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

Procedures and Events

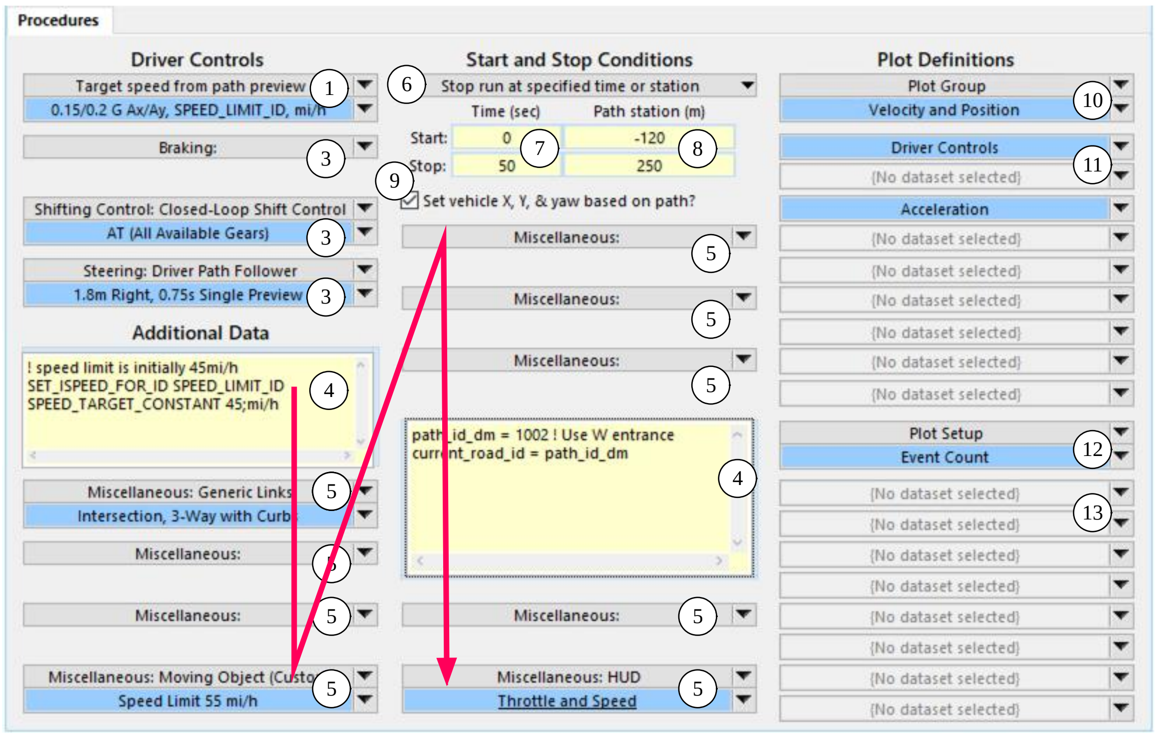
Two library screens provide most of the details of a simulated test procedure in VehicleSim products.

- A **Procedures** dataset assembles information about a simulated test procedure, with driver controls, starting and stopping conditions, associated plot descriptions, and other information that might be associated with a type of procedure. Most **Run Control** datasets will use a **Procedures** dataset for the second link (under the Math Model link).
- An **Events** dataset defines one or more conditions under which the test procedure and/or the simulated vehicle can be changed while a simulation is in progress. **Events** datasets may be linked from many locations in the database.

Both libraries are found in the category **Procedures** under the **Libraries** menu.

The Procedures Screen

The **Procedures** screen (Figure 1) assembles conditions such as driver controls (steering, braking, throttle, etc.), road conditions, and other datasets that you might use to define a test. It can provide all the information to set up a run for any potential vehicle.


Procedures

Driver Controls
Target speed from path preview
0.15/0.2 G Ax/Ay, SPEED_LIMIT_ID, mi/h
Braking:
Shifting Control: Closed-Loop Shift Control
AT (All Available Gears)
Steering: Driver Path Follower
1.8m Right, 0.75s Single Preview

Additional Data
```
! speed limit is initially 45mi/h
SET_ISPEED_FOR_ID SPEED_LIMIT_ID
SPEED_TARGET_CONSTANT 45;mi/h
```
Miscellaneous: Generic Links
Intersection, 3-Way with Curb
Miscellaneous:
Miscellaneous:
Miscellaneous: Moving Object (Custo
Speed Limit 55 mi/h

Start and Stop Conditions
Stop run at specified time or station

	Time (sec)	Path station (m)
Start:	0	-120
Stop:	50	250

☑ Set vehicle X, Y, & yaw based on path?
Miscellaneous:
Miscellaneous:
Miscellaneous:
```
path_id_dm = 1002 ! Use W entrance
current_road_id = path_id_dm
```
Miscellaneous:
Miscellaneous: HUD
Throttle and Speed

Plot Definitions
Plot Group
Velocity and Position
Driver Controls
{No dataset selected}
Acceleration
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
Plot Setup
Event Count
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}
{No dataset selected}

Figure 1. The Procedures screen.

Driver Controls

(1) Speed / Acceleration option. This drop-down list has options for controlling the vehicle speed (Figure 2). If the selected option implies that more information is needed, then a yellow field and/or blue link are also shown.

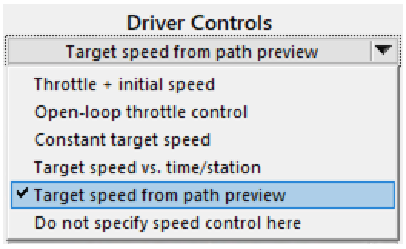

Driver Controls
Target speed from path preview
Throttle + initial speed
Open-loop throttle control
Constant target speed
Target speed vs. time/station
✓ Target speed from path preview
Do not specify speed control here

Figure 2. Drop-down list of speed control options.

1. If closed-loop throttle control is selected for a target speed that is not constant, then a link is shown for connecting to a dataset in the library associated with the selection (e.g., **Target speed from path preview**).

2. If an open-loop control is selected with an associated initial speed, then a yellow field is used to set the initial speed and a link is shown for connecting to an open-loop throttle control dataset.

3. If a **Constant target speed** is selected, then a yellow field is shown for specifying that speed, along with a checkbox (2) (Figure 3).

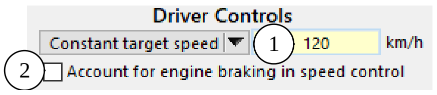


Figure 3. Appearance when a constant target speed is selected.

Three of the options involve the use of the built-in Closed-Loop Speed Controller. For the two options that begin with the words "Target speed," a blue link is shown to link to a dataset from the appropriate library (Figure 1). The linked dataset specifies the target speed using various table options such as Constant or Linear interpolation and extrapolation, and possibly parameters for the speed controller. The controller can be tuned using feedback parameters (proportional, integral, and cubic gains), and the brake system may also be included (e.g., if trying to maintain a constant speed going down a hill).

The library for Target Speed vs. Time and Station provides datasets for the built-in Configurable Function `SPEED_TARGET` that calculates a target speed. The target speeds from this library may also be used with traffic vehicles. To avoid conflicts when using multiple `SPEED_TARGET` datasets, the screen for the `SPEED_TARGET` library has a checkbox to show or hide the controller feedback parameters. When linking to a `SPEED_TARGET` dataset for use by the simulated ego vehicle, be sure to choose one that also specifies parameters for the speed controller.

A VS Math Model supports up to 200 `SPEED_TARGET` datasets specifying target speed as potential functions of time and/or station along a path. Datasets are added to the model as needed by the Parsfiles associated with the linked datasets from the speed control libraries.

2 Checkbox to account for engine braking in the speed controller. If the option is set to **Constant target speed**, then a yellow field is shown for the speed, plus this checkbox (2, Figure 3) to specify whether the controller will account for engine braking. In this case, the speed controller gains and brake system performance parameters will retain their default settings (e.g., `OPT_BK_SC = 0`). With the Constant target speed option, a new `SPEED_TARGET` dataset is automatically added to the model, with the specified value for speed assigned using the `SPEED_TARGET_CONSTANT` command for the new datasets. If this is the first reference to a `SPEED_TARGET` dataset, the index will be 1, and the speed will appear in the Echo file with the keyword `SPEED_TARGET_CONSTANT(1)`. However, if target speed datasets are specified before the **Procedures** dataset (e.g., `SPEED_TARGET` datasets might be linked to ADAS Sensors associated with the vehicle), then the index will be higher than 1. With the Constant target speed option, a command "`OPT_SC 3`" is also written to enable the speed controller.

Note For more information regarding the settings for the Closed Loop Speed Controller, view the Echo file in the section `DRIVER MODEL: SPEED CONTROLLER`. Information regarding the `Speed_Target` datasets are in the *Configurable Functions* section. The Echo file can be viewed using the **View** button in the lower right corner of the **Run Control** screen.

3 Links to datasets for braking, shifting, and steering.

Additional Data

Additional links and fields are used to specify more information about the simulation conditions. These may include the path whose `PATH_ID` matches `PATH_ID_DM`, the road surface identified with the parameter `CURRENT_ROAD_ID`, model extensions made with VS Commands, Moving Objects, and VS Event datasets (described in the next section, page 8).

④ **Miscellaneous yellow data fields**. These two fields may contain any text that would be recognized by the VS Math Model. They can be used to assign new values to any of the hundreds of parameters in the model. (To see the parameter names, go to the **Run Control** screen and use the **View** button in the lower-right corner to view an Echo file.) The format for each line of text should consist of a parameter name, then a blank space, and then the parameter value or a formula. For more detail, please see *VS Math Models Manual* and *VS Commands Manual*.

⑤ **Miscellaneous** links. These links can be made to datasets from many of the VS Browser libraries. Some popular links are VS Commands, VS Events, road conditions, and extra animation settings such as animated arrows to show tire forces. If you add animation settings, please be aware of the additive effects of multiple specifications of animation settings, as described in the section **Overriding Data** on page 26.

The data associated with the Miscellaneous links and data fields are written into the Parsfile top to bottom, left to right, as indicated by the zigzag red line (Figure 1, page 2). This is the order in which they are provided to the VS Math Models. Advanced users may apply this knowledge to coordinate the data from the fields ④ and links ⑤. For example, parameters can be defined in the first Miscellaneous data field so they can be referenced in linked datasets; alternatively, parameters defined in linked datasets can be overridden with new values in the second Miscellaneous data field.

Start and Stop Conditions

Most simulations specify a reference path with the parameter `PATH_ID_DM` that may be used by the built-in closed-loop steering controller. Even if the steering controller is not used, the path is still used as a reference to define a few output variables, such as `Station`. If not specified, the `PATH_ID_DM` parameter is automatically set to the ID of the first path.

⑥ Drop-down control for showing start and stop options. This list has five options (Figure 4). Based on this selection, appropriate controls are displayed below (⑦ and ⑧, Figure 1 and Figure 5).

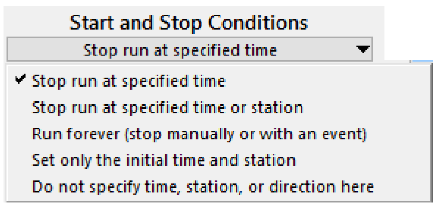

Start and Stop Conditions
Stop run at specified time
✓ Stop run at specified time
Stop run at specified time or station
Run forever (stop manually or with an event)
Set only the initial time and station
Do not specify time, station, or direction here

Figure 4. Options for setting start and stop conditions.

The first three options in the list set a value for the system parameter `OPT_STOP`; the values set are:

1. 0 (stop at specified time), or
2. +1 (stop at time or station), or
3. -1 (run forever)

For the second option, fields are shown for both start and stop positions (Figure 5).

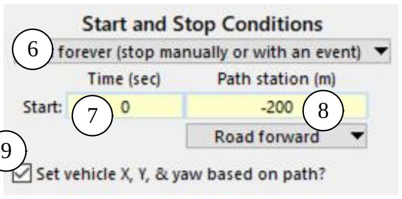

Start and Stop Conditions
6
forever (stop manually or with an event)
Time (sec)
Path station (m)
Start:
7
0
-200
8
Road forward
9
Set vehicle X, Y, & yaw based on path?

Figure 5. Option to stop at specified time or station.

With all three options, data fields are shown for the start time ⑦ (keyword = `TSTART`) and starting station ⑧ (keyword = `SSTART`) along the path with ID `PATH_ID_DM`. The direction the vehicle is travelling relative to the path is determined either by comparing start and stop station values (Figure 1), or by setting it directly with a drop-down control with options for forward or reverse (Figure 5) (keyword = `OPT_DIRECTION)`.

The fourth and fifth options do not specify the mode for stopping or the initial direction. They are available for advanced users who will set the parameters elsewhere.

The fourth option (**Set only the initial time and station**) shows the start time and station fields only ⑦ and ⑧). The direction and stop conditions are not specified using the GUI. (If a stop time is not specified for the run, the default value for `TSTOP` (10 seconds) is used).

The last option (**Do not specify time, station, or direction here**) hides the related settings and has no effect on the simulation.

⑦ Simulation Time starting and stopping values (keywords = `TSTART`, `TSTOP`). When a run is started, the time is set to the value specified here. The start time is normally zero, but other values may be used.

Note Some real-time systems require that the start time always be zero. If you are running with one of these systems, the start time will be ignored. (Please see the **Help** menu for information about your RT system.)

1. A stop time is specified when one of the first two options is selected from the drop-down control ⑥.
2. If the **Run forever** option is selected, the field for setting stop time is hidden and a value of `TSTOP = -1` is automatically written in support of external programs such as Simulink that use a negative value to indicate the simulation should run until stopped by some custom condition. The Echo file will not show `TSTOP`.

3. If either the fourth or fifth options are selected, the field for the stop time is hidden and nothing is written into the screen Parsfile with the keyword `TSTOP`. If not set elsewhere, the default value (10s) is used.

(8) Initial location and direction of the vehicle on the reference path whose `PATH_ID` matches the parameter `PATH_ID_DM`. Locations on paths are defined by station (longitudinal distance along the path).

Every simulation includes at least one path. If none are specified, a default is created: a straight line oriented such that station = global X and lateral coordinate = global Y. (For details on Reference Paths, please see the document: *Paths and Road Surfaces* from the submenu **Help > Paths, Road Surfaces, and Scenes**.)

All of the Path libraries have the convention that each of their Parsfile sets `PATH_ID_DM` to the path when it is defined. Thus, the last one that is scanned before the simulation starts is the one used. If this is not your intent, then you can set `PATH_ID_DM` using a Miscellaneous field that will be scanned after all of the Path datasets (e.g., the Misc. yellow field in the center of the **Procedure**s screen, or the Misc. yellow field at the bottom center of the **Run Control** screen, visible after checking the **Advanced settings** checkbox).

The starting location (keyword = `SSTART`) is shown for all options of the control (6) except the last if the checkbox (9) is checked (**Set X, Y, and Yaw based on path)**.

When the option is chosen to stop at a specified time or station, then a field is shown for the stopping station (keyword = `SSTOP`). If the `SSTOP` field is shown, the `SSTART` field is also shown, even if the checkbox (9) is not checked. (This is done to determine the direction of travel relative to `SSTOP`.)

If the choice is made to stop at a specified time or to run forever, a drop-down control is shown to specify the direction of travel (Figure 5, keyword = `OPT_DIRECTION`).

The direction **Road forward** means the vehicle is oriented in the direction of increasing station (`OPT_DIRECTION` = 1); the direction **Road reverse** means the vehicle travels in the direction of decreasing station (`OPT_DIRECTION` = -1).

Note If station is defined using a Reference Path that is looped, then the length of the loop limits the range of valid station values. The concept of using a value of station to stop the run does not always work with looped paths. In these cases, Events or other VS Commands are typically used to stop the run. (For more information, please see *VS Commands Manual*.)

If either the fourth or fifth options are selected, then the field for stop station and the drop-down control for direction are hidden; in these two cases, nothing is written into the screen Parsfile with the keyword `OPT_DIRECTION` or `SSTOP`.

(9) The checkbox **Set vehicle X, Y, and yaw based on path?** allows the default location of the vehicle based on station to be disabled. If this box is not checked, the field for `SSTART` is hidden (8) unless the drop-down control (6) is set to the option: **Stop run at specified time or station**.

To locate the vehicle using X, Y, and Yaw, uncheck this box, and, in a miscellaneous field (e.g., (4)), type:

- `Xo` followed by the desired X coordinate in meters,
- `Yo` followed by the desired Y coordinate in meters, and
- `Yaw` followed by the desired yaw angle in degrees.

For more information about the initialization details in CarSim and TruckSim, please see the two Help documents: *Paths and Road Surfaces* (initialize location) and the Technical memo *Initialization in CarSim and TruckSim* (location, initialize speed, and equilibrium).

Plot Definitions

Plotting simulation data in CarSim, TruckSim, and BikeSim is specified via datasets linked using the **Plot: Setup** library.

When you click the **Plot** button on the **Run Control** screen, the same information sent to the VS Math Model is also scanned for plot setup information. Any links to datasets from the **Plot: Setup** library will be used to generate plots which are viewed using VS Visualizer.

(10) The first Plot link has an upper drop-down control to select a library. One of the options is the **Plot: Setup** library. Other options are generic libraries that may be used to group multiple datasets from the **Plot: Setup** library. Whatever library is chosen here is automatically linked for the following nine links (11).

(11) These links are made to the library selected at the top of the list (10).

(12) This link also has an upper drop-down control to select a library. Whatever library is chosen here is automatically linked for the following eight links (13).

(13) These links are made to the library selected at the top of the lower list (12).

Working with Multiple Vehicles in a Single Run

Starting with v2020.1, CarSim and TruckSim can run simulations with multiple vehicles created in a single solver. In these cases, controls normally associated with “the single ego vehicle” are extended to work with multiple vehicles. Controls such as the starting location, steering controller datasets, speed controller datasets, etc. apply for the “current vehicle” as defined with the index parameter `IVEHICLE`.

The `IVEHICLE` parameter is automatically set to 0 in every **Run Control** dataset. It is automatically incremented by all datasets for lead units. In CarSim, there is a single library for lead-unit datasets: **Vehicle Assembly**. In TruckSim, there are four, for lead units with 2, 3, 4, or 5 axles (e.g., **Vehicle Lead Unit with 3 Axles**). In a typical Run setup, the VS Math Model is initially given a line of input that sets `IVEHICLE` = 0. When the linked Parsfile for the lead unit is read, `IVEHICLE` is incremented with the line of text: `IVEHICLE = IVEHICLE + 1`. From that point on, all text read by the VS Math Model is done with `IVEHICLE` set to 1 or higher.

This behavior requires that at least one lead unit dataset (Parsfile) is read before any controls such as steering or speed. The layout of the **Run Control** screen ensures that this is true for all simulations with a single vehicle when the vehicle and procedure links are used as shown in existing examples.

In new simulation setups for multiple vehicles, there will often be multiple **Procedures** datasets specified, with one per vehicle. The links that specifically exist for driver controls (① - ③, Figure 1) and starting conditions (⑥ - ⑨) are applied for the most recently specified lead unit. Other links and fields (④, ⑤, ⑩ - ⑬) may be used for the simulation environment. Be aware that the additions such as paths, road, moving objects, events, etc. from all linked **Procedures** datasets will be added to the simulation. If multiple **Procedures** datasets link to the same optional feature (e.g., a set of 5 moving objects), then those features will accumulate as the linked datasets are read from all linked **Procedures** datasets.

A recommended practice is to create **Procedures** datasets with vehicle-specific controls only (e.g., ① - ③ and ⑥ - ⑨), linked immediately after the link for the vehicle that will use those controls.

VS Events

The **Procedures** screen is used to determine controls and conditions at the start of a simulation. In some scenarios of interest, you will want to change controls or other parts of the model while the simulation is running. VS Events are used to make these changes.

What Are VS Events?

A VS Math Model can monitor a set of conditions that you specify with formulas, possibly coupled with a pathname for a Parsfile. A formula-pathname pair is called a *Pending Event*. At every time step of the simulation, each formula on the Pending Event list is evaluated in succession. If a result is nonzero, a VS Event is triggered. The possible actions associated with an Event are simple:

1. the run stops if there is no coupled pathname, or
2. the coupled pathname is used to read a Parsfile (and possibly other linked Parsfiles), and the simulation continues.

The options that are enabled by reading a Parsfile during an Event are almost unlimited: the new data files can change values of parameters, math model keywords, and control settings, some of which may even be properties describing the vehicle and road. Event formulas can include any of the functions and operators supported in VS Commands, and all variables and parameters that have keywords recognized by the VS Math Model.

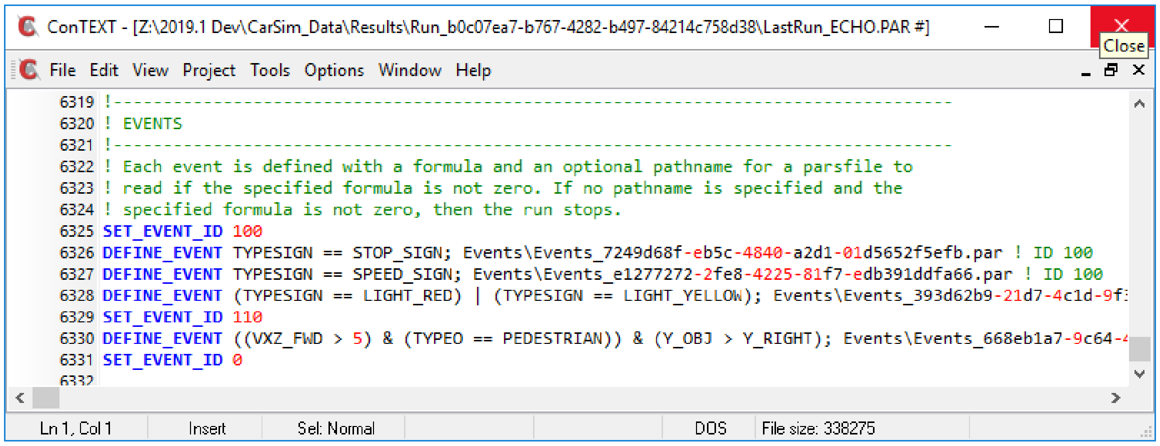
The only restrictions are that you cannot change simulation settings that define what data is written to file and/or exchanged with other software. For example, you cannot change the time step, the array of active import and/or export variables, etc. You also cannot change modules in the VS Math Model (e.g., change the number of vehicles, lose a trailer, change a suspension type, etc.).

Examples of Events usage include:

- Instructions for test sequences, such as accelerating to a speed, holding the speed for a while, braking or steering until something happens, resuming control and speed, and repeating.

- Mimicking driver behavior in ADAS situations, such as looking for signs and pedestrians, and reacting accordingly.
- Changing roads and paths in traffic scenarios.
- Resetting the vehicle's position (e.g., during a stopping distance test).
- Monitoring the number of times the vehicle completes a looped path (i.e., the number of laps).
- Monitoring a set of conditions and repeating some part of the run when those conditions necessitate jumping back in time using with the VS Command `RESTORE_STATE`.

New pending Events are created with the VS Commands `DEFINE_EVENT` and `MAKE_EVENT`, which are described along with other VS Commands in *VS Commands Manual*. Pending Events that have been defined at the start of the run are listed in the Echo file, near the end, after all other VS Commands (Figure 6).


ConTEXT - [Z:\2019.1 Dev\CarSim_Data\Results\Run_b0c07ea7-b767-4282-b497-84214c758d38\LastRun_ECHO.PAR #]
Close
File Edit View Project Tools Options Window Help
Ln 1, Col 1
Insert
Sel: Normal
DOS
File size: 338275

```
6319 !-------------------------------------------------------------------------------
6320 ! EVENTS
6321 !-------------------------------------------------------------------------------
6322 ! Each event is defined with a formula and an optional pathname for a parsfile to
6323 ! read if the specified formula is not zero. If no pathname is specified and the
6324 ! specified formula is not zero, then the run stops.
6325 SET_EVENT_ID 100
6326 DEFINE_EVENT TYPESIGN == STOP_SIGN; Events\Events_7249d68f-eb5c-4840-a2d1-01d5652f5efb.par ! ID 100
6327 DEFINE_EVENT TYPESIGN == SPEED_SIGN; Events\Events_e1277272-2fe8-4225-81f7-edb391ddfa66.par ! ID 100
6328 DEFINE_EVENT (TYPESIGN == LIGHT_RED) | (TYPESIGN == LIGHT_YELLOW); Events\Events_393d62b9-21d7-4c1d-9f
6329 SET_EVENT_ID 110
6330 DEFINE_EVENT ((VXZ_FWD > 5) & (TYPEO == PEDESTRIAN)) & (Y_OBJ > Y_RIGHT); Events\Events_668eb1a7-9c64-
6331 SET_EVENT_ID 0
6332
```

Figure 6. Listing of pending Events near the end of an Echo file.

The Echo file identifies each pending Event with the VS Command `DEFINE_EVENT`, followed by a formula that will either be zero or not zero when it is evaluated, and possibly, the pathname of a Parsfile.

At every simulation time step, each pending Event is checked in succession by evaluating its formula to see if it is nonzero. For example, if the variable `TYPESIGN` is equal to `STOP_SIGN`, then the Event defined in line 6326 (Figure 6) would be triggered. In that case, the VS Math Model would immediately read the specified Parsfile `Events\Events_7249….par`, erase the Event, and continue the simulation.

In addition to the condition and Parsfile specified in the `DEFINE_EVENT` command, each Event has an associated ID number that may be used to delete a group of Events while leaving others intact. The ID for each Event is shown as a comment at the end of the line of text. In this example, the first three Events have the ID 100 and the last one has the ID 110.

The command `SET_EVENT_ID` is used to specify the ID that will be used for new Events created with the `DEFINE_EVENT` command. In the example, `SET_EVENT_ID` sets the ID to 100 (line

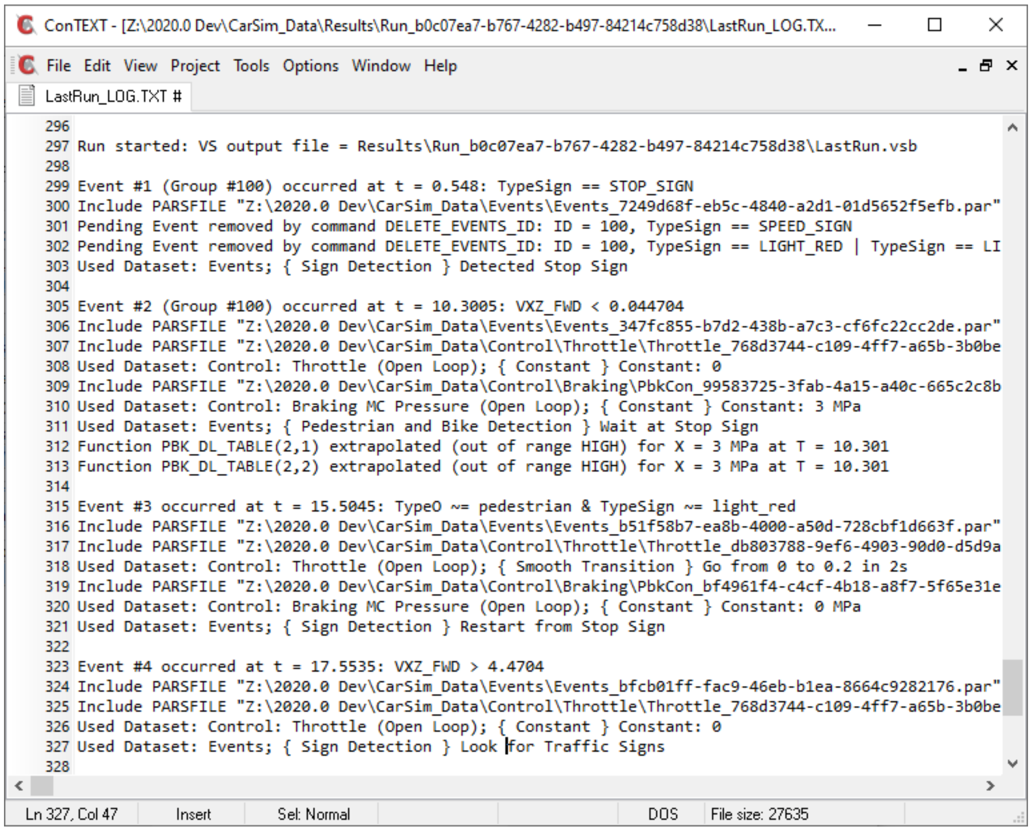
6325), which is used for the following Events, until the next `SET_EVENT_ID` is applied in line 6329. The default ID is 0, so after all Events are defined, the ID is set back to 0 (line 6331).

As noted above, if an Event contains no coupled pathname — i.e., an **Events** dataset does not contain links to more **Events** datasets — then the simulation will terminate if that Event is triggered.

The Log File

Whenever a simulation is run, a log file is generated that shows which Parsfiles were read, error messages that were generated, Events that were triggered, and when the run was stopped. The log file for a simulation can be viewed any time after a simulation has been made using the **View** button in the lower-right corner of the **Run Control** screen.

Figure 7 shows part of the log file for the same simulation used to generate the Echo file shown earlier (Figure 6). The log file lists each Parsfile by title after it is read, with a line that begins with the text "Used Dataset:".


ConTEXT - [Z:\2020.0 Dev\CarSim_Data\Results\Run_b0c07ea7-b767-4282-b497-84214c758d38\LastRun_LOG.TX...
File Edit View Project Tools Options Window Help
LastRun_LOG.TXT #

```
296
297 Run started: VS output file = Results\Run_b0c07ea7-b767-4282-b497-84214c758d38\LastRun.vsb
298
299 Event #1 (Group #100) occurred at t = 0.548: TypeSign == STOP_SIGN
300 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Events\Events_7249d68f-eb5c-4840-a2d1-01d5652f5efb.par"
301 Pending Event removed by command DELETE_EVENTS_ID: ID = 100, TypeSign == SPEED_SIGN
302 Pending Event removed by command DELETE_EVENTS_ID: ID = 100, TypeSign == LIGHT_RED | TypeSign == LI
303 Used Dataset: Events; { Sign Detection } Detected Stop Sign
304
305 Event #2 (Group #100) occurred at t = 10.3005: VXZ_FWD < 0.044704
306 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Events\Events_347fc855-b7d2-438b-a7c3-cf6fc22cc2de.par"
307 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Control\Throttle\Throttle_768d3744-c109-4ff7-a65b-3b0be
308 Used Dataset: Control: Throttle (Open Loop); { Constant } Constant: 0
309 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Control\Braking\PbkCon_99583725-3fab-4a15-a40c-665c2c8b
310 Used Dataset: Control: Braking MC Pressure (Open Loop); { Constant } Constant: 3 MPa
311 Used Dataset: Events; { Pedestrian and Bike Detection } Wait at Stop Sign
312 Function PBK_DL_TABLE(2,1) extrapolated (out of range HIGH) for X = 3 MPa at T = 10.301
313 Function PBK_DL_TABLE(2,2) extrapolated (out of range HIGH) for X = 3 MPa at T = 10.301
314
315 Event #3 occurred at t = 15.5045: TypeO ~= pedestrian & TypeSign ~= light_red
316 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Events\Events_b51f58b7-ea8b-4000-a50d-728cbf1d663f.par"
317 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Control\Throttle\Throttle_db803788-9ef6-4903-90d0-d5d9a
318 Used Dataset: Control: Throttle (Open Loop); { Smooth Transition } Go from 0 to 0.2 in 2s
319 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Control\Braking\PbkCon_bf4961f4-c4cf-4b18-a8f7-5f65e31e
320 Used Dataset: Control: Braking MC Pressure (Open Loop); { Constant } Constant: 0 MPa
321 Used Dataset: Events; { Sign Detection } Restart from Stop Sign
322
323 Event #4 occurred at t = 17.5535: VXZ_FWD > 4.4704
324 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Events\Events_bfcb01ff-fac9-46eb-b1ea-8664c9282176.par"
325 Include PARSFILE "Z:\2020.0 Dev\CarSim_Data\Control\Throttle\Throttle_768d3744-c109-4ff7-a65b-3b0be
326 Used Dataset: Control: Throttle (Open Loop); { Constant } Constant: 0
327 Used Dataset: Events; { Sign Detection } Look for Traffic Signs
328
```

Figure 7. Log file showing activities involving Events.

After the run starts (line 297), the log file reports when Events are triggered (e.g., line 299), and continues to report each dataset that is read. For example, line 300 reports the pathname for a Parsfile that was read, and line 303 reports the library (`Events`), category (`Sign Detection`), and title (`Detected Stop Sign`) for the dataset. The log file also reports when pending events were removed (e.g., lines 301 and 302).

Resetting Clock Times

A VS Math Models maintain several measures of time. The simulation time starts at a value specified with a parameter (`TSTART`) and increases as the run proceeds. The simulation variable has two names (`T` and `T_Stamp`). It can be used in plots to provide an absolute time stamp which serves as an alternative to the recording time that post-processing programs calculate by multiplying the sample interval by the count of samples read from the file.

The recording time is identified in plots with the name `Time` and the time stamp is identified with the name `T_Stamp`. If time is used in Events or equations added with VS Commands, the symbol `T` or `T_Stamp` should be used (both keywords are attached to the same internal variable).

Note In older versions of the software, `T` and `T_Stamp` were names of different internal variables. However, in recent versions, they are alternate names of the single built-in variable for simulation time.

Most of the open-loop driver controls are defined by Configurable Functions with the form:

$$native\_control = f([T - tstart]/tscale)$$

where

native_control is the built-in vehicle control available for use in the simulation,

f is a Configurable Function that uses a method you define on the screen (e.g., table lookup with spline interpolation),

`T` is the current simulation time,

tstart is a parameter that can be set in one of the Miscellaneous fields on the Event screen, causing the open-loop control to be applied as if the time were currently zero, and

tscale is a parameter to scale a waveform. The default value is one, but can be set to another value by the user in a Misc. yellow field.

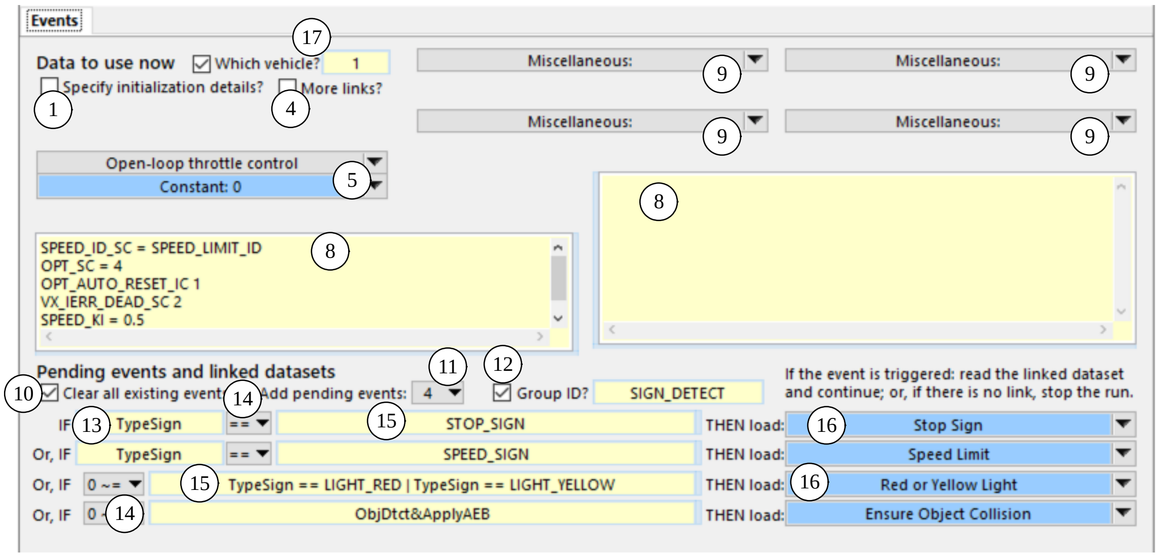
Table 1 lists the time-offset keywords used to specify *tstart.*

Table 1. Summary of time offset parameters.

Time offset	**Tables that use the offset**
TSTART_T_EVENT	No tables; used for `T_Event`
TSTART_BRAKES	Open-loop brake control
TSTART_CLUTCH	Open-loop clutch control
TSTART_F_BRAKE_PEDAL	Open-loop brake pedal force
TSTART_GEAR	Open-loop shift control
TSTART_PBK_CON	Open-loop brake master cylinder pressure
TSTART_SPEED_CTRL	Closed-loop target speed
TSTART_STEER	Open-loop steer control (angle)
TSTART_STEER_TQ	Open-loop steer control (torque)
TSTART_THROTTLE	Open-loop throttle control
TSTART_TRANS	Open-loop transmission mode control
TSTART_WIND	Open-loop wind speed and heading angle

The Events Screen

Figure 8 shows the Events screen that was used to define the first three Events that were listed in the Echo file shown earlier (Figure 6, page 9). The **Events** screen is used to change simulation conditions while a run is in progress.



Events

Data to use now ☑ Which vehicle? 1

☐ Specify initialization details? ☐ More links?

Miscellaneous: | Miscellaneous: | Miscellaneous: | Miscellaneous:

Open-loop throttle control

Constant: 0

```
SPEED_ID_SC = SPEED_LIMIT_ID
OPT_SC = 4
OPT_AUTO_RESET_IC 1
VX_IERR_DEAD_SC 2
SPEED_KI = 0.5
```

Pending events and linked datasets

☑ Clear all existing events Add pending events: 4 ☑ Group ID? SIGN_DETECT

If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF	TypeSign	==	STOP_SIGN	THEN load:	Stop Sign
Or, IF	TypeSign	==	SPEED_SIGN	THEN load:	Speed Limit
Or, IF	0 ~=		TypeSign == LIGHT_RED | TypeSign == LIGHT_YELLOW	THEN load:	Red or Yellow Light
Or, IF	0		ObjDtct&ApplyAEB	THEN load:	Ensure Object Collision

Figure 8. The Events screen, showing multiple new Events.

Four types of controls are available:

1. A subset of the controls that are available on the **Procedures** screen for assembling data involving driver controls (① - ⑦, not all are shown in Figure 8).
2. Miscellaneous fields ⑧ and links ⑨ that can be used as needed to modify the simulation conditions.
3. Define a set of pending Events, using the controls in the bottom part of the screen (⑩ - ⑯).
4. The vehicle ID ⑰, for simulations with multiple vehicles.

Initialization settings

The VS math models have options to set clock times, vehicle position, suspension deflections, and wheel speeds at the start of a run. These options may also be set when an **Events** dataset is read.

Note that an **Events** dataset Parsfile will be read by the VS Math Model before the runs starts if it is linked to the **Run Control** dataset, the **Procedures** dataset for the run, or any other dataset that used to setup the run. These are sometimes called *top-level Event datasets*. However, **Events** datasets are typically linked to other **Events** datasets ⑯ that are read only when a pending Event is triggered, after the run has started. In this subsection, these are called *triggered datasets*.

① **Specify initialization details?** Check this to view four more checkboxes (Figure 9). When unchecked (Figure 8), nothing is written to the Parsfile involving the four initialization options.

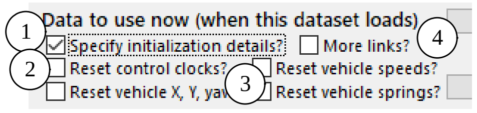


Figure 9. Initialization details that may optionally be displayed.

(2) **Reset all control clocks?** If this box is checked, then all of the time offsets listed in Table 1 (page 11) are set to the current time `T`. A built-in variable called `T_EVENT`, available for use in some advanced scenarios, is calculated every time step as:

```
T_EVENT = T - TSTART_T_EVENT
```

where `TSTART_T_EVENT` is one of the parameters listed in Table 1. This checkbox offers a convenient way to reset the clocks. However, if you want to reset some clocks but not all, then you should enter VS Commands in one of the Miscellaneous fields (8) to specify the time offsets you want to modify.

For example, if you want to apply an open-loop braking control when an Event occurs, you can enter the following line into one of the Miscellaneous fields (8), and use one of the Miscellaneous links (9) to access an open-loop brake control dataset.

```
TSTART_BRAKES = T
```

(3) Vehicle initialization option checkboxes. A VS Math Model has three options for applying automatic initializations (Table 2).

Table 2. Vehicle initialization options.

Checkbox	Parameter	When checked…
Reset X, Y, & yaw?	`OPT_INIT_PATH`	The vehicle's X, Y, and Yaw state variables are set using `Station`, applied for the path with ID `PATH_ID_DM`; the lateral coordinate is calculated with the `LTARG` dataset `LTARG_ID_DM`; and the yaw angle is set based on the heading of the path and the value of `OPT_DIRECTION`.
Reset vehicle speeds?	`OPT_INIT_SPEED`	The forward vehicle speed is set to match the target speed from the speed controller, wheel speeds are set to match the vehicle speed, and spins in the powertrain are also set. If the speed controller is not active, the target speed for the initialization is set to the state variable `SV_VXS` (X component of lead-unit sprung mass velocity).
Reset vehicle springs?	`OPT_INIT_CONFIG`	Initialize vehicle pitch, roll, Zo, and all ride spring compressions, such that the vehicle is in approximate equilibrium on the 3D ground surface.

If these boxes are not visible (i.e., the box (1) is not checked), then the three option parameters all keep their values. If the Event dataset is read before the run has started, the three option parameters are typically set to 1; if the dataset is read after the run has stated, the three parameters are typically 0.

If these boxes are visible, then the individual initialization options are specifically turned on or off, as described in Table 2. If the vehicle initialization checkboxes are visible but unchecked — i.e., the vehicle initialization parameters are set to zero — the VS Math Model behavior that occurs depends on whether the Events dataset is top-level or triggered.

1. If this is a top-level Event and the initialization option boxes are visible but unchecked, this disables the initializations. This will usually cause a problem for the initializations specified with `OPT_INIT_SPEED` and `OPT_INIT_CONFIG`.

2. If this is a triggered Event (the run is in progress) and the boxes on the **Events** screen are unchecked, the vehicle state variables are left alone.

3. If this is a triggered Event and the boxes on the Event dataset are checked, the vehicle initialization will be performed even though the run is in progress. This is sometimes done to simulate a sequence of tests in a single simulation run, with the vehicle being reset for each test.

For more information about the initialization details in CarSim and TruckSim, please see the two documents: *Paths and Road Surfaces* and the technical memo *Initialization in CarSim and TruckSim*. There is also a technical memo that shows how to initialize the position using variables imported from external software such as Simulink: *Initialize a Vehicle Using Imported Variables*.

4 **More links?** Check this box to show two additional Miscellaneous links 9. To make room, the screen is refreshed with the Miscellaneous yellow fields 8 resized.

Speed Control

5 Speed / Acceleration option. This drop-down list has options for controlling the vehicle speed using open-loop throttle or the closed-loop speed controller.

If the choice is made for **Constant target speed**, two more controls are shown (Figure 10). As with the **Procedures** screen, a yellow field is shown for the target speed and a checkbox is shown to enable the use of engine braking by the speed controller 7.

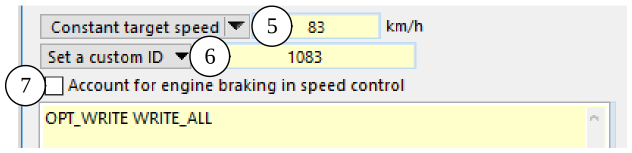


Figure 10. Setting a constant target speed.

Unlike the **Procedures** screen, the **Events** screen also shows a pair of controls for setting the ID of the `SPEED_TARGET` Configurable Function 6.

6 Drop-down control with two options may affect the number of `SPEED_TARGET` datasets used in the simulation.

If the first option is chosen (**Add a dataset for this speed to the model**), then the Parsfile for this **Events** dataset will include commands to add a new `SPEED_TARGET` dataset to the simulation and configure it with the specified value of the speed ⑤. The ID for the new dataset is set automatically.

If the second option is chosen (**Set a custom ID**), then a yellow field is shown to specify an ID for the `SPEED_TARGET` dataset, as shown in Figure 11. This must be an integer greater than or equal to 999, or a formula that will be ≥ 999 when evaluated.

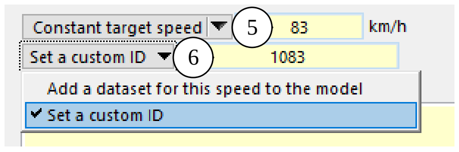

Constant target speed ⑤ 83 km/h
Set a custom ID ⑥ 1083

Figure 11. Drop-down control for choosing options for the ID of the speed control dataset.

If there is a `SPEED_TARGET` dataset with the specified ID, then it is reconfigured as a constant with the specified value. If no dataset exists with the specified ID, then the following steps are taken:

1. a dataset is created,
2. the ID is set to the specified integer value ⑥, and
3. the new dataset is configured as a constant with the specified value ⑤.

This handling of a constant target speed is more complicated than is done for the **Procedures** screen, which has no option for dealing with the ID of a `SPEED_TARGET` dataset. The reason is that Event datasets are sometimes read repeatedly in a simulation that involves iteration. If no custom ID is specified, then every time the Parsfile for the Event dataset is read by the VS Math Model, it will add a new `SPEED_TARGET` dataset. Using the custom ID allows the VS Math Model to reuse an existing dataset through unlimited iterations.

Note A VS Math Model supports up to 200 `SPEED_TARGET` datasets for a given run. Most users will never need to use all 200, but if an Event is used to load a `SPEED_TARGET` dataset, the use of the automatically assigned ID number means each time the **Events** dataset is loaded, a new `SPEED_TARGET` dataset is created, and the internal ID is incremented by one. With the built-in limit of 200, eventually the simulation could stop due to reaching this limit. If an Event is used to load a `SPEED_TARGET` dataset, and that Event is going to be loaded repeatedly throughout the run, it is a good idea to set a custom ID.

⑦ When the speed control option is set to Constant target speed (Figure 10), this checkbox is used to specify whether the speed controller will account for engine braking. Other settings for the speed controller will keep their existing settings. If needed, their values can be changed by setting them in one of the Misc. yellow fields.

Miscellaneous Fields and Links

⑧ Miscellaneous yellow fields. Enter math model keywords and the values or formulas you want assigned to them. The format is that each line has a keyword and value that are separated with white space and/or an ‘=’ sign (Figure 8).

⑨ Miscellaneous links to other datasets. Use these links to assemble information about driver controls (open-loop and closed-loop) and other conditions that might be associated with the test.

Limits Some datasets are intended only for specifying conditions prior to the start of a run, and should not be included in an **Events** dataset that is not triggered until the run is in progress. Data that cannot be changed after the run starts are simulation parameters (time steps, start time, etc.), VS Commands to create new import variables, etc.

Be aware that VS Visualizer only receives setup data that is available before the run starts. Datasets for animator shapes, reference frames, plots, etc. that are accessed by an **Events** dataset after the simulation starts will not be used. For example, if a tire dataset is linked on an **Events** dataset, then when the Event is triggered, the linked tire data (e.g., Fx vs. Slip Ratio) will be used by the math model and replace what was linked directly to the vehicle using the **Vehicle: Assembly** screen. However, if the linked tire dataset has animation shapes, those animation shapes will not be shown in VS Visualizer. The VS Visualizer will continue to show the tire animation shapes associated with the tires that were linked directly to the vehicle.

Defining Pending Events

⑩ Checkbox to clear the list of existing Events. When this dataset is encountered, there might already be a list of pending Events. If this box is checked, the existing list is cleared. This option is handy if independent tests are run in sequence; it clears all Event information from the previous test when starting a new one.

Note that an Event is automatically cleared when it is triggered. This checkbox is used to clear other Events that might be pending.

If the Group ID checkbox is shown ⑫ and a value is provided in the adjacent field, then only Events with the specified ID are deleted if the Clear box ⑩ is checked. If the checkbox ⑫ is not checked, then all pending Events are cleared, regardless of their associated ID numbers.

As shown earlier, the log file reports all occasion when pending Events are removed (Figure 7, page 10).

⑪ Number of Events defined on the current screen. Up to 10 Events may be defined in one dataset. To specify more than 10 pending Events, you can link to more **Events** datasets using one of the Miscellaneous links ⑨.

When more than one Event is pending, it is shown as being connected to the other pending Events with a logical OR. The first condition on the list is evaluated, and if it is not triggered the next one is tested, and so on.

⑫ Checkbox to specify a group ID for all Events referenced in the current dataset. When checked, an adjacent yellow field is shown in order to specify an integer ID.

In the example, the ID is set with a user-defined parameter `SIGN_DETECT`. The ID is specified in the Parsfile with the command `SET_EVENT_ID`, which is also listed in the Echo file if the Event is pending when the Echo file is written (Figure 6, page 9).

If the Clear box ⑩ is checked and a value is provided in the ID yellow field, then a VS Command `DELETE_EVENTS_ID` is written in the Parsfile to clear all Events with this ID. For the example `ID SIGN_DETECT`, the Parsfiles contains a line of text:

```
DELETE_EVENTS_ID SIGN_DETECT
```

⑬ Variable used to make an Event, along with an operator ⑭, *reference* ⑮, and possibly a link to another Parsfile that will be read if the Event is triggered ⑯. This field is hidden if the operator drop-down control ⑭ (Figure 12) is set to either of the first two options.

⑭ The definition of an Event is shown on the screen with the form:

`IF` *formula* [`THEN load` *dataset*]

This drop-down control determines how the Event formula is set up. The options available from this control are listed in Table 3. With the first two options (**0 ~=** and **0 ==**), a single yellow field ⑮ is shown for a formula that will trigger an Event when the formula is not zero (**0 ~=**, as shown for the third event in the figure), or when the formula is zero (**0 ==**). The other field ⑬ is hidden in these cases.

Table 3. Relational operators that can define an Event condition.

Operator	Logical Condition Description	VS Command
0 ~=	True if *formula* is not zero.	`DEFINE_EVENT`
0 ==	True if *formula* is zero.	
==	True if *variable* is equal to *reference*.	`MAKE_EVENT`
~=	True if *variable* is not equal to *reference*.	
>	True if *variable* is greater than *reference*.	
>=	True if *variable* is greater than or equal to *reference*.	
<	True if *variable* is less than *reference*.	
<=	True if *variable* is less than or equal to *reference*.	

The Event is defined by establishing a formula that is evaluated each time step. There are two layouts available for entering the formula, with both shown in Figure 8 and Figure 12.

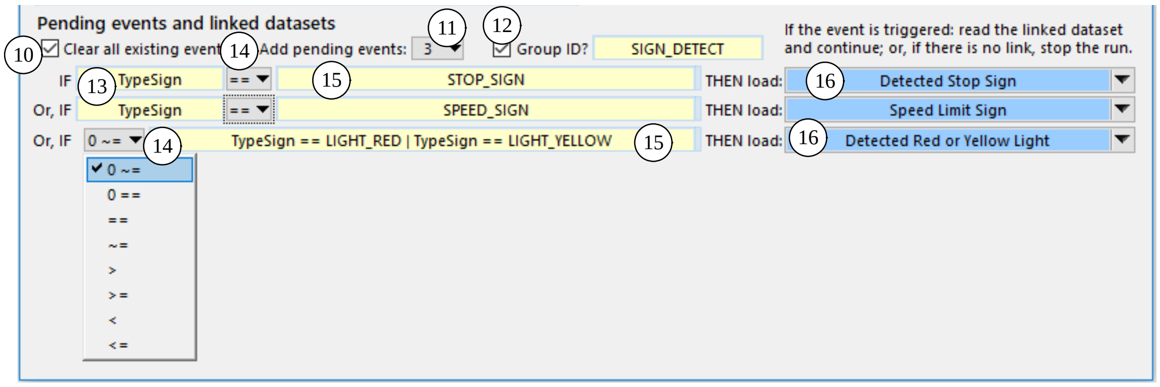
In the first case (**0 ~=**), the VS Command `DEFINE_EVENTS` is used to create the pending Event using whatever formula is provided in the adjacent field ⑮. The statement (seen in the Echo file) has the form:

`DEFINE_EVENT` *formula* [; [*pathname*]]

In the second case (**0 ==**), the same command is used, but the formula from the yellow field is put in parentheses and preceded with the character '~' (the "not" operator):

`DEFINE_EVENT ~(` *formula* `)` [; [*pathname*]]

With the "not" operator, the expression evaluates as "true" when *formula* is equal to zero.



Pending events and linked datasets

☑ Clear all existing events   Add pending events: 3   ☑ Group ID? SIGN_DETECT

If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF	TypeSign	==	STOP_SIGN	THEN load:	Detected Stop Sign
Or, IF	TypeSign	==	SPEED_SIGN	THEN load:	Speed Limit Sign
Or, IF	0 ~=		TypeSign == LIGHT_RED | TypeSign == LIGHT_YELLOW	THEN load:	Detected Red or Yellow Light

✓ 0 ~=
0 ==
==
~=
>
>=
<
<=

Figure 12.Options for specifying the formula for an Event.

With the other six options, two fields are shown, separated by the Boolean relational operator. For example, the first Event in Figure 12 uses the == operator to create a formula with a named *variable* (`TypeSign` ⑬) and a *reference* formula (`STOP_SIGN` ⑮). In this case, the VS Command `MAKE_EVENT` is used to create the pending Event. This command has the form:

`MAKE_EVENT` *variable operator reference* [; [*pathname*]]

where *variable* is the name of any variable or parameter that appears in an Echo file produced by the VS Math Model, including output variables, floating point parameters (i.e., those with units), state variables, and import variables, and *reference* is a number or formula involving any variables, parameters, and functions that are allowed in VS Command formulas.

Alert Be careful about using the == and ~= operators with formulas. When formulas are applied numerically, truncation/round-off errors can occur that will cause two expressions that should mathematically be identical to differ. For example, (*x*/3)*3 might not be equal to *x* because the result of the divide (x/3) will have truncation error, and multiplying the result by 3 will usually not undo that error.

In addition, there may also be errors in the conversion made by the computer in going from decimal to binary. For example, the decimal number 0.025 is represented internally as `0.025000000000000001`.

Consider using the operators > or < instead of ==, and possibly adding a tolerance. For example, if you want an Event to trigger when T = 1.0s, consider testing for T >= 0.99999.

The exception to this concern is testing for zero. There is no truncation or round-off: zero is zero.

More information about differences between `DEFINE_EVENT` and `MAKE_EVENT` is provided in the next subsection.

(15) This field is either:

1. a self-contained formula, if the drop-down control is set to either of the first two options listed in Table 3, or
2. just the right-hand side of a logical expression, if any of the other six options are specified with the drop-down control.

(16) Link to another Event dataset to read if the Event is triggered. If no dataset is specified here and the current Event is triggered, then the run terminates. As shown earlier, all triggering of Events is recorded in the log file (see Figure 7, page 10).

Differences between DEFINE_EVENT and MAKE_EVENT

The Event was introduced in 2004, before the VehicleSim architecture was developed for the solver modules. In the original version, the command `DEFINE_EVENT` command was required to have the form:

`DEFINE_EVENT` *variable operator reference* [; [*pathname*]]   {*Old Form: up to 2019.0*}

where *operator* was either '<' or '>' and *reference* was a number. In later versions, *operator* was extended to include the six options listed in Table 3, and *reference* was extended to include symbolic formulas.

Starting with version 2019.1, VS Math Models can handle Boolean operators in formulas. With this capability, the syntax was changed to the current one, used for showing Events in an Echo file:

`DEFINE_EVENT` *formula* [; [*pathname*]]

The new form is far more versatile and allows a single pending Event to include AND and OR conditions more easily than before.

However, the original syntax supported a feature that was convenient in the case where *reference* is a number; it automatically converted from user units to internal units. For example, Figure 13 shows an Event dataset used to start a run and accelerate until a target speed of 80km/h is reached, at which time the Event is triggered and a linked dataset is read.

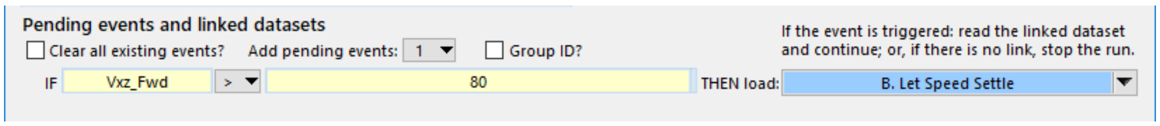
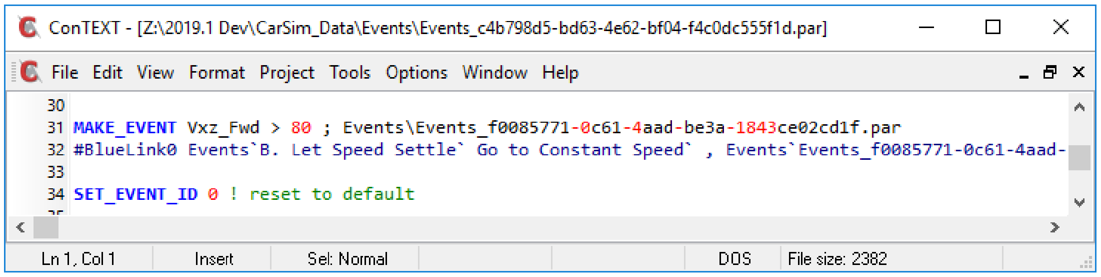

Pending events and linked datasets
☐ Clear all existing events? Add pending events: 1 ☐ Group ID?
If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.
IF Vxz_Fwd > 80 THEN load: B. Let Speed Settle

Figure 13. Example where reference is a number.

Figure 14 shows a portion of the Parsfile containing the data for the screen display shown in Figure 13, which uses the `MAKE_EVENT` command followed by a formula that closely resembles the screen display. On the other hand, Figure 15 shows the resulting `DEFINE_EVENT` command in which the numerical speed value was converted from 80km/h to 22.2222m/s.

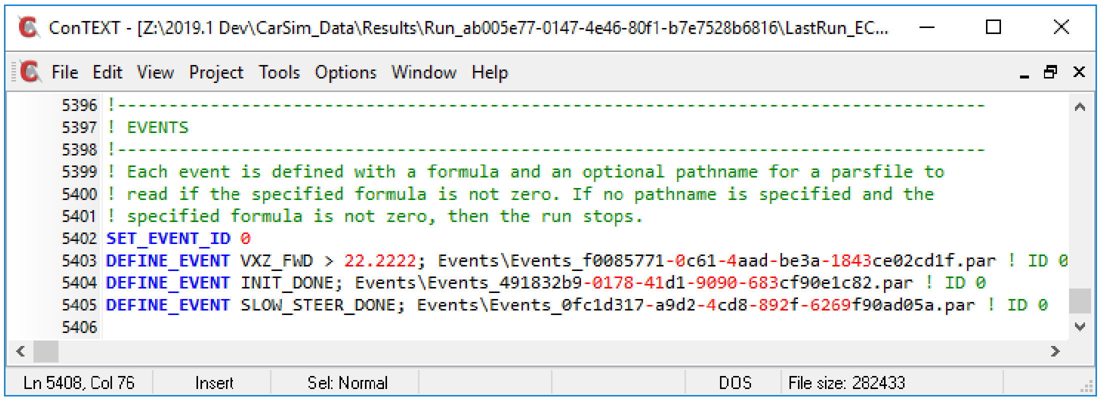

ConTEXT - [Z:\2019.1 Dev\CarSim_Data\Events\Events_c4b798d5-bd63-4e62-bf04-f4c0dc555f1d.par]
File Edit View Format Project Tools Options Window Help
Ln 1, Col 1
Insert
Sel: Normal
DOS
File size: 2382

```

MAKE_EVENT Vxz_Fwd > 80 ; Events\Events_f0085771-0c61-4aad-be3a-1843ce02cd1f.par
#BlueLink0 Events`B. Let Speed Settle` Go to Constant Speed` , Events`Events_f0085771-0c61-4aad-

SET_EVENT_ID 0 ! reset to default
```

Figure 14. The use of the MAKE_EVENT command in a Parsfile.


ConTEXT - [Z:\2019.1 Dev\CarSim_Data\Results\Run_ab005e77-0147-4e46-80f1-b7e7528b6816\LastRun_EC...
File Edit View Project Tools Options Window Help
Ln 5408, Col 76
Insert
Sel: Normal
DOS
File size: 282433

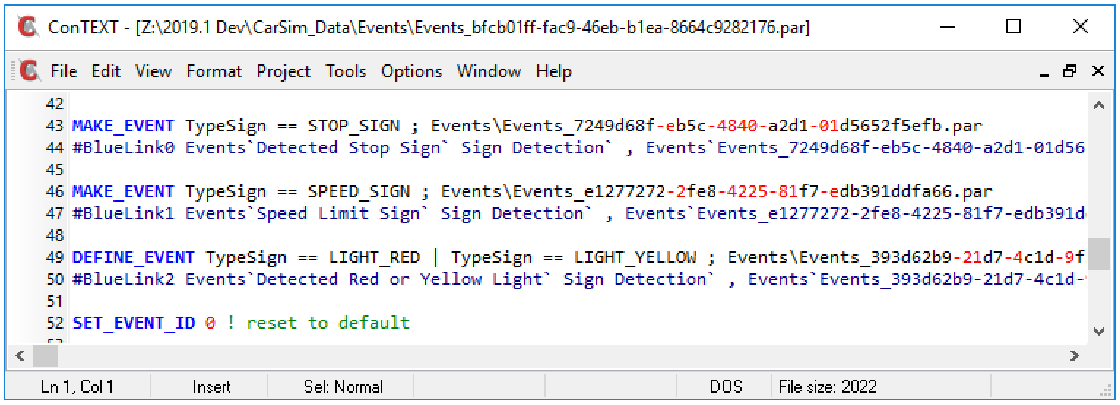
```
!-------------------------------------------------------------------------------
! EVENTS
!-------------------------------------------------------------------------------
! Each event is defined with a formula and an optional pathname for a parsfile to
! read if the specified formula is not zero. If no pathname is specified and the
! specified formula is not zero, then the run stops.
SET_EVENT_ID 0
DEFINE_EVENT VXZ_FWD > 22.2222; Events\Events_f0085771-0c61-4aad-be3a-1843ce02cd1f.par ! ID 0
DEFINE_EVENT INIT_DONE; Events\Events_491832b9-0178-41d1-9090-683cf90e1c82.par ! ID 0
DEFINE_EVENT SLOW_STEER_DONE; Events\Events_0fc1d317-a9d2-4cd8-892f-6269f90ad05a.par ! ID 0

```

Figure 15. The DEFINE_EVENT command created by a MAKE_EVENT command.

The MAKE_EVENT command was introduced in 2019.1 solely to provide backward compatibility with older datasets. It has exactly the same behavior as the original (pre 2019.1) DEFINE_EVENT command. The Browser always writes the MAKE_EVENT command when two fields are used (*variable* and *reference*).

For example, Figure 16 shows part of the Parsfile for the data shown in Figure 8 and Figure 12. The first two Events are created with the MAKE_EVENT command; the third uses the DEFINE_EVENT command.


ConTEXT - [Z:\2019.1 Dev\CarSim_Data\Events\Events_bfcb01ff-fac9-46eb-b1ea-8664c9282176.par]
File Edit View Format Project Tools Options Window Help
Ln 1, Col 1
Insert
Sel: Normal
DOS
File size: 2022

```

MAKE_EVENT TypeSign == STOP_SIGN ; Events\Events_7249d68f-eb5c-4840-a2d1-01d5652f5efb.par
#BlueLink0 Events`Detected Stop Sign` Sign Detection` , Events`Events_7249d68f-eb5c-4840-a2d1-01d56

MAKE_EVENT TypeSign == SPEED_SIGN ; Events\Events_e1277272-2fe8-4225-81f7-edb391ddfa66.par
#BlueLink1 Events`Speed Limit Sign` Sign Detection` , Events`Events_e1277272-2fe8-4225-81f7-edb391d

DEFINE_EVENT TypeSign == LIGHT_RED | TypeSign == LIGHT_YELLOW ; Events\Events_393d62b9-21d7-4c1d-9f
#BlueLink2 Events`Detected Red or Yellow Light` Sign Detection` , Events`Events_393d62b9-21d7-4c1d-

SET_EVENT_ID 0 ! reset to default
```

Figure 16. Parsfile with both MAKE_EVENT and DEFINE_EVENT commands.

Echo files will never show MAKE_EVENT commands; the command exists only to handle unit conversions to make a correct DEFINE_EVENT statement.

Identifying the Vehicle in CarSim or TruckSim

As noted earlier, CarSim and TruckSim can run simulations with multiple vehicles created in a single solver. Controls such as the starting location, steering controller datasets, speed controller datasets, etc. apply for the "current vehicle" as defined with the index parameter `IVEHICLE`.

Linked datasets can be put in the correct order to ensure that controls for each vehicle are associated with the vehicle at the start of the run, as described earlier for Procedure datasets (page 7). However, when a Pending Event is triggered, the value of `IVEHICLE` will be set to last one added, unless changes to `IVEHICLE` were made. If a Pending Event is intended to be used with a specific vehicle, the checkbox and control ⑰ may be used to set `IVEHICLE`.

⑰ Checkbox available in CarSim and TruckSim to show an adjacent yellow field that can be used to set `IVEHICLE`. This setting is only relevant if a simulation has two or more vehicles. If there is only a single vehicle, the specified `IVEHICLE` is ignored.

If there are two more vehicles, the `IVEHICLE` setting determines which vehicle is affected by the initialization controls (① - ③), speed control (⑤ - ⑦), and other controls set with keywords ⑧ and links ⑨.

Examples That Involve Events

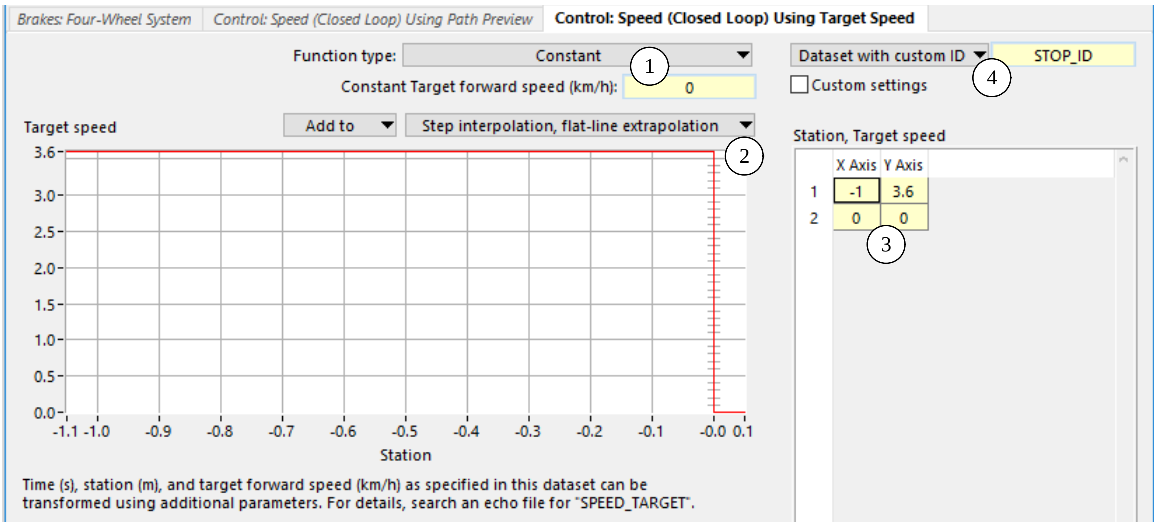
CarSim, BikeSim, and TruckSim include many example **Events** datasets with Notes describing the purpose of the dataset. In many cases, there will be a sequence of Events to simulate a complicated test or scenario. A convention in these cases is that all of the Event datasets are in the same user-defined category. If they are applied in a simple sequence, they will often have prefixes (A, B, …) such that when they are sorted into alphabetical order, the sequence is the same as how they are triggered.

Stopping at a Stop Sign

Many of the example ADAS scenarios that are included with CarSim, TruckSim, and BikeSim make use of Events that can be triggered in various sequences. The first pending Event in the **Events** dataset shown in Figure 8 and Figure 12 is triggered when a stop sign is detected.

This example also makes use of the **Procedures** datasets shown earlier (Figure 1, page 2). Note that the speed control was set with a link to the library for using the speed controller with path preview ① (Figure 1). In this mode, the controller attempts to follow the target speed, subject to limits in acceleration in four directions: throttle, braking, turning left, turning right. In this mode, the speed controller can be used to brake the vehicle such that it will come to a stop at a designated position. To do this, set the target speed to the current vehicle speed, and change the target to 0 at the intended stopping point. The speed controller will use the target speed data and generate a new (internal) target speed profile that will reach the target with acceptable acceleration.

Figure 17 shows a dataset with a normalized target `SPEED_TARGET` set to V = 1 m/s (3.6 km/h) until S = 0 m. As with most Configurable Functions, the inputs and outputs can be transformed using built-in parameters. In this case, the `SPEED_TARGET` gain and station offset are set in an **Events** dataset that is loaded when a stop sign is detected (Figure 18). (This is the dataset link for the first Event shown in Figure 8 and Figure 12.)


Brakes: Four-Wheel System | Control: Speed (Closed Loop) Using Path Preview | **Control: Speed (Closed Loop) Using Target Speed**

Function type: Constant ①
Constant Target forward speed (km/h): 0

Dataset with custom ID: STOP_ID ④
☐ Custom settings

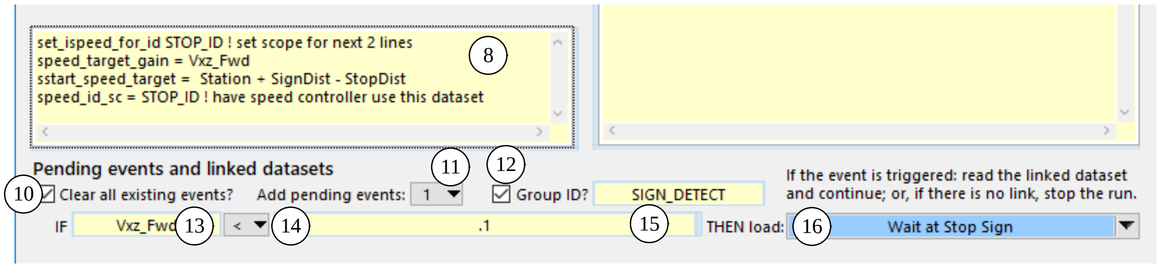
Add to | Step interpolation, flat-line extrapolation ②

Station, Target speed ③

	X Axis	Y Axis
1	-1	3.6
2	0	0

Time (s), station (m), and target forward speed (km/h) as specified in this dataset can be transformed using additional parameters. For details, search an echo file for "SPEED_TARGET".

Figure 17. Speed target dataset for constant deceleration to stop.


```
set_ispeed_for_id STOP_ID ! set scope for next 2 lines
speed_target_gain = Vxz_Fwd
sstart_speed_target =  Station + SignDist - StopDist
speed_id_sc = STOP_ID ! have speed controller use this dataset
```
⑧

Pending events and linked datasets
⑩ ☑ Clear all existing events? Add pending events: 1 ⑪ ☑ Group ID? ⑫ SIGN_DETECT
If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.
IF Vxz_Fwd ⑬ < ⑭ .1 ⑮ THEN load: ⑯ Wait at Stop Sign

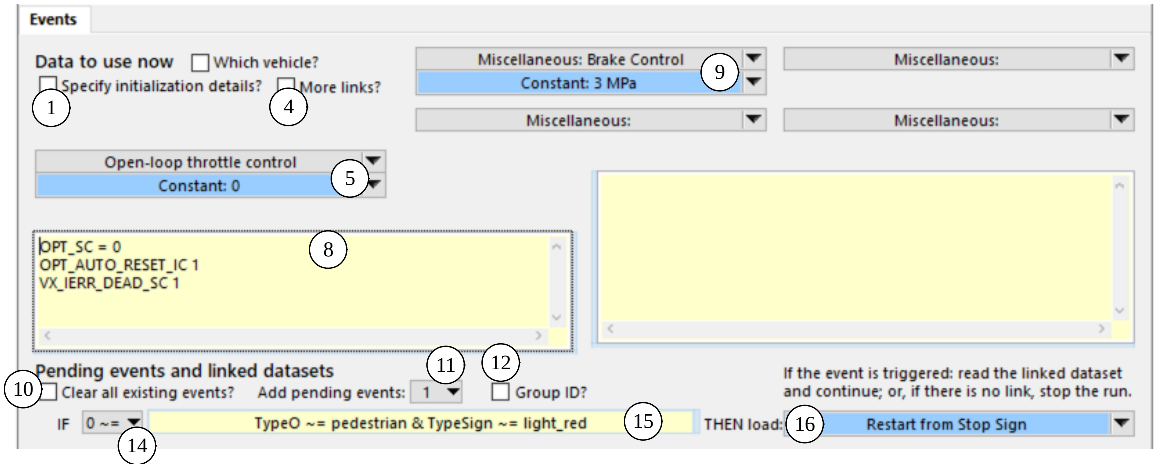
Figure 18. Event triggered when Stop Sign is detected.

The Miscellaneous field ⑧ has a command to set the gain `SPEED_TARGET_GAIN` to match the current forward speed of the vehicle: `Vxz_Fwd`. The offset for the independent variable (station) `SSTART_SPEED_TARGET` is set to the station value where the vehicle should stop, which is: `Station + SignDist – StopDist`, where `Station` is the location of the origin of the vehicle sprung mass, and `SignDist` and `StopDist` are parameters defined elsewhere.

If the vehicle stops moving, it is no longer possible to track changes based on motion (e.g., `Station` will not change). For this reason, a pending Event is defined to trigger just before stopping, when the target speed `Vxz_Fwd` ⑬ < ⑭ 0.1 km/h ⑮.

Notice that the group ID is set to `SIGN_DETECT` ⑫, a user-defined parameter defined elsewhere. Because the ID is specified, the checkbox **Clear all existing events?** ⑩ only applies to Events with the same ID. That is, pending Events with different ID values will remain in effect.

The dataset loaded when the vehicle is almost stopped (Figure 19) sets the throttle to 0 ⑤, turns off the speed controller with the command `OPT_SC` = 0 ⑧, and applies some brake pressure ⑨. It also adds a pending Event for two conditions based on user-defined parameters and output variables from two ADAS sensors: if there is not a detected pedestrian (`TypeO ~= pedestrian`) AND there is not a detected red light (`TypeSign ~= light_red`), then load the Parsfile to restart the vehicle motion ⑯. In this example, different parameters represent red lights and stop signs. If a stop sign is in view, it's OK to start. But if it's a signal, the vehicle waits until the light changes.


Events
Data to use now
Which vehicle?
Specify initialization details?
More links?
1
4
Miscellaneous: Brake Control
Constant: 3 MPa
9
Miscellaneous:
Miscellaneous:
Miscellaneous:
Open-loop throttle control
Constant: 0
5
8
Pending events and linked datasets
Clear all existing events?
Add pending events:
1
Group ID?
10
11
12
If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.
IF
0 ~=
14
15
THEN load:
16
Restart from Stop Sign

```
OPT_SC = 0
OPT_AUTO_RESET_IC 1
VX_IERR_DEAD_SC 1
```

TypeO ~= pedestrian & TypeSign ~= light_red

Figure 19. Event when waiting at a stop sign.

Steady-State Circle Series

ISO 4138 defines a long test series to determine understeer under steady turning conditions. The vehicle is driven at a constant speed on a constant radius path until it is in equilibrium, then the speed is increased and held constant until equilibrium occurs, and so on. Outputs are written only when the vehicle has settled into the new equilibrium. Although the simulated test might cover several minutes, there are typically only 40 or so recordings made, with the objective of creating cross-plots of variables of interest (Figure 20).

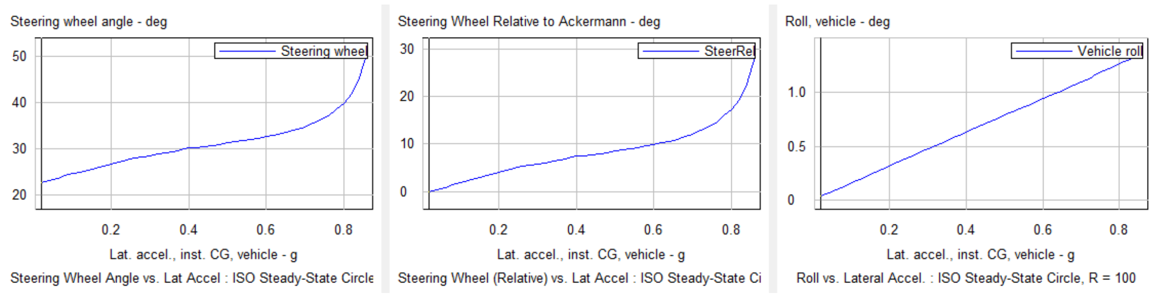


Figure 20. Cross-plots made in CarSim from steady-state turning per ISO 4138.

Example simulations in CarSim and TruckSim make use of five Event datasets. As noted earlier, Event titles often begin with letters or numbers if they will be applied in a sequence, so they can be viewed easily in the sequence in which they are normally triggered (Figure 21).

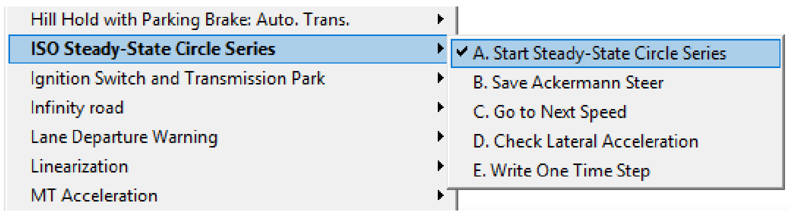
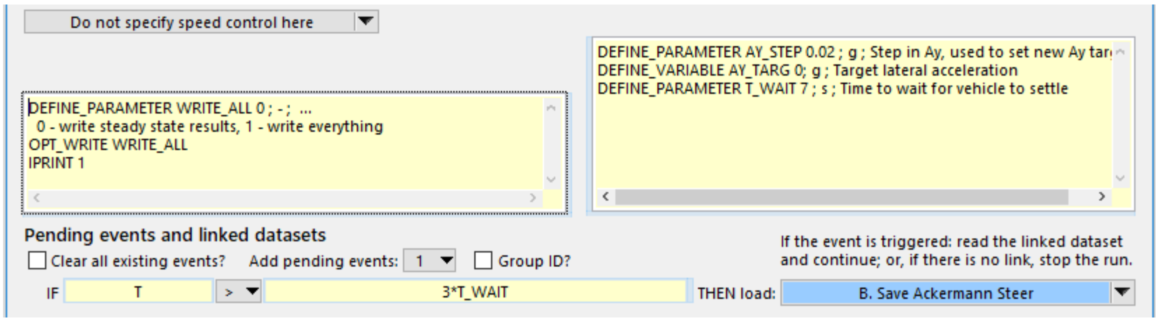

Hill Hold with Parking Brake: Auto. Trans.
ISO Steady-State Circle Series
Ignition Switch and Transmission Park
Infinity road
Lane Departure Warning
Linearization
MT Acceleration
✓ A. Start Steady-State Circle Series
B. Save Ackermann Steer
C. Go to Next Speed
D. Check Lateral Acceleration
E. Write One Time Step

Figure 21. Menu showing five Events for a Steady Turn series of tests.

Figure 22 shows the first Event in the series, which is linked to the **Procedures** dataset, and is therefore read by the VS Math Model before the simulation starts. The Miscellaneous fields use the VS Commands `DEFINE_PARAMETER` and `DEFINE_VARIABLE` to add new variables to the model, including one named `T_WAIT`. A pending Event is defined that will be triggered when `T > 3*T_WAIT`. When that occurs, the VS Math Model will load the linked Event "B" dataset (Figure 23).



Do not specify speed control here

```
DEFINE_PARAMETER WRITE_ALL 0 ; - ;  ...
 0 - write steady state results, 1 - write everything
OPT_WRITE WRITE_ALL
IPRINT 1
```

```
DEFINE_PARAMETER AY_STEP 0.02 ; g ; Step in Ay, used to set new Ay tar
DEFINE_VARIABLE AY_TARG 0; g ; Target lateral acceleration
DEFINE_PARAMETER T_WAIT 7 ; s ; Time to wait for vehicle to settle
```

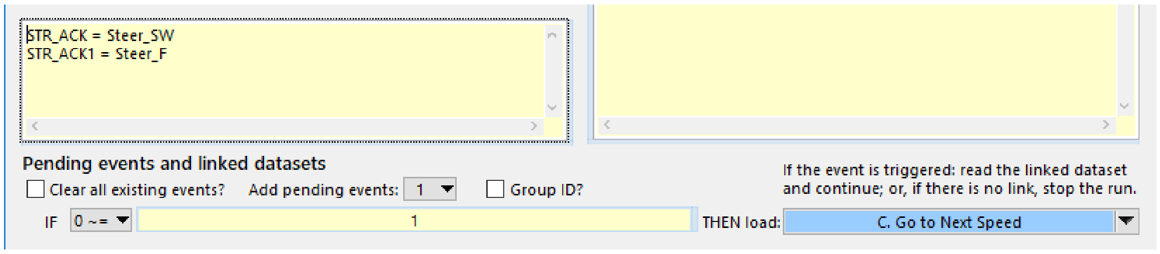
Pending events and linked datasets

☐ Clear all existing events? Add pending events: 1 ☐ Group ID?

If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF T > 3*T_WAIT THEN load: B. Save Ackermann Steer

Figure 22. Event: A. Start Steady-State Circle Series.



```
STR_ACK = Steer_SW
STR_ACK1 = Steer_F
```

Pending events and linked datasets

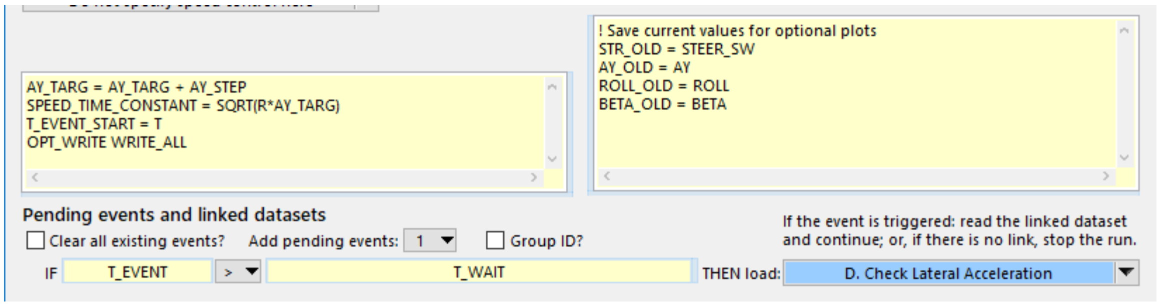
☐ Clear all existing events? Add pending events: 1 ☐ Group ID?

If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF 0 ~= 1 THEN load: C. Go to Next Speed

Figure 23. Event: B. Save Ackermann Steer.

The "B" dataset (Figure 23) saves two variables that are used later as references: these are the steer angles needed to follow the specified radius of the road at low speed, called Ackermann steer angles. One is the steering wheel angle (`Steer_SW`) and the other is the average of the two front road-wheel steer angles (`Steer_F`). A pending Event is set to be triggered unconditionally (0 ~=1) at the next time step, causing the linked "C" Event dataset (Figure 24) to be loaded.



```
AY_TARG = AY_TARG + AY_STEP
SPEED_TIME_CONSTANT = SQRT(R*AY_TARG)
T_EVENT_START = T
OPT_WRITE WRITE_ALL
```

```
! Save current values for optional plots
STR_OLD = STEER_SW
AY_OLD = AY
ROLL_OLD = ROLL
BETA_OLD = BETA
```

Pending events and linked datasets

☐ Clear all existing events? Add pending events: 1 ☐ Group ID?

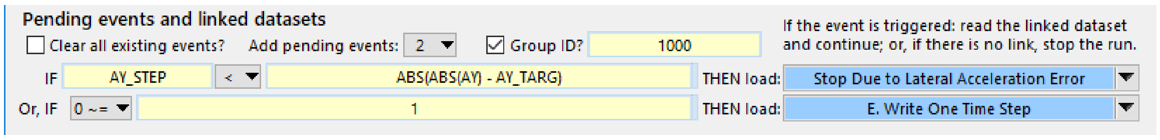
If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF T_EVENT > T_WAIT THEN load: D. Check Lateral Acceleration

Figure 24. Event: C. Go to Next Speed.

The "C" dataset (Figure 24) increments a target lateral acceleration `AY_TARG` and calculates a new target speed with the formula `SQRT(R*AY_TARG)`, where `R` is the radius of the circular path.

Writing to file is controlled by the parameter OPT_WRITE. It is set to a user-defined parameter WRITE_ALL that is typically zero. (The WRITE_ALL parameter is provided to enable writing of the entire time history of the simulation easily if needed for diagnostics.) The T_EVENT clock is reset by setting T_EVENT_START = T. A new Event is defined that will trigger when T_EVENT is greater than T_WAIT and cause the "D" dataset to be loaded (Figure 25).


Pending events and linked datasets

☐ Clear all existing events? Add pending events: 2 ☑ Group ID? 1000

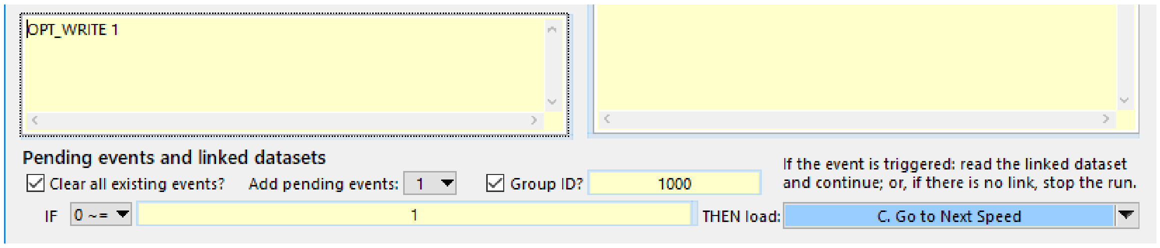
If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF AY_STEP < ABS(ABS(AY) - AY_TARG) THEN load: Stop Due to Lateral Acceleration Error

Or, IF 0 ~= 1 THEN load: E. Write One Time Step

Figure 25. Event: D. Check Lateral Acceleration.

The "D" dataset (Figure 25) adds two new pending Events. The first Event checks to see whether the current lateral acceleration (Ay) differs from the target AY_TARG by more than the step AY_STEP. If it does, then the vehicle has reached its limit in lateral acceleration and the VS Math Model will load a Parsfile that will cause the run to Stop. Otherwise, it will unconditionally load the "E" dataset (Figure 26).


```
OPT_WRITE 1
```

Pending events and linked datasets

☑ Clear all existing events? Add pending events: 1 ☑ Group ID? 1000

If the event is triggered: read the linked dataset and continue; or, if there is no link, stop the run.

IF 0 ~= 1 THEN load: C. Go to Next Speed

Figure 26. Event: E. Write One Time Step.

The "E" dataset (Figure 26) sets the parameter OPT_WRITE = 1, so the VS Math Model will write to file at the completion of the time step. The next time step, it will unconditionally load the linked Event dataset "C" shown earlier (Figure 24) for increasing the target speed for the next step in lateral acceleration.

The simulation will continue going through Events C, D, and E until the acceleration no longer increases, at which time the run will be stopped as a result of the check made in Event "D" (Figure 25).

Limitations on Creating and Changing Variables with Events

Recall that top-level Event datasets are read by the VS Math Model before the run starts, and other Event datasets are read after the simulation has started, as the result of another Event being triggered.

Some parameters and VS Commands can be handled before the run starts, but become locked or invalid once the simulation is running.

The following cannot be changed in an Event after the run starts:

1. Parameters identified in the Echo file as locked [L] (e.g., TSTEP).
2. VS Commands DEFINE_IMPORT, DEFINE_TABLE, and EQ_DIFFERENTIAL.

If an attempt is made in an Event to change a locked parameter or apply a block VS Command, the VS Math Model will generate an error message noting the illegal action.

Overriding Data

Data from the **Run Control** screen and linked datasets are used by two kinds of programs, each with different effects. In both case, the VS Browser generates a single *All Parsfile* that contains the contents of the **Run Control** dataset plus all directly linked datasets. This All Parsfile does not include Parsfiles that are linked to Events that trigger after the run has started.

Simulation Modules (VS Math Models)

When you click the **Run** button to run a simulation, the VS Browser creates a new All Parsfile that is immediately read by a function in the VS Solver library, and used to construct a VS Math Model that processes the rest of the All Parsfile. Much of the data in that file provide values for parameters and tables that are built into the VS Math Model. If multiple settings are provided for the same parameter or table, the last one read is used in the simulation. For example, one brake control dataset can be specified in the linked **Procedures** dataset and another on the **Run Control** screen. If the dataset from the **Run Control** screen is read last, then it will be used because it will have overridden the values for parameters in the model.

With this in mind, it is usually a good idea to provide all driver controls in a **Procedures** dataset. If the procedure will be used with a different setting (e.g., a different speed target), the alternate setting can be applied from the **Run Control** screen to override the setting from the **Procedures** dataset.

Not all datasets apply to parameters and tables that already exist. Some datasets have commands that add modules to the simulation. These include payloads, sensors, paths, roads, and moving objects. These also include target speed and lateral target Configurable Function datasets, which are used to generate driver controls and also to control moving object.

For CarSim and TruckSim, the datasets from leading units (e.g., **Vehicle Assembly** datasets in CarSim) add a vehicle to the simulation.

VS Visualizer

When you click a **Video** or **Plot** button to view results, the VS Browser also creates a new All Parsfile that is immediately read by VS Visualizer, with the same content that was sent to the VS Math Model to run the simulation.

Video

There is a fundamental difference in the concept of modeling with a VS Math Model and VS Visualizer. The VS Math Model includes a math model with at least a full vehicle model, with parameters and Configurable Functions that have default values. On the other hand, there is no predefined 3D world for the VS Visualizer. It is a tool used for many scenarios involving 3D objects, which might not even include a vehicle. Each part of the vehicle or environment that appears in a video was added by the All Parsfile.

Repeated datasets do not override anything in VS Visualizer; they always add to the visual environment. It is common to define many 3D objects using the same animation asset, with each copy given a unique location. For example, many vehicle animations use the same shape for all tires.

Another difference between VS Math Models and VS Visualizer is that VS Visualizer does not read any of the Parsfiles that the VS Math Model might read after the run starts. This means that any animation or plot setup data that is referenced from a Parsfile that is read when an Event is triggered will not be used.

Plots

Links to datasets from the **Plot: Setup** library are used to generate separate plots. Links can be made on the **Procedures** screen and any other screens where the assumption is that certain plots should always be generated if a link is made to that screen. If the same plot setup is linked multiple times, then duplicate plots are generated.

Road Surfaces

When reading a dataset for a road, the **Road: 3D Surface (All Properties)** screen has an option that helps determine whether the dataset will be used to redefine an existing road or add a new one to the simulation. This option is in the form of a user-defined ID number.

VS Visualizer ignores any information about road parameters and ID numbers. However, there are 3D surfaces associated with a Road surface dataset, those shapes are added to the animation if specified at in the run setup. However, if a road surface is added when processing an Event, the associated animation shapes will not be shown by VS Visualizer.

If multiple roads will be used in a simulation, the recommendation is to define them using a Generic screen such as a **Generic Group of Links**, provide each road with a unique ID number, and then in the Events, specify the road ID to be used by the vehicle using the parameter `CURRENT_ROAD_ID`, followed by the user ID of the road of interest, e.g.:

```
CURRENT_ROAD_ID 1234
```

Troubleshooting Examples

This section includes troubleshooting steps related to issues that might arise when setting up **Procedures** and **Events** datasets.

Procedure Datasets

The **Procedures** screen is the standard location in a VehicleSim product used to set up a test scenario. Although it is not the only screen that can be used, the confluence of Driver Controls; Misc. links and yellow fields (including links to Events); fields for setting the start and stop conditions; access to the vehicle initialization options; and links to **Plot: Setup** datasets makes the **Procedure** screen the best place to start when constructing a new test.

What is the minimum set of driver controls required to run a simulation?

A vehicle math model in CarSim, TruckSim, or BikeSim is expecting four core controls:

1. Throttle (open loop always, plus optional closed loop)
2. Braking (open or closed loop)
3. Transmission shifting (open or closed loop)
4. Steering (open or closed loop)

These can be set directly in the VS Browser using the provided library and dataset links, and/or with import variables using VS Commands in Misc. yellow fields and/or the **I/O Channels: Import** screen. They are typically set from the **Procedures** screen or the **Run Control** screen.

Note that open-loop throttle is always used, even if the speed controller is active. In that case, the throttle from the speed controller is added to the open-loop throttle. This can complicate switching modes with Events, as described later.

What happens if there is no link to a driver control?

When no driver control input is specified, the VS Math Model uses default settings and data. Although the default data may be acceptable in some cases, we don't recommend counting on default settings. Sometimes unlinking a dataset for a required library and relying on default data can prevent a simulation from running as expected due to an incompatible control option with an import variable, or vehicle data (e.g., tires or suspension) that may be completely inappropriate for the sprung mass and powertrain selections.

Here are a few reasons to always set driver control behavior:

1. The default data may be inappropriate for the simulation.
2. Linking to a dataset suggests a decision was made concerning that library, e.g., Open Loop Throttle vs. Time. When viewed sometime later, or by someone else who did not build the original simulation, it may not be clear why a library does not have a linked dataset: was a dataset not linked on purpose, or was it unlinked by accident? That doubt can lead to a lot of time spent troubleshooting a problem that does not exist.
3. Linking a dataset to a library gives easy access to the GUI screen associated with that dataset, and on that dataset additional information exists that includes the math model keyword for that option (i.e., a data field with right-click Notes or a Configurable Function table). The Echo files that the math model creates include these math model keywords along with their values representing the math model input data.

   When troubleshooting a simulation, the Echo files contain crucial information. Navigating the Echo file is much easier if you can browse to a screen, see the math model keyword, and then search on that keyword in the Echo file. Without a direct link to a given dataset from the current simulation, searching the Echo file to a parameter value is time-consuming unless you already know what the math model keyword is.

What is the minimum data required to run a VehicleSim simulation?

It is possible to run a simulation with no input data, although the results are probably useless for anything except confirming that the software exists and generates output files.

In general, if a screen that is used in a run has links dedicated to specific libraries, then those links should be used. For the **Run Control** screen, there are dedicated links for a vehicle dataset, a **Procedures** dataset, and an **Animator: Camera Setup** dataset. In most cases, all three are used.

If we look at a vehicle screen (e.g., in CarSim, **Vehicle: Assembly**), dedicated library links exist that represent different aspects of the vehicle as a system. Those links should also be used.

In what order are the links on a screen read and processed?

The answer is: it depends on which screen we are looking at. There are two ways to obtain this information:

1. View the dataset Parsfile (click the Parsfile icon at the top of the GUI window), and/or
2. View the **Linked Library Tree** in the lower left corner of the GUI window.

Each screen has unique requirements when it comes to how the data must be sent to the VS Math Model and in what order. It is those requirements that dictate the order in which datasets are read, and from the standpoint of the end-user, will determine where overriding content can be linked. Do not assume that each screen reads Parsfile s top to bottom, left to right. In fact, this is rarely the case.

The order in which links are processed is especially important when it comes to writing VS Commands that will either depend upon previously defined content (e.g., modifying the gain of a table), or be used to set up content that will be read later (e.g., defining a new parameter for use in Events). It is for this reason that Figure 1 (page 2) shows the sequence in which information from the screen is written to the dataset Parsfile.

Suppose you want to write some VS Commands on a **Procedures** screen that modify one or more of the Driver Controls, also specified on the **Procedures** screen. Either of the two Misc. yellow fields on the **Procedures** screen can be used to modify the Driver Controls located in the upper left portion of the screen, since these Misc. yellow fields appear in the Parsfile after the four Driver Controls.

One additional point: what happens if you link to a Closed Loop Speed dataset on the **Procedures** screen, and then link to it again (whether the same speed dataset or a different one) on the **Run Control** screen? In the context of the entire simulation, we must also consider where the screen is linked. In this case, the answer is: the VS Math Model will use the one from the **Run Control** screen. (This can be determined by viewing the **Linked Library Tree** for the **Run Control** dataset, which shows the **Procedures** link before the Speed Target link; the last one wins.)

If we are unsure about which data is being used for the Closed Loop Speed Controller, we can check the Echo file with Initial Conditions and search for the following:

- `DRIVER MODEL`
- `SPEED_TARGET`

The Echo file section `DRIVER MODEL: SPEED CONTROLLER` will indicate that there are two `SPEED_TARGET` datasets installed (`N_SPEED_TARGET = 2`), and that the `SPEED_ID_SC = 2`, indicating that the Closed Loop Speed Controller is using the second `SPEED_TARGET` dataset.

How do I import target speed for the speed controller?

If you want to use an imported speed command for the simulation (e.g., from Simulink), there are two requirements:

1. You must enable the built-in speed controller.
2. You must setup the connection to the Simulink model to import target speed.

The speed controller (throttle and braking) is not installed unless you link to closed-loop speed control dataset. This is typically done from the **Procedures** screen. If you do not provide this link, the math model will not use the speed controller, and no matter what comes in from Simulink, it will not be used.

To import a target speed (e.g. from Simulink), select the variable `IMP_SPEED` on the **I/O Channels: Import** screen for use with your Simulink model. Choose “replace” so the imported variable replaces the target speed generated internally.

Why did the simulation stop before reaching the Stop Time and/or Station settings?

If a simulation ends irrespective of the Stop Time and Stop Station settings on the Procedure screen and it is not clear why, then check the end of the Log file, found using the drop-down menu in the lower right corner of the **Run Control** screen. The Log file always states why the simulation ended.

Typical reasons are:

- Stop time reached.
- Station limit reached.
- Limit reached on road surface.
- Event triggered.
- Error occurred (with description).

Event Datasets

Events are not difficult to use, but due to the number of options available when creating an Event series, care must be taken to build them to ensure that the flow of the test proceeds as expected.

Why are my Events not being triggered?

The first step in diagnosing Event triggering is to look at the end of the Log file, available by using the drop-down menu in the lower right corner of the Run Control screen. The Log file will contain:

- The list of Events that were triggered for that run, with the Group ID if applicable.
- At what time they were triggered.
- Which dataset(s) were loaded with each Event.

If we are expecting an Event to be triggered but it was not, we can use the Events dataset list at the end of the Log file to see which Events were triggered. Based on that information, we can browse to the Events dataset that was supposed to be used to trigger our expected Event (i.e., the last one successfully processed) as the starting point for our investigation.

When Events don't trigger as expected, many possibilities exist. Things to look for include:

- Boolean operator (e.g., it is <=, but should be >=).
- Sign on the formula and variables (e.g., If I am checking the roll angle of the sprung mass, should I be checking for the absolute value of the roll angle, `ABS(ROLL)`?).
- Magnitudes of the formula or variables.
- Whether it is possible the trigger condition can be met (e.g., we are checking for `Vxz_Fwd` > 50 kph, but the vehicle never achieves that speed).

From a simulation building standpoint, one approach is to add Events one at a time and link each to the End Events dataset (included as part of the shipping database for VehicleSim products). The advantage to this approach is that you can make the run, look at the end of the Log file, and see if the newest Event triggers the dataset End Events.

- If it does, then you have set up our Event as intended and can therefore continue building our Events series.
- If the End Events dataset does not get triggered, however, then you know that the issue is with the most recent Event that was added.

The End Events dataset has the VS Command `STOP_RUN_NOW` specified in a Misc. yellow field. When the VS Math Model reads this VS Command, the VS Math Model terminates the run with a message specified in the command.

I've started with a new Events dataset. Why doesn't the vehicle move?

When creating a new, empty **Events** dataset, the default behavior is for the box "**Specify initialization details?"** to be checked, but all other boxes are unchecked. This causes the three `OPT_INIT` parameters to be set to zero: `OPT_INIT_PATH = 0`,`OPT_INIT_SPEED =0`, and `OPT_INIT_CONFIG = 0`.

These settings are appropriate if the Event dataset is loaded after the simulation starts, but not if this is a top-level Event dataset linked to the **Run Control** or the **Procedure**s screen. If this is a top-level dataset, it disables the normal initialization that would set the vehicle speed to match a target.

Note The default settings of the checkboxes for a new empty data were set to be compatible with older versions, in which the checkbox "**Specify initialization details?"** did not exist. The recommended setting is to uncheck that box, unless you have a specific reason to change something.

How do I switch between Open Loop Throttle and Closed Loop Speed Control?

As mentioned earlier, the throttle applied to the powertrain is the sum of the open-loop throttle (from the Configurable Function `THROTTLE_ENGINE`) and possibly throttle from the speed controller. If the speed controller is disabled (`OPT_SC` = 0), the throttle comes mainly from the `THROTTLE_ENGINE` function (with modifications for shifting and/or Imports). If the speed controller is active (`OPT_SC` > 0), the throttle includes a request from the speed controller plus the open-loop throttle from `THROTTLE_ENGINE`.

Using an Event to switch from closed-loop speed control to open-loop is easy: just link to dataset from the **Control: Throttle (Open Loop)** library, using the dedicated speed control link (⑤, Figure 8, page 12). The linked dataset will apply the intended dataset and also turn off the speed controller.

Switching from open-loop to closed-loop is more complicated, because you need to turn off the open-loop contribution to throttle. There are several ways to do this.

- If you are linking to a speed control dataset, then use the dedicated speed control link (⑤, Figure 8) to link an open-loop throttle dataset with zero throttle. Use a miscellaneous link ⑨ to link to a speed controller dataset with the intended target speed. The sequence of the links is important here: The linked open-loop dataset includes a command to turn off the speed controller; however, the next dataset (closed loop) includes a command to turn it back on.
- If you want to set a constant speed from the Event screen (Figure 11, page 12), you can set the open-loop throttle table to zero with a line of text in a miscellaneous field: `THROTTLE_ENGINE_CONSTANT 0`. This will not affect the speed controller setting (`OPT_SC`) and ensures there is no open-loop throttle.

What is the correct way to apply a time offset for a control?

All of the open-loop driver controls can be defined as a Configurable Function of time. Also, the speed controller target may also be specified as a function of time.

The default interpretation is that the independent variable in those functions matches the simulation time, represented in VS Command formulas with either the keyword `T` or `T_STAMP`.

However, when a control dataset is linked to an Event that is triggered while the simulation is in progress, it is often convenient to reset time such that T = 0 in the table corresponds to the time that the Event is triggered. The Configurable Functions are designed to handle this, as described earlier (page 110, using time-offset parameters that are associated with each table (Table 1, page 11).

There are two ways to reset the clocks on the driver control tables using the time-offset parameters:

1. From an Event in which one or more of these controls will be used, check the "**Specify initialization details?**" box ① (Figure 9, page 13) and also check the box "**Reset control clocks**?" ② (Figure 9). This will reset the control clocks for all open-loop controls.
2. Assign the time-offset parameter of interest to the current time, using one of the miscellaneous filed. The advantage to this approach is that only the Event time offsets of interest are used. For example, to reset the clock for open-loop brake control, type the line `TSTART_BRAKES = T` into one of the miscellaneous fields.

The second approach has several advantages:

- Only the Event time offsets of interest are specified.
- The vehicle math model is not unintentionally subject to a reinitialization (or a lack thereof) as a function of checking (or not checking) the three vehicle initialization checkboxes.

Recall the previous comments regarding the use of the "**Specify initialization details?**" on the first Event in a series.

- It facilitates learning the math model parameters.

How do I use a Generic Table in an Event?

Using a Generic Table in an Event requires that the math model keyword for the table be installed before the run starts, using the VS Command `DEFINE_TABLE`. Within the Events the table is called with a VS Command such as `EQ_IN` or `EQ_OUT`. Note that both CarSim and TruckSim include Events-based examples that demonstrate this technique.

For example, on the **Procedures** screen:

- Link to a **Generic VS Commands** library and make a dataset with the command: `DEFINE_TABLE MYTABLE 1`.
- To view the output of the table in plots, make a new output variable: `DEFINE_OUTPUT MY_VARIABLE = 0`.

On the **Events** dataset where the table data will be applied:

- Link to the **Generic Table** dataset, making sure the root keyword is the same as the table definition using `define_table`.
- Use a VS Command to refer to that table, and apply the table output to some other part of the model.

In a Misc. yellow field on the Events screen, the VS Commands syntax would look like this:

```
eq_in my_variable = my_table(0, t, 1)
```

This table uses a single independent variable (time, `t`), with an index of 1. Since the table does not contain a second independent variable, the first argument is 0.

Why are VS Visualizer plots blank, with a video of objects in the wrong locations?

There will be no plots if the there is no output data for VS Visualizer. If this happens, any video will show all vehicle parts at the global origin.

It is possible for a VS Math Model to create output files with no content. The most common reason for this condition is that the parameter `OPT_WRITE` is set to zero in a Misc. yellow field on the **Procedures** screen or an **Events** screen, with the intention that writing to file is turned off when the run begins because we are waiting for the vehicle to perform some set of tasks — e.g., get up to a test speed, reach a certain position on the path, etc. — and we are not interested in saving this initial data.

If it is not apparent from looking at the **Procedures** and **Events** dataset where `OPT_WRITE` is being set to zero, you can use the **Find Text** option (**Tools > Find Text in the Database**.

As reference, consider the CarSim example involving a Fishhook handling test.

The Event-driven Fishhook test has three sets of pending Events linked on the **Procedure** screen, the first of which to get the vehicle up to a target test speed. The second Event series starts where the first series leaves off, beginning with a Slowly Increasing Steer maneuver. We aren't interested in saving the simulation results when the vehicle is getting up to the target test speed, but we are interested in saving the data for the Slowly Increasing Steer portion of the test. For this reason, `OPT_WRITE` is set to zero until the first Event series completes, and set to one at the beginning of the second Event series, thereby turning on the functionality to write to file. If the vehicle never completes the first Event series by reaching the target test speed, the second Event series never begins, and `OPT_WRITE` never changes from zero to one. Therefore, when we look at the simulation results in VS Visualizer, we will see that the simulation ran for 360 seconds (i.e., the Stop time set on the **Procedure** screen using `TSTOP`), but the Slowly Increasing Steer and Fishhook portions of the test would never have been run, and there will not be any data available for plots and animation.

Why does the vehicle move even though Constant Target Speed = 0 km/h.

The option Constant Target Speed is a unique use-case of the `SPEED_TARGET` Configurable Function: the Closed Loop Speed Controller is enabled, but default values are used for the three controller gains, and the options to account for engine braking and the vehicle's brake system are turned off by default.

When the option Constant Target Speed is set to 0 km/h, the Closed Loop Speed Controller is able to calculate a throttle, but does not apply the vehicle's brakes. Without the brakes, the vehicle will move if it is in gear.

If the goal is to have the vehicle start at rest and/or come to rest during the test, then the better choice of controls might be:

- Throttle + Initial Speed, where throttle = 0 and initial speed = 0 km/h.
- Brake Control (e.g., Constant Master Cylinder Pressure = 1 MPa, or whatever is necessary to hold the vehicle at rest. Pedal Force control is also available; check the Brake System dataset for the compatible control setting)

Events can then be used to change the control options by releasing the brakes, and applying throttle either via Open Loop Throttle or the Closed Loop Speed Controller.

Why does the vehicle go too fast down a hill with Constant Target Speed?

As noted above, selecting the Speed Controller option Constant Target Speed means access to the vehicle's brake system is turned off by default. Instead of using the short-cut constant speed option, link to a Speed Target vs. Time/Station dataset, and set the speed controller parameters directly.