use by VS Visualizer), and to make a global setting involve minimum requirements when new sensors are defined.

Datasets similar to the one shown in Figure 5 are used for all four vehicles, with the exception that only vehicle 1 includes the Group of VS Commands link 6.

The first link 3 goes to a vehicle dataset. This can be any configuration; however, the code shown should be repeated in the code field 1.

Note	The solver functions in CarSim and TruckSim will automatically change the suspension code "`Ind`" to '`I`', and the suspension code "`SA`" to '`S`'. The shorter versions may be used to conserve space (and reduce the amount of manual typing).

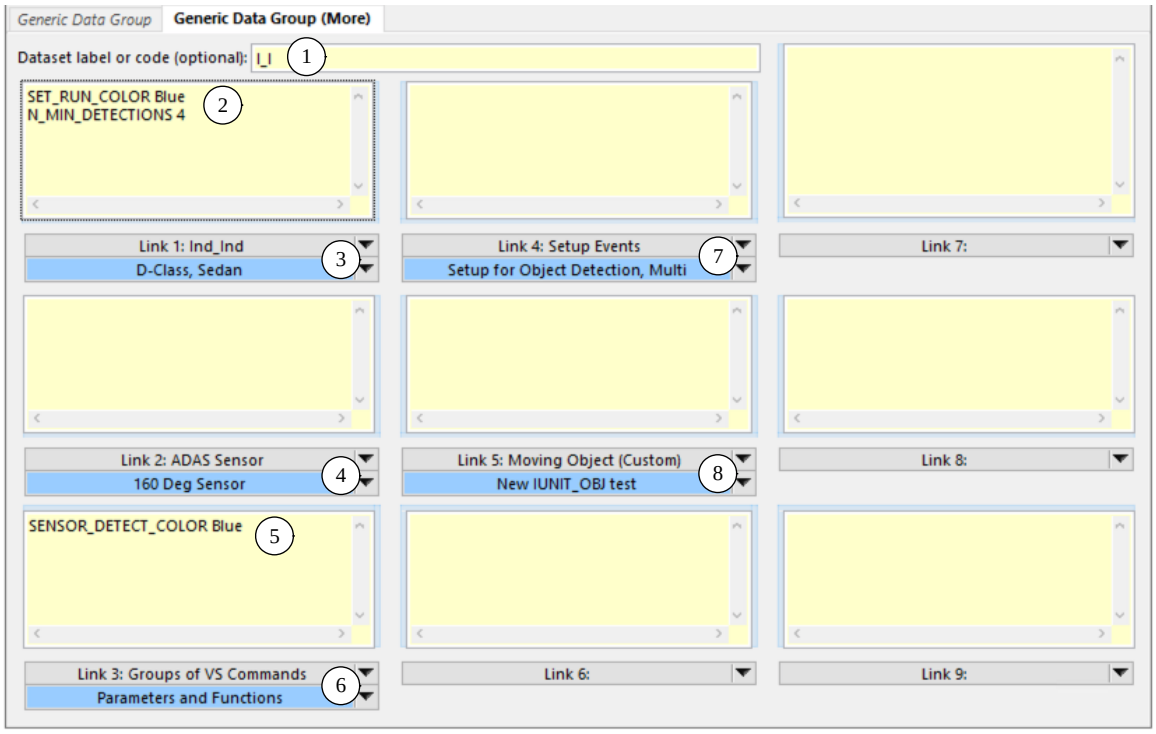

Figure 5. Data for the first vehicle, plus sensor and target, plus an Event setup dataset, plus simulation setup data.

The second link attaches an ADAS sensor to the vehicle sprung mass (4). On the **ADAS Sensors for Range and Tracking** screen (Figure 6), the vehicle part for attaching the sensor is set to the sprung mass of the current vehicle unit (i.e., the most recently defined vehicle or trailer).

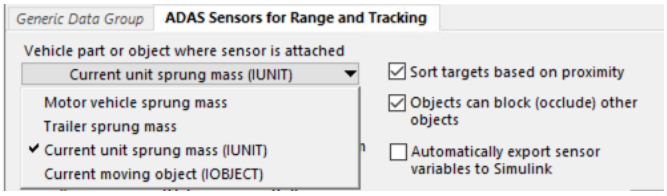

Figure 6. Attaching a sensor to the sprung mass of the current vehicle unit.

The color for the animated sensor detection lines is set in the next miscellaneous field (5). In this example, the same Sensor definition is used for each vehicle. In each case, the color is set to match the color set for the vehicle, as shown here.

The next link (6) provides definitions for the simulation that are independent of the number of vehicles. The linked dataset defines a number of parameters used to identify targets (stop signs, pedestrians, vehicles, etc.). It also defines a set of custom functions that are used in VS Commands associated with each vehicle. As noted earlier, this link is only used for vehicle 1. The VS Commands and custom functions are used in other VS Commands that are specific to each vehicle, provided with the next link (7).

The final link is to a Moving Object dataset (8). All four vehicle setups in this example make use of the same dataset, from the **Single Moving Object (Custom)** library (Figure 7).

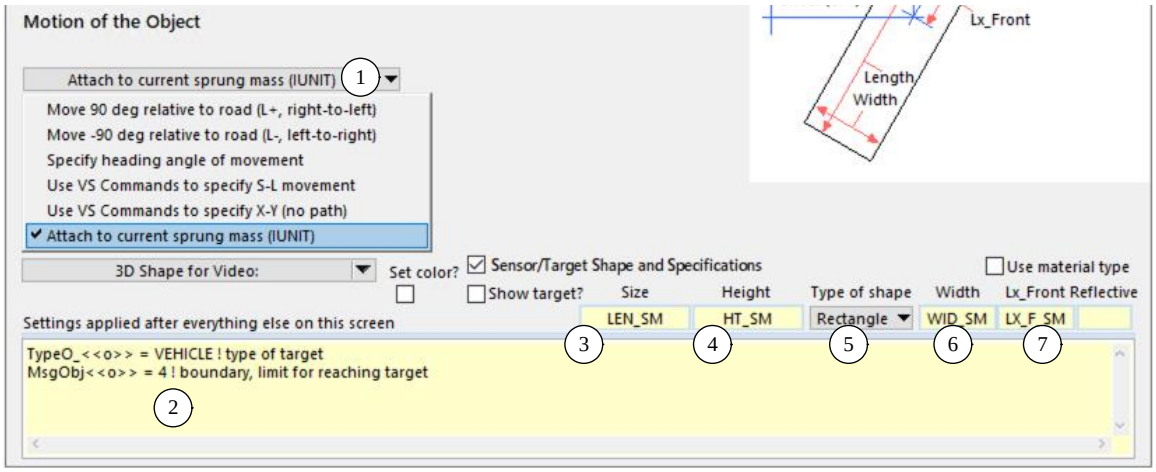

Figure 7. The Single Moving Object (Custom) screen can easily connect to a sprung mass.

The drop-down control has an option to connect to the current sprung mass (1). When selected, it sets the target parameter `IUNIT_OBJ` to `IUNIT`, which is the current vehicle unit number. This is intended to be used from a vehicle assembly screen, a vehicle loaded combination screen, or after a vehicle link (as in Figure 5).

In this example, the dimensions of the target object are assigned to parameters associated with the current sprung mass (identified by `IUNIT`). If these were set on the sprung mass screen, then the target object is automatically configured to have the proper dimensions, and is assigned to follow the origin of the sprung mass coordinate system as it moves.

Recall that the datasets for each vehicle (Figure 5, page 8) include a link to a set of VS Commands that are specific to that vehicle. These datasets are very similar for the four vehicles in this example. Consider the case of vehicle 3, whose set of VS Commands is shown in Figure 8.

The first set of commands (1) defines new parameters, including `S3` — the ID of the sensor that was most recently defined. Notice that all of the parameters and outputs (3) end with the digit '`3`'. Nearly identical datasets were also created for the other vehicles in this example. Variables for the first vehicle have no suffix, and the other two (vehicles 2 and 4) use the suffixes '`2`' and '`4`', respectively.

A link at the bottom of the screen (3) connects to an **Events** dataset designed to work with a specific vehicle, in this case, vehicle 3 (Figure 9).

The specification of vehicle number in the top yellow field (1) causes the line `IVEHICLE 1` to be written at the top of the Parsfile for the dataset. This provides a context for any controls that are set in the top part of the screen, such as throttle (2). When multiple vehicles are handled by a single VS Math Model, each has its own throttle (and other systems related to vehicle control, such a braking, steering, powertrain, etc.). In this case, the specified throttle is set for vehicle 2; the other vehicle throttles are unaffected.

In the same manner, the speed controller `OPT_SC_ID` is set for the controller in vehicle 3, as are the other parameters named in the field (3).

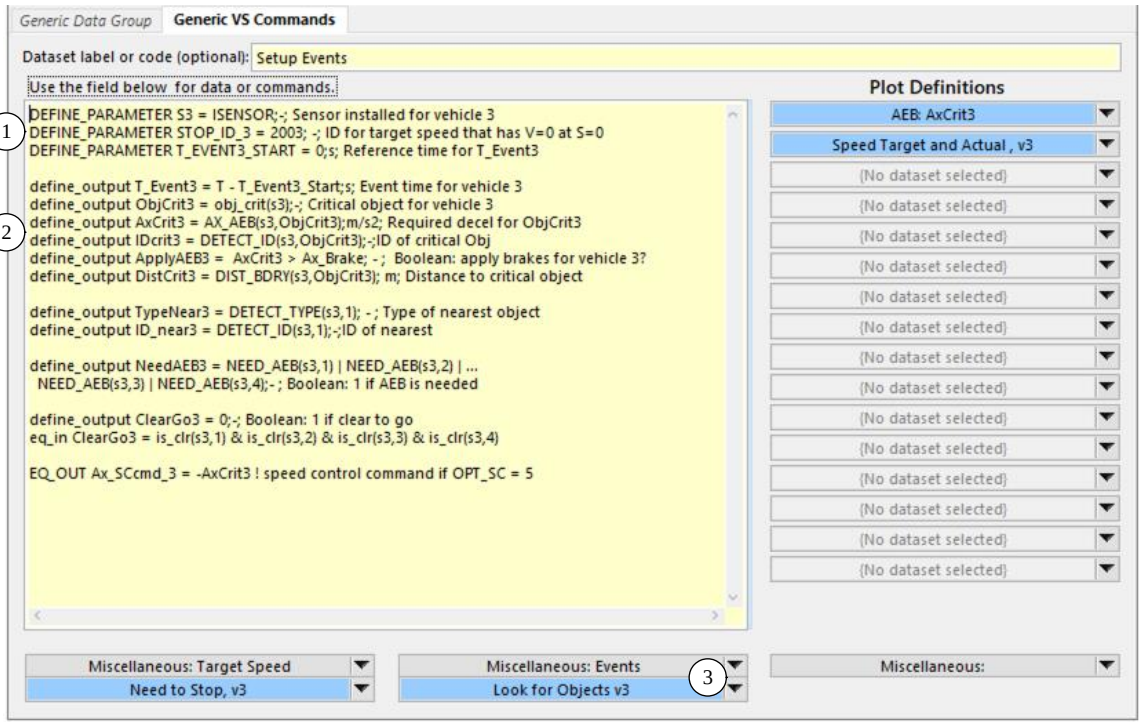

Figure 8. VS Commands to setup Events for vehicle #3.

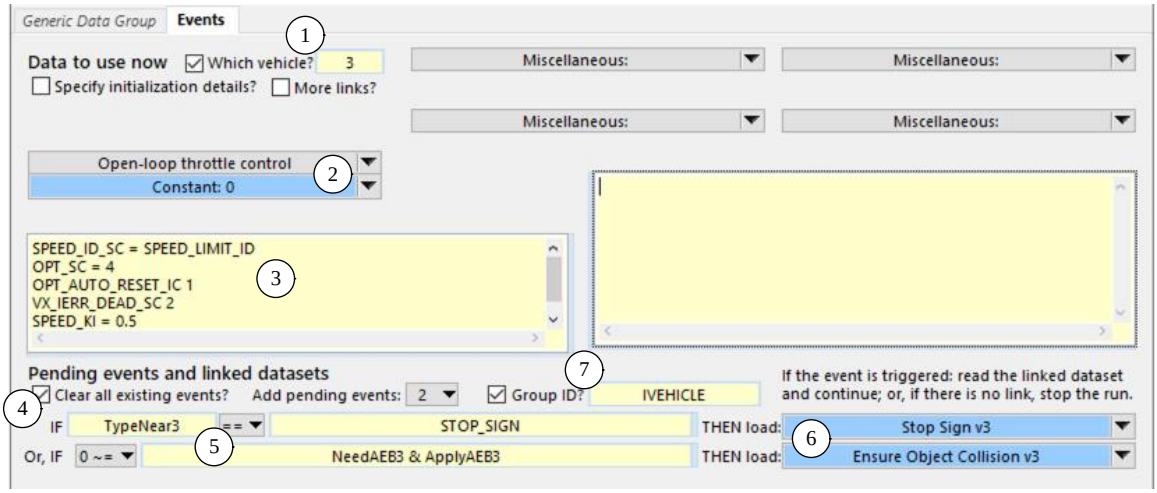

Figure 9. Startup Event dataset that defines a set of pending events for vehicle 3.

Two pending Events are defined in this dataset, with both involving outputs created specifically for vehicle 3 (5). The datasets that will be read if an Event is triggered are also created specifically for vehicle 3 (6).

The **Group ID** for this dataset is set to `IVEHICLE` (7), which we know is 3. The checkbox to **Clear all existing events** is checked (4). Because the **group ID** is set to `IVEHICLE` (3), only Events with this ID are cleared. This helps ensure that Events set up for one specific vehicle do not interfere with Events set up for other vehicles running from the same Solver.

For example, Figure 10 shows the end of the Log file made for the example simulation made with four vehicles. Each line reporting an Event identifies the group. E.g., lines 836-841 report that Event #33 from Group 2 (i.e., vehicle 2) triggered based on a condition involving the variables `t_event2` and `AxCrit2`, that another Event was removed from the list of pending Events (line 838), and that two new datasets were processed (lines 840 and 841).

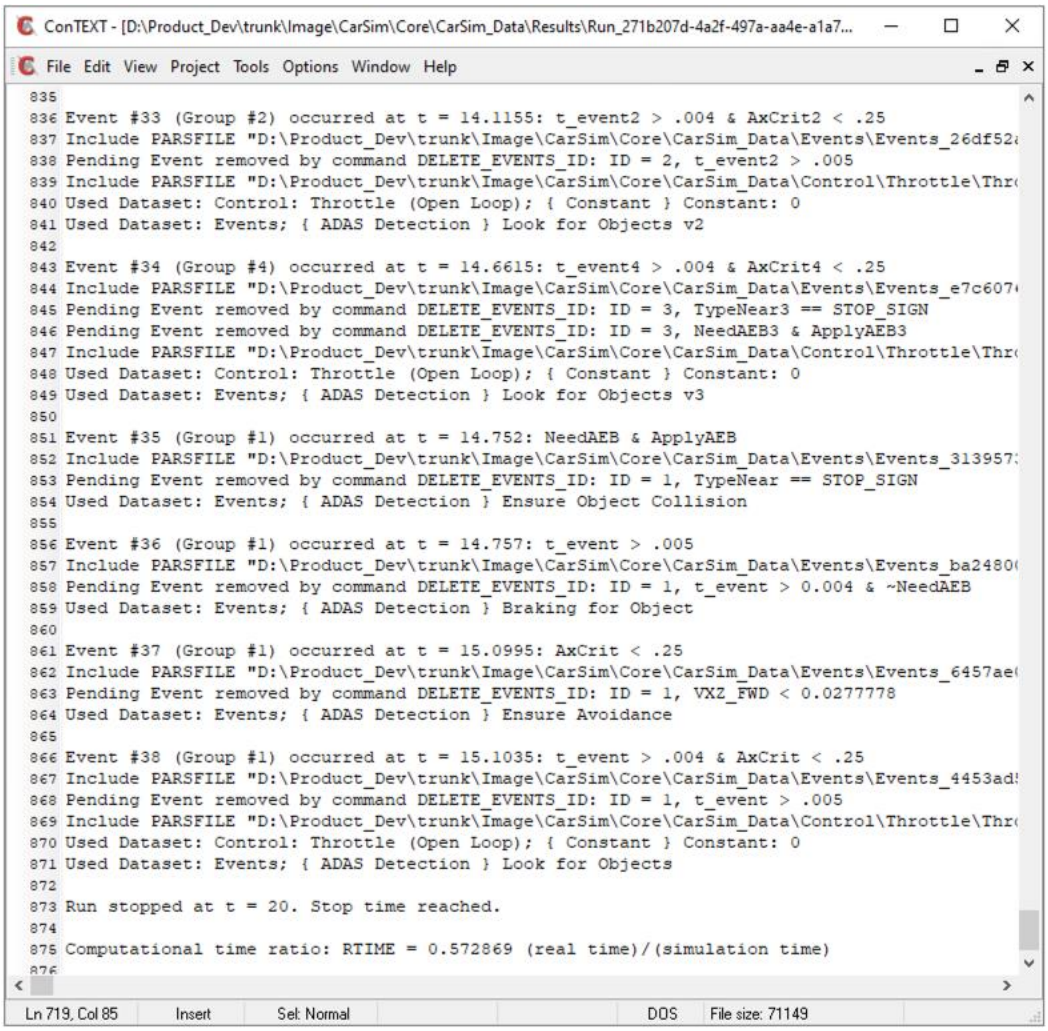

Figure 10. End of Log file for simulation including four vehicles in one solver.

Note also that the time ratio for the example is 0.57 (line 875), meaning that the 20s simulation took about 11s to run.

An advantage of having both vehicles running from the same solver is that the road environment is only defined once, as are target objects other than the vehicles themselves. Both vehicles detect the same building, walking pedestrian, and bicyclist (out of view in the figures).

In order to run multiple vehicles from the same solver, an optional Multiple Vehicle license is needed; it enables a total of four vehicles to be run from the solver.

Comparison of Options

The preceding three sections described three methods for simulating ADAS scenarios with multiple vehicles. Table 1 summarizes the technical advantages and disadvantages of each.

Table 1. Comparison of three methods for simulating multiple vehicles.

Method	Advantages	Disadvantages
One full vehicle model with moving object targets used for traffic vehicles	• Easy to set up • Runs fast • Many examples • Does not require third-party software	• Motions are limited to simple physics and offtracking • Controls are limited • Hard to make reactive vehicles
Multiple solvers running in parallel from third-party software	• Users may be familiar with third-party tool • Has been available a long time • All vehicles have full control capability • Supported by **Parallel Solvers** screen • Separate vehicles may use same set of Event datasets • No extra license needed for two vehicles • Supports up to 20 solvers • Advanced users can use multiple core CPUs	• Requires third-party tool • All runs must be set up identically for ground, environment • Requires careful mapping of import and exports for targets • Requires VS Commands to match target motions to external vehicles • Can run slowly (more overhead) • Optional Parallel Solvers license needed if more than two solvers are used
Multiple vehicles running from one solver	• Easy to make each vehicle visible to the others • Same terrain is used by all vehicles • Same target moving objects are visible to all vehicles • Runs efficiently • Third-party software not essential, but is supported	• Users need to make custom setups • Must match ADAS responses to specific vehicle • If using Events, need separate sets for each vehicle • Optional license needed for multiple vehicles • Limited to four vehicles • Limited to one core CPU

The first option, using moving object targets for other vehicles, is the most convenient if the additional vehicles are intended to behave passively, without responding to the simulated ego vehicle.

If the intent is to simulate multiple vehicles that are all capable of reacting to the situation, then the Parallel Solver or Multiple Vehicle methods may be used. The choice between these two options involves trade-offs, as summarized in the table.