

Making Advanced Procedures with VS Commands

Basic Concepts for Using VS Commands	2
Example: Steady-State Circular Turning (ISO 4138).....	2
Understeer Gradient	3
The ISO 4138 Constant-Radius Test.....	4
Defining the Test with Events and Other VS Commands	4
The Circular Track.....	7
The Procedure Dataset	9
Time Step in the Output Files.....	10
The Chain of Events for the Understeer Test	11
Stopping the Run.....	17
Lateral Tracking Error	17
Speed Error.....	19
Steering Limits	19
The Programming Process	21
Simulate the Basic Test	21
Automate a Series of Tests with Events	22
Refine and Extend the Procedure	22
Review of the Methods Used	23
Programming Approach.....	23
Use of Support Tools in the Browser	25

VehicleSim (VS) products support a scripting capability through VS commands. This capability can be applied to simulate complicated performance tests such as those defined in standards and regulations. The detailed reference material for VS commands is provided in the [VS Commands Reference Manual](#).

This tech memo explains how to use VS commands to automate sequences of tests with features such as the following:

1. Results from early tests are used to define conditions for subsequent tests.
2. Tests are repeated with increasing severity until a desired limit is reached (e.g., the vehicle passes the test), or until an unacceptable condition occurs (e.g., the vehicle fails).
3. Transitions from one step in a test to the next can be based on almost any criteria involving thousands of variables in the simulation.
4. Statistics or measures are obtained that are specific to a particular test or standard.

The general approaches are illustrated with an example procedure that is provided for CarSim 8.1.1 and TruckSim 8.1 — a constant-radius understeer test (ISO 4138) used to characterize vehicle handling. If you are not familiar with understeer testing, don't worry: this is not a memo about handling or understeer: rather, it's about advanced capabilities of CarSim and TruckSim

that can be used to script standard test procedures. Please keep reading, and imagine how the same methods shown here would be used in your application of interest.

This memo assumes that you are familiar with the basic use of your VS product. At a minimum, you should have gone through the Quick Start Guide. The memo also assumes some familiarity with the way VS solvers work, documented in the [VS Solvers Manual](#).

Basic Concepts for Using VS Commands

VS commands are used to extend the capability of the software tool to handle conditions that are not built in, but which can be defined at runtime. Some capabilities that will be demonstrated are:

1. Parameters and variables can be assigned values using formulas involving other parameters and variables in the math model.
2. Any parameter or variable can be monitored to trigger an *event* in which simulation conditions are modified (i.e., vehicle properties can be changed, controls can be changed, the run can be terminated, etc.).
3. New variables can be defined at runtime as needed to directly describe a specific custom test procedure.
4. New equations can be added at runtime for evaluation at critical times in the simulation, such as initialization, every time step, etc.
5. A few advanced settings will be used that require entering keywords and values in miscellaneous yellow fields. These include parameters that modify configurable functions and a parameter that determines whether results are being written to file for subsequent plotting and animation.

The VS commands that are used in the examples in this memo are summarized in Table 1.

Table 1. VS Commands used in this memo.

Command	Action
DEFINE_EVENT*	Define a new pending event.
DEFINE_OUTPUT	Define a new output variable for export, plotting, and animation.
DEFINE_PARAMETER	Define a new parameter for the solver.
STOP_RUN_NOW	Stop the run and write a custom message into the log file.

* DEFINE_EVENT commands are generated automatically by controls on the **Events** screen.

Example: Steady-State Circular Turning (ISO 4138)

Steady-state circular turning is used to obtain the understeer gradient for a vehicle, which is considered a fundamental characterization of vehicle dynamics and road-holding ability.

If you have no interest in understeer, you can skim through the next two subsections to page 4: “Defining the Test with Events and Other VS Commands.”

Understeer Gradient

Understeer gradient is a measure of how the steering needed for a steady turn changes as a function of lateral acceleration. Steering at a steady speed is compared to the steering that would be needed to follow the same circular path at very low speed (this is called the Ackermann steer angle). The vehicle has a positive understeer gradient if the difference between required steer and the Ackermann steer increases with respect to incremental increases in lateral acceleration; it has a negative gradient if the difference in steer decreases with respect to lateral acceleration.

A vehicle is said to understeer when the understeer gradient is positive; a vehicle is said to oversteer when the gradient is negative; a vehicle is said to have neutral steer when the gradient is zero.

Several tests can be used to determine understeer gradient: constant radius (repeat tests at different speeds), constant speed (repeat tests with different steering angles), or constant steer (repeat tests at different speeds). The Ackermann angle remains constant when the constant-radius method is used, and the gradient of steering wheel angle with respect to lateral acceleration is therefore a valid measure of understeer gradient in this specific test.

Figure 1 shows the relationship between steering wheel angle and lateral acceleration for steady-state turning on a circular track with a radius of 100 m for an example vehicle provided in CarSim. Results from two runs are overlaid: a baseline (the top blue line), and one where the tires were changed (the bottom red line).

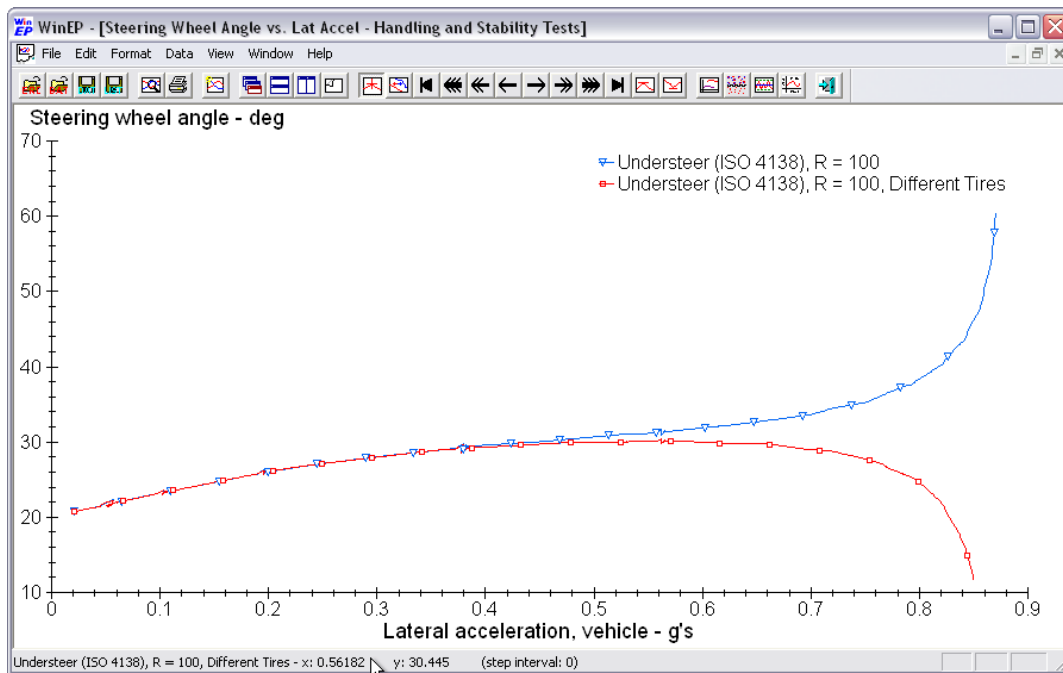


Figure 1. Understeer behavior shown for two vehicle configurations for a circular track with a radius of 100 m.

In this example, the gradient is always positive for the baseline, meaning the vehicle always understeers for the conditions covered in the simulated test.

The red line shows performance simulated for the same vehicle on the same 100-m radius path, but with the tire properties modified slightly to provide more grip in front and less in the rear. In this case, the steering increases (understeer) up to a lateral acceleration of 0.56 g's. At 0.56 g's, the gradient is zero, indicating neutral steer. As the acceleration increases the gradient becomes negative, indicating that the vehicle is oversteering.

Both plots show that the Ackermann angle at the steering wheel for this vehicle is about 20° (on the 100-m radius path) — the value when lateral acceleration approaches zero.

Note Ackermann angle for a vehicle steered by the front wheels is usually defined formally as the average of the front wheel steer angles. However, when testing, it is often convenient to associate the name with the corresponding angle measured at the steering wheel (the hand wheel).

The ISO 4138 Constant-Radius Test

The test used to generate the plots from Figure 1 is ISO 4138 (Passenger cars — Steady-state circular driving behaviour — Open-loop test methods), which describes several tests to determine understeer gradient. A second ISO standard is targeted at trucks and other heavy vehicles: ISO 14792 (Road vehicles — Heavy commercial vehicles and buses — Steady-state circular tests). The passenger car standard has more details and specifications and was used to develop the example datasets described in this memo.

Both ISO standards (4138 and 14792) recommend a radius of 100-m as a standard for the constant-radius test, and both allow other radii to be used. ISO 4138 limits the radius to a minimum of 30 m with a recommendation that the radius should not be less than 40 m. Figure 2 compares results obtained with 30-m and 100-m radii. Large differences are seen in the angles of the steering and front road wheels (top two plots). However, the gradients and overall curves are quite similar when the Ackermann steer is subtracted (bottom two plots).

ISO 4138 describes several methods for generating plots of steering vs. lateral acceleration. This memo covers the constant-radius test made with slowly increasing speed. In this test, the vehicle is driven at a low speed around a circular path; when the response has settled, the observed steering wheel angle is defined as the Ackermann angle for the path radius. The understeer behavior is then determined by driving with a slowly increasing speed, with the restriction that the rate at which lateral acceleration increases must be limited to 0.1 (m/s²)/s. (For a constant radius path, lateral acceleration $A_y = V^2/R$.)

Defining the Test with Events and Other VS Commands

The constant-radius test is not hard to set up in CarSim or TruckSim: provide a circular track, run the vehicle with closed-loop steering, and use the closed-loop speed controller to maintain a constant low speed long enough to reach a steady state and determine the Ackermann steer angles. Then, run on the same circular track with slowly increasing speed to increase lateral acceleration by 0.1 (m/s²)/s until the road-holding limits are reached.

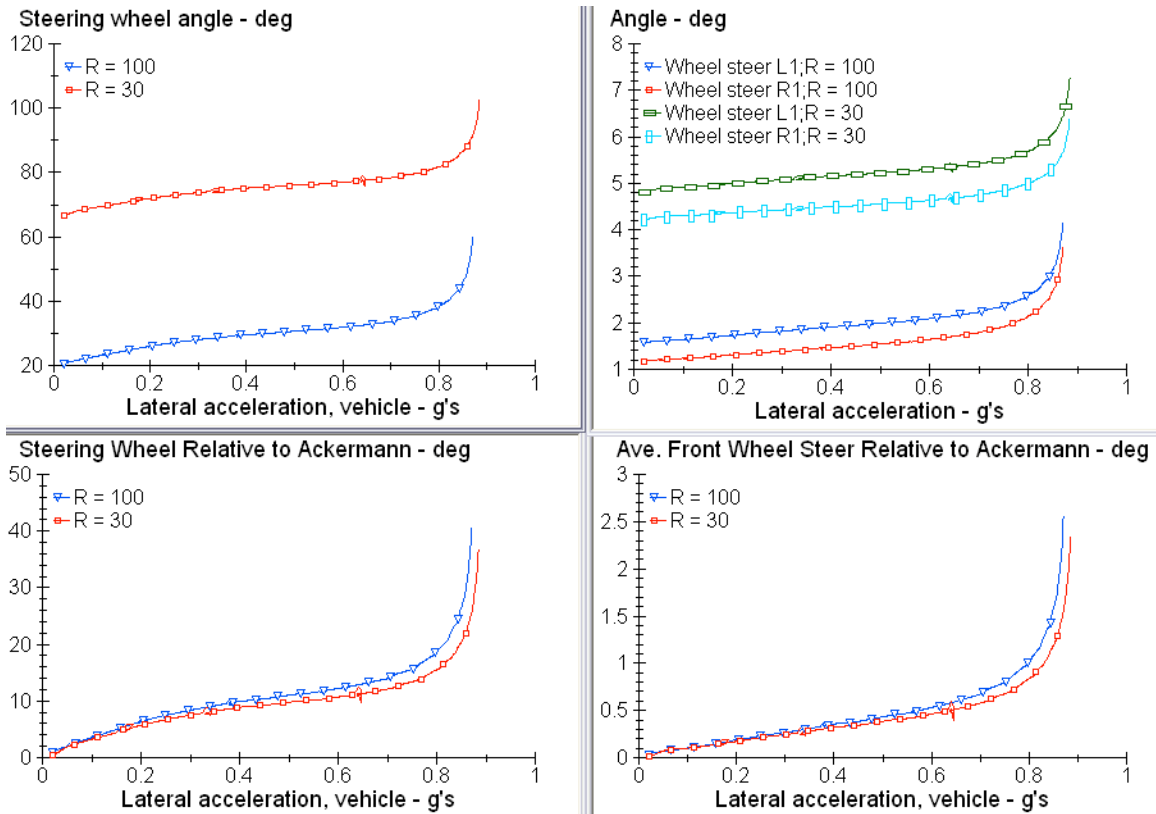


Figure 2. Comparison of results for 30-m and 100-m radius paths.

VS commands are used to extend the simulation environment to automate the generation of the types of plots shown earlier. Here are some considerations concerning the example procedure:

- Two phases of the test are needed: constant-speed conditions to find the Ackermann angles, and slowly-increasing speed to generate plots showing gradients relative to lateral acceleration.
- Plots are made relative to Ackermann steering; therefore, new output variables are defined that have the Ackermann steer removed (e.g., the bottom plots in Figure 2).
- ISO 4138 specifies lateral acceleration increasing at 0.1 m/s^3 ; therefore, the speed target should result in lateral acceleration increasing at a specified constant rate for the given path radius. However, for validation and debugging purposes, it should be easy to change this rate.
- Although ISO suggests that the turn radius be 100 m, there is some sensitivity to the radius and the procedure should allow other radii to be simulated easily.

There are really only two parameters for this test: (1) the radius of the turn, and (2) the direction (steering left or right). However, we might want to make adjustments for some vehicles (e.g., the amount of time needed to establish steady-state in the low-speed testing).

Part of the automation involves stopping the simulation. Some of the conditions for stopping can be monitored once the basic test is functioning, and will be discussed in a later section (page 17).

Figure 3 shows part of the **Run Control** screen used to make the example run for a CarSim example vehicle (B-class sports car) with a 100-m radius turn. (This run can be found using the **Datasets** menu in the category **Handling and Stability Tests**.) Three links are used to set up a run: the vehicle (1), the procedure (2), and the road (3). The road is linked on this screen (rather than from within the **Procedure** screen) to make it easy to apply the procedure for a different radius (i.e., just link to a different road dataset to change the radius).

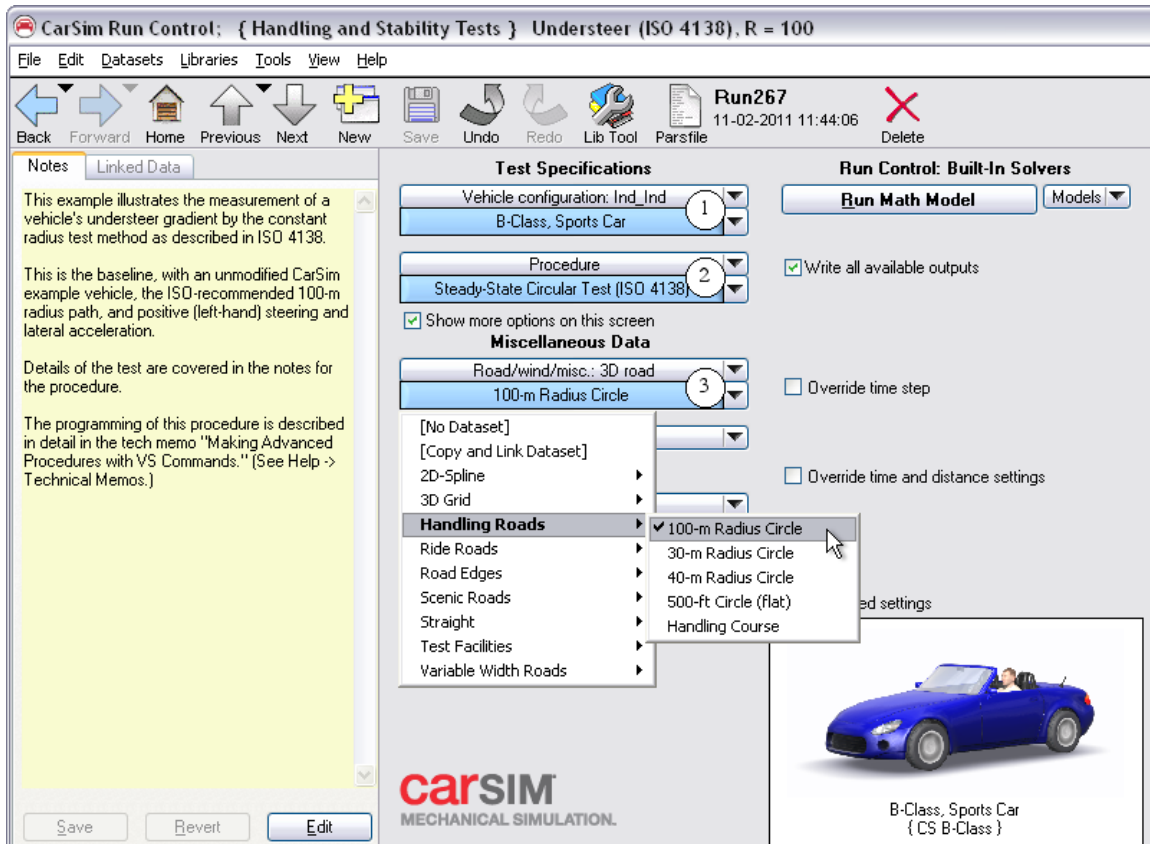


Figure 3. Run Control screen for steady-turn simulation ($R = 100$ m).

The vehicle normally travels along a road or driver target path in the direction of increasing station (longitudinal distance along a path), unless specified otherwise. As indicated by the sign convention in the plots, the tests involved positive lateral acceleration, associated with turning to the left. Results can be obtained for right-hand turns (negative lateral acceleration) from the **Run Control** screen, as shown in Figure 4. Check the box to override time and distance settings (1), choose the option to stop the run at a specified time (2), leave the yellow fields for start and stop time blank (3), and choose the path direction **Road reverse** (4).

Note The fields for start and stop time are left blank to avoid conflicting settings for the same parameters made from the linked Procedures dataset, which will be discussed shortly.

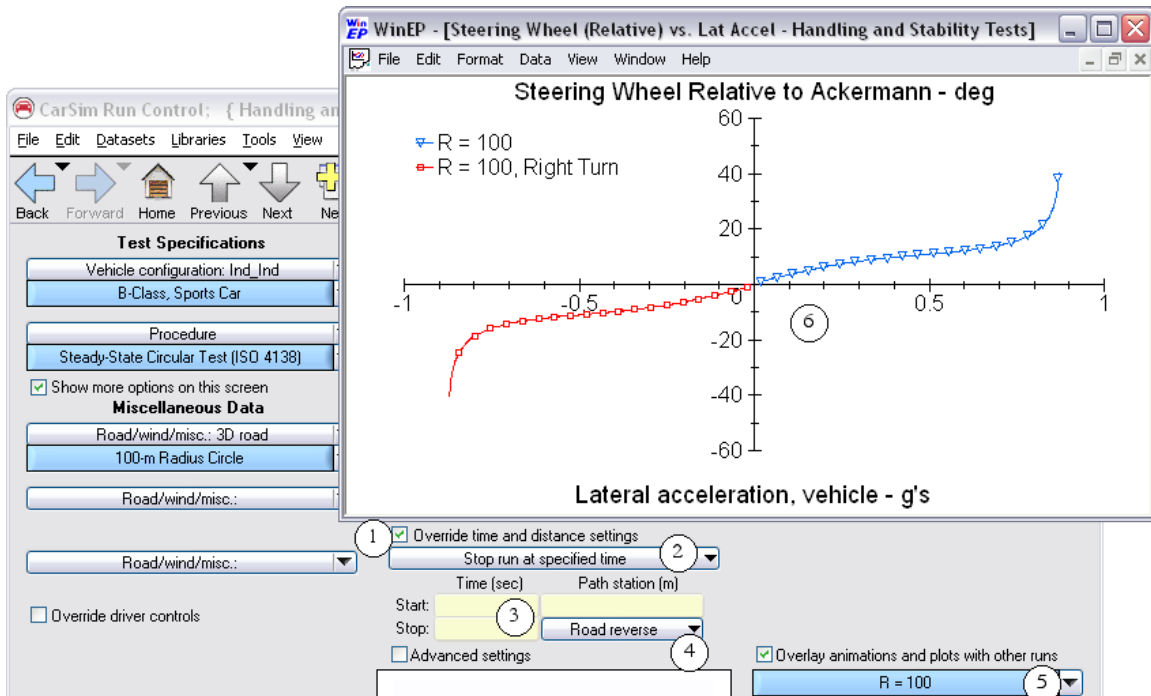


Figure 4. Overlay of two runs showing both steering directions.

In this example, the results of the right turn are overlaid with results for the left turn (5). The plot shows symmetric vehicle behavior (6).

The Circular Track

A circular track is easily defined with the **Road: Segment Builder** screen (Figure 5), which is linked on the **Road: 3D Surface (All Properties)** library (Figure 6). This is in turn linked to the **Run Control** screen (3) (Figure 3).

The **Road: Segment Builder** dataset (Figure 5) makes the provision that the road should be looped (1), and specifies a single turn to the left (2) with an angle of 360° (3) and a radius of 100 m (4). A table will be generated with 360 values (5) to provide good resolution.

The dataset for the entire road environment is located in the **Road: 3D Surface (All Properties)** library (Figure 6). It includes the previously mentioned link to the circular track from the **Road: Segment Builder** (1), a link to a surface friction dataset (2), and some animation data (4).

A checkbox is used to show a yellow field for advanced settings (3); this yellow field is in turn used to enter the VS command `DEFINE_PARAMETER` to add a new parameter R with a value of 100. The parameter R will be used in a formula on another screen that will be described shortly.

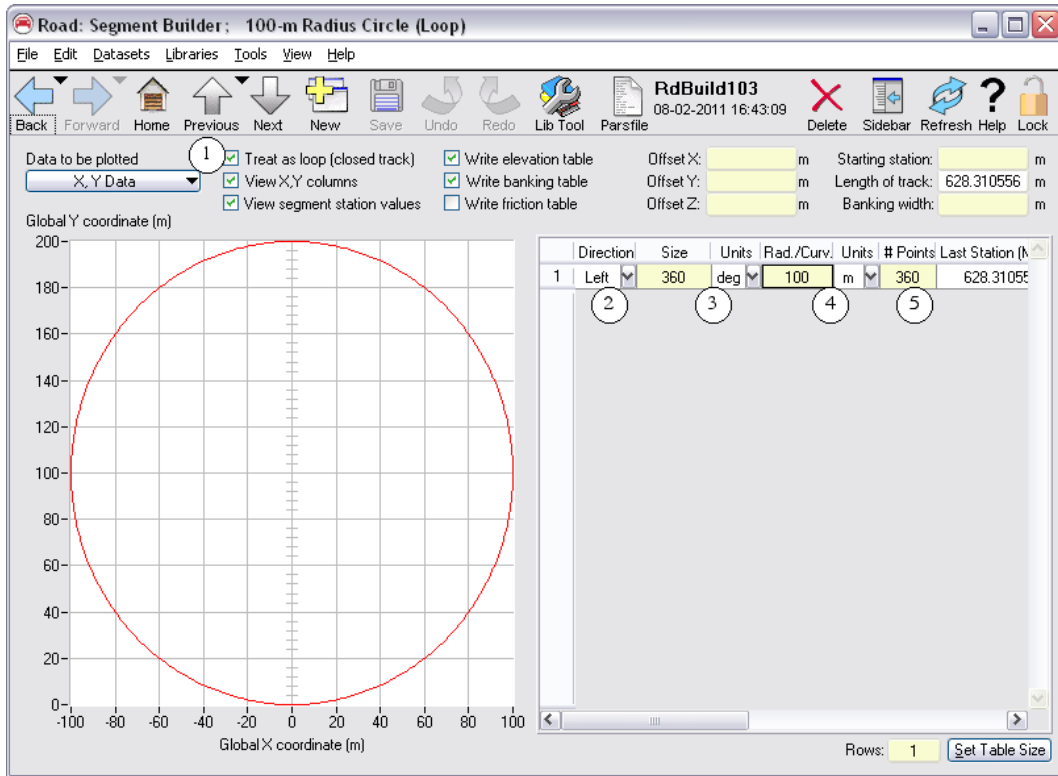


Figure 5. Definition of a circular road for the understeer test.

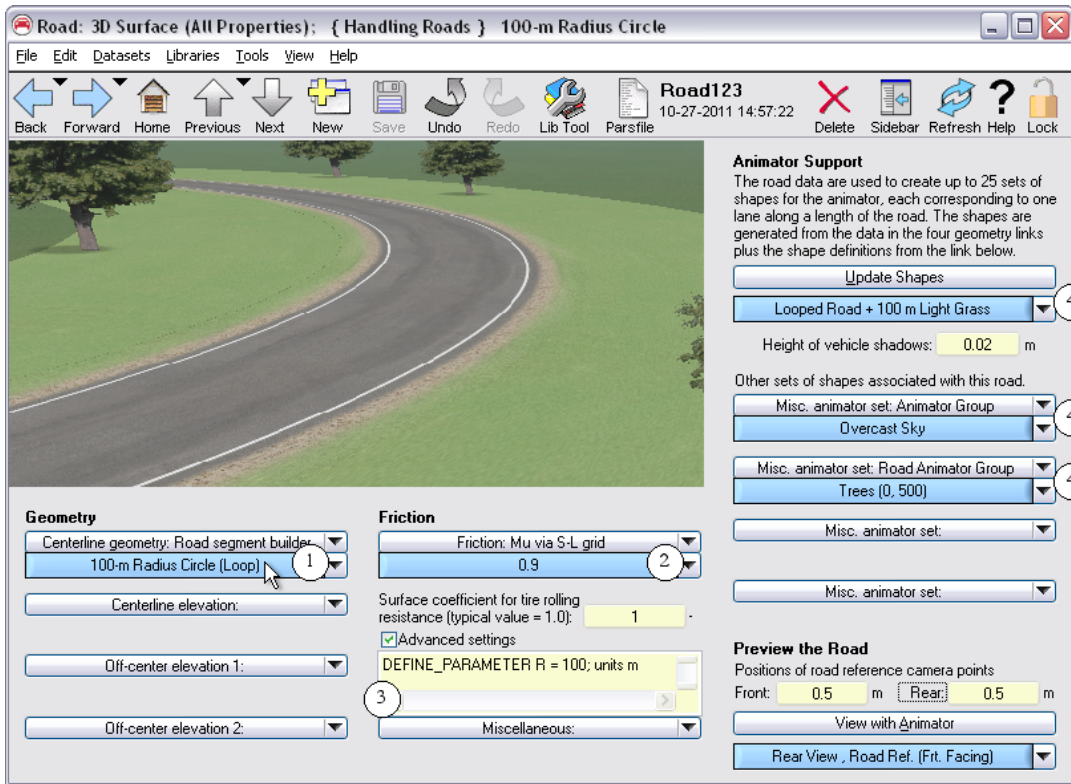


Figure 6. Entire road assembly of datasets and VS command.

Note The parameter R is given a value on the road screen rather than the screen where it is used in order to make the datasets more modular. If R were defined somewhere else and someone changes the link to the **Road** dataset from the **Run Control** screen, the value of R would not match unless (1) the user knows that the parameter is used in the procedure, and (2) the user finds the location of R and changes the value to match the road geometry.

With the definition of R on the **Road** screen, changing the radius for the test is done with a single action: change the **Road** link on the **Run Control** screen.

The Procedure Dataset

Figure 7 shows part of the **Procedures** screen for this run plus the associated notes (1).

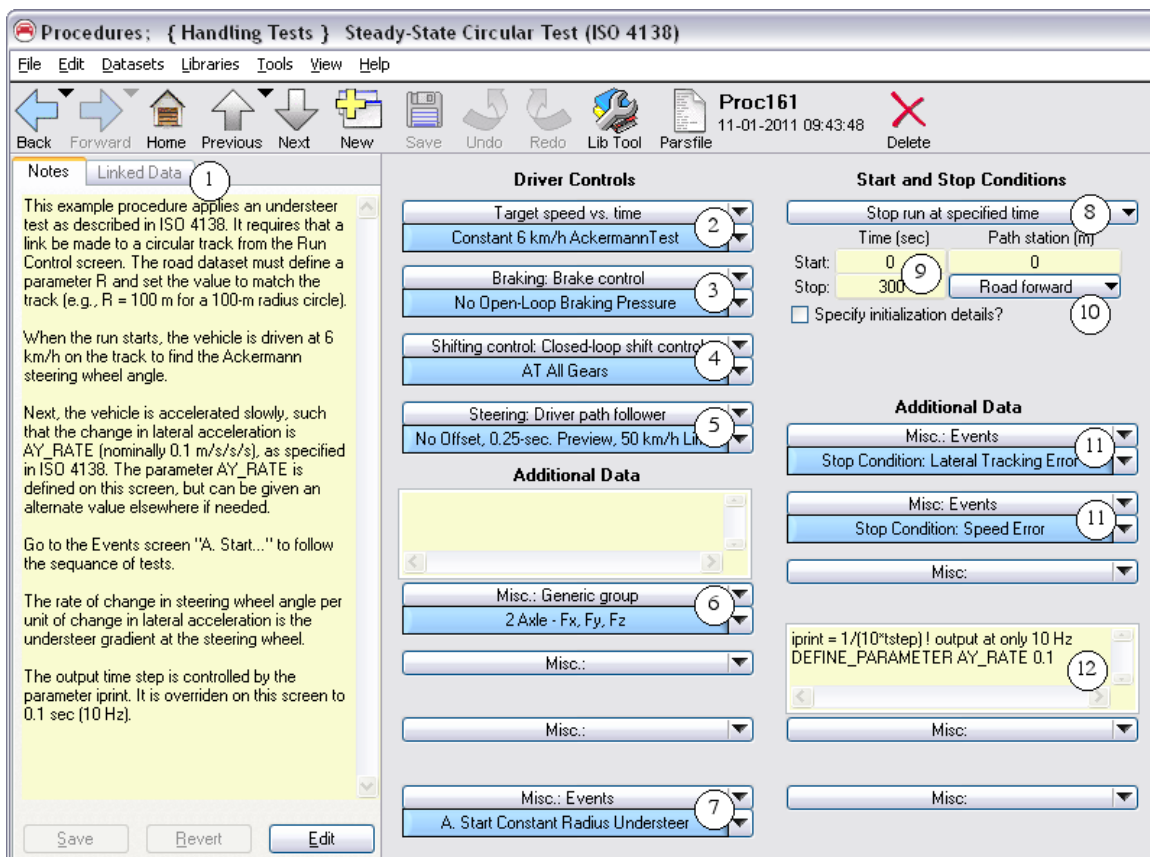


Figure 7. Procedure dataset for an understeer test using ISO 4138.

Going through the settings, we see the kinds of control and environmental settings appropriate for the start of the run:

- The speed control is initially set using the built-in closed-loop controller, with target speed set as a function of time (2). The title of the linked dataset indicates that the

target speed is initially 6 km/h. (Details about the built-in speed controller are provided in the document **Driver Controls**, available from the **Help** menu.)

- No open-loop braking is applied (3).
- Shifting is performed with a closed-loop controller that will use the shift schedules for the vehicle, using all gears available in the vehicle (4).
- Steering is set with a closed-loop path follower that will follow the target path (road centerline) using a 0.25 second preview time (5) and a low-speed limit of 50 km/h (13.9 m/s). This setting is tuned somewhat for the circular track, with a tight tolerance at high speed and a preview distance set for about 3.5 m (13.9*0.25) at speed less than 50 km/h. A different setting is used in TruckSim that is more stable but does not track the target path as closely.

Details about the built-in path follower are provided in the document **Driver Controls**, available from the **Help** menu.

- The example in CarSim has tire force arrows added for animations (6). The TruckSim version does not have these, because there is more variability in size and the number of axles. (They are typically added on the **Run Control** screen, where they can be matched to the type of vehicle being simulated.)
- We expect that the run will be stopped by an event (as described later), but just in case, a setting is made to stop at a specified time (8). The specified time limit (300 sec (9)) is longer than the expected duration of the test.
- The direction on the road is set such that the vehicle goes forward (10). The road was defined such that going forward implies a turn to the left (Figure 5, (2)).
- The run will be stopped if tracking error exceeds a limit, or if the closed-loop speed controller cannot maintain speed within a tolerance relative to the target speed. These conditions are defined with pending events (11). (These and other stopping conditions are described in more detail later.)
- As noted earlier, this test has a few associated parameters. One is the rate of change of lateral acceleration: `AY_RATE`. It is not built into CarSim, but is defined with the VS command `DEFINE_PARAMETER` and given a value of 0.1 (12). This parameter is used together with the path radius `R` in a formula presented later.

Time Step in the Output Files

The print interval for writing to ERD/BIN files is set to about 10 Hz to avoid unnecessarily large files. This is done because the intended plots (Figure 1 - Figure 2) show quasi-static relationships; high-frequency information is not expected. The time step in the output file is a multiple of the simulation time step: $I\text{PRINT} * T\text{STEP}$, where $T\text{STEP}$ is the simulation time step and $I\text{PRINT}$ is a system parameter. The time step value is not specified on this screen. In order for the procedure to be portable — usable with possible external controllers from Simulink or alternate tire models or powertrain models — we cannot assume a value for the time step. Given that we want the output time step to be 1/10, $I\text{PRINT}$ is set to $1 / (10 * T\text{STEP})$ (12, Figure 7).

The Chain of Events for the Understeer Test

The **Procedures** screen (Figure 7) has a link to an **Events** dataset (7) that is used to initiate the running of the understeer test and is described in this subsection. Other links to **Events** datasets (1) set up conditions for stopping the test and are described in the next section.

As with any dataset linked to the **Procedure** dataset, the associated parsfile is read by the VS solver before the run starts. This **Events** dataset (Figure 8) sets up a pending event that will trigger when we think the simulation has run long enough such that the turning can be considered steady state, and we can look at the values of steer angles to use as Ackermann values.

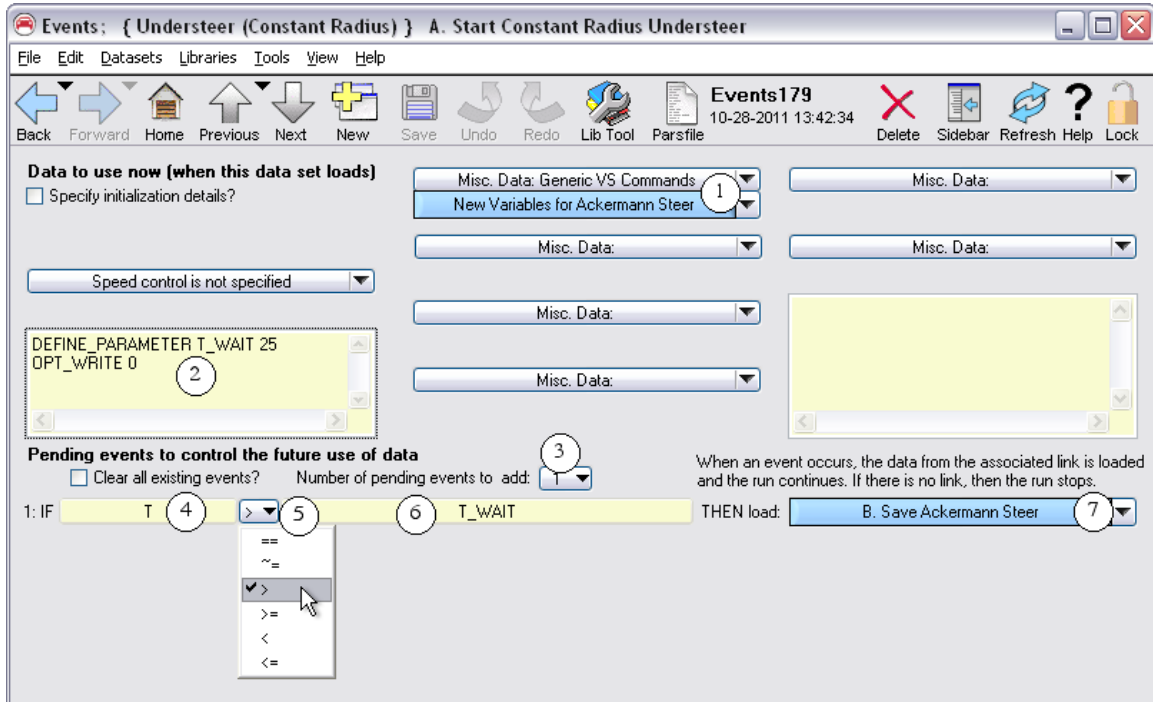


Figure 8. Setting up the test sequence.

A pending event is defined with a *Boolean* expression (a logical expression that must have a value of true or false) of the form:

$$\text{variable operator threshold}$$

where *variable* is any parameter or variable in the math model that can be identified with a keyword (e.g., T (4)), *operator* is a comparison operator available from a pull-down list (5), and *threshold* is any expression that can be evaluated by the VS solver (e.g., T_WAIT (6)). This can be a number, a symbol, or an algebraic expression.

As the simulation runs, all pending events are evaluated each time step. The *threshold* expression is evaluated using current values of any parameters or variables included in the expression, and the result is compared to the current value of *variable*. If the Boolean expression is false, no action is taken. However, if the Boolean expression is true, then the event is said to be *triggered* and one of two possible actions are taken:

1. If another dataset is specified (e.g., **B. Save Ackermann Steer** ⑦), then that dataset is read by the VS solver and processed, and the run continues using any changes in the vehicle properties or test conditions that are specified in the new dataset.
2. On the other hand, if there is no linked dataset associated with the pending event, then the simulation run is immediately terminated.

If there is a linked dataset, the VS solver removes the event that was just triggered from the list of pending events. Any other pending events remain pending.

The parameter `T_WAIT` is not built into CarSim or TruckSim; it is created in the yellow field ② with the command `DEFINE_PARAMETER` and given a value of 25.

If we thought a waiting time of 25 seconds was OK for all vehicles and turn radii, we could simply put the number in the threshold field ⑥ and would not need to define a parameter. However, if we want to change the wait time (e.g., some large trucks take a long time to reach a steady state), it can be changed from another screen using the keyword `T_WAIT`, so long as the change is sent to the VS solver after this parsfile. For example, this could be done using the miscellaneous field from the top-level **Run Control** screen or the second miscellaneous field from the **Procedures** screen (⑫, Figure 7).

The behavior of the vehicle while it is reaching steady state is often not of interest. We obtain simpler plots if writing to file does not start until the vehicle is slowly increasing speed to generate increasing lateral acceleration. Writing during a simulation can be enabled and disabled using the system parameter `OPT_WRITE` ②. We set this parameter to 0 to disable writing to file when the run starts.

Note Sometimes it is helpful to plot all parts of the simulation, such as when working with a new vehicle or doing validation checking. In these cases, the same parameter can be overridden by using the same keyword `OPT_WRITE` to set the value to 1 (e.g., on the **Run Control** screen).

The plots shown earlier involving average front wheel steer and steer relative to Ackermann involve some variables that are not built into CarSim. Any output variables that will be written to file (e.g., for plotting) must be defined before the output files are created when the run starts. Several new outputs are defined for this test, and the definitions are all written in a linked dataset ① (Figure 8) using the library **Generic: VS Commands** shown in Figure 9.

The **Generic VS Commands** screen is nice for specifying VS commands because the yellow field is large ③; however, it can be easier to read and edit using a text editor by clicking the **Parsfile** button ①. The associated text file (in this case, with the name `GVS122.par`) has the information shown on the screen, plus additional information used to manage the database. Notice that the text is color-coded, with VS commands such as `DEFINE_PARAMETER` and `DEFINE_OUTPUT` shown in bold color.

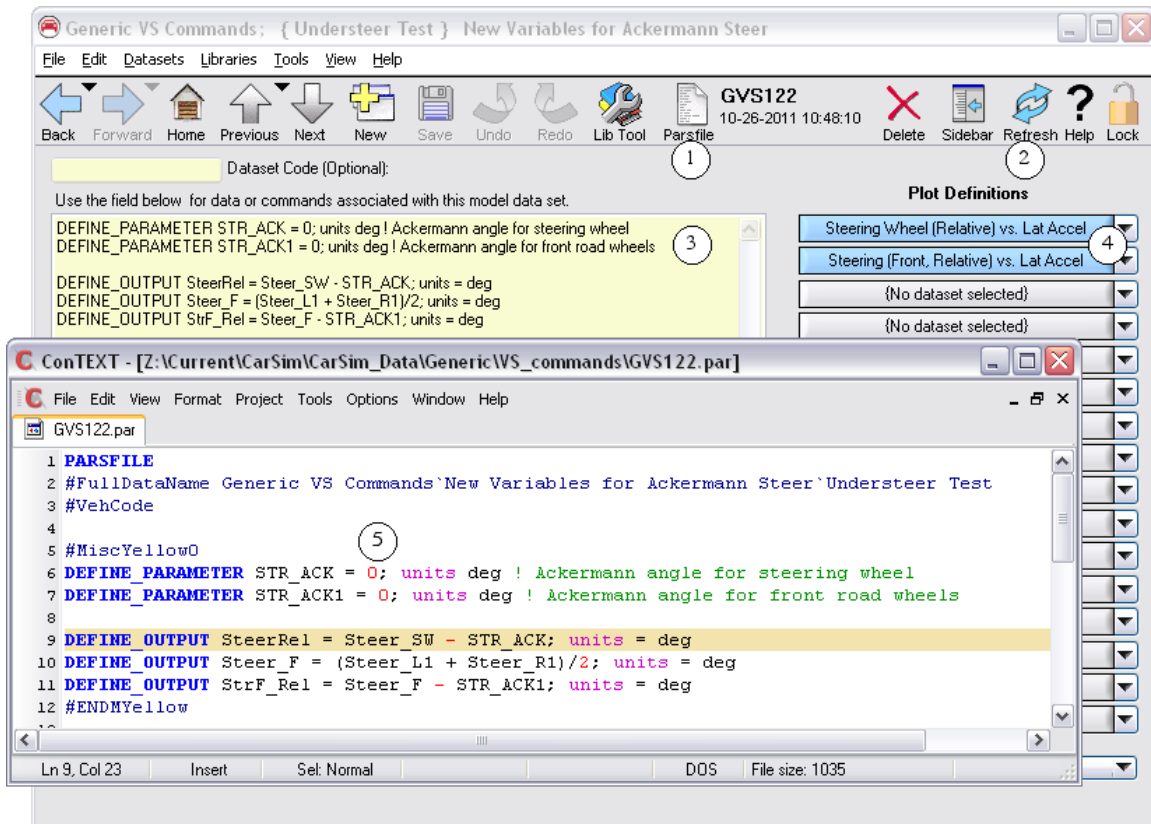


Figure 9. Defining variables with the Generic VS Commands screen.

If you use a text editor, it is important to only edit the text that appears in the big yellow field (3). In the text file, the content of the big yellow field starts after a line that reads #MiscYellow0 (5), and continues until a line #ENDMYellow. Be sure not to edit anything outside the lines that define the beginning and end of the field. Once done editing, save the file; otherwise, CarSim/TruckSim will not use the changes made. To see the changes made by editing the text file directly, click the **Refresh** button (2).

The VS commands shown in Figure 9 define three output variables that appeared in the plots for the understeer test (Figure 1 and Figure 2). In each case, the DEFINE_OUTPUT command provides the variable name, a formula for calculating it, and the units for the new variable. The variables are:

1. Steering wheel angle relative to the Ackermann angle: $SteerRel = Steer_SW - STR_ACK$ (degrees).
2. Average steer of the two wheels of the front axle: $STEER_F = (Steer_L1 + Steer_R1) / 2$ (degrees).
3. Average steer of the two wheels of the front axle relative to the associated Ackermann angle: $StrF_Rel = Steer_F - STR_ACK1$ (degrees).

The equations for the output variables include two parameters that are not built into CarSim: STR_ACK and STR_ACK1. Therefore, they are defined before they are referenced, using the DEFINE_PARAMETER command. Both of the new parameters are given initial values of zero

because the syntax of the command requires a value if units are to be assigned. The values are reset later using results obtained in simulated test.

The plots that use these three new output variables (Figure 2) were generated by **Plot Setup** datasets that are linked from this screen (4). The plot settings are not useful unless VS commands are used to define the new variables with this screen; hence, the plot links are made from the same screen from which the variables are defined.

When the simulation time exceeds T_WAIT seconds, the VS solver will read the dataset file with the title **B. Save Ackermann Steer** (7, Figure 8). Figure 10 shows this dataset, and illustrates important capabilities that are commonly used when loading **Events** datasets after the run is in progress. All of the settings on this screen make use of information that didn't exist before the run started; it is not until the vehicle has settled (when T reaches T_WAIT seconds) that the settings on this screen are useful.

As indicated by the title, one of the purposes of this dataset is to save current steer values by setting the Ackermann parameters STR_ACK and STR_ACK1 (3) to the current values of model variables $Steer_SW$ and $Steer_F$. Thus, the Ackermann angles are the values of steering observed after T_WAIT seconds of running on the circular road at 6 km/h.

This dataset also adds two pending events (5). In both cases, we want to add information when the lateral acceleration reaches some limit. Due to possible transients that occur shortly after the run starts (while the closed-loop path follower is steering the vehicle), variations in lateral acceleration can occur that might trigger events involving lateral acceleration. To avoid this possibility, the events are not added until the vehicle has reached a quasi-static condition.

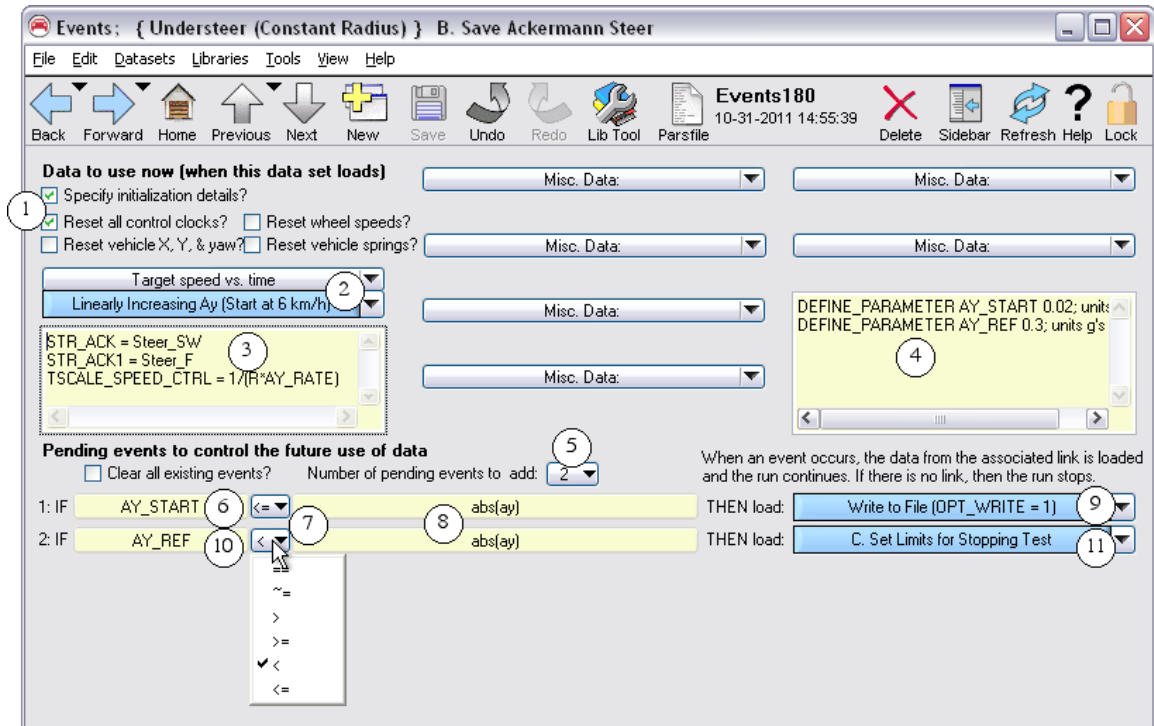


Figure 10. Dataset loaded to save Ackermann steering values and start increasing speed.

Because we want the steady-turning procedure to work for turns in both directions (in a left-hand turn, lateral acceleration is positive; in a right-hand turn it is negative), we use the absolute value of lateral acceleration when adding these two events. To do this effectively, it is helpful to fully understand the definition of a pending event as represented on this screen. Recall that a pending event is defined with a Boolean expression of the form:

$$\text{variable operator threshold}$$

where *variable* must be a parameter or variable in the math model that can be identified with a keyword, and *threshold* is any expression that can be evaluated by the VS solver.

Whenever the Boolean expression would involve a formula with a function such as $\text{abs}(a_y)$, the Boolean must be defined such that the formula is located in the *threshold* field (8).

The formula must be compared each time step to a variable or parameter. If the math model doesn't have such a parameter or variable, then we add one, typically with the VS command `DEFINE_PARAMETER`, as is done in this example (4).

The first pending event is added to activate writing to file (9). Recall that when the run started, writing to the ERD/BIN output file was disabled by setting `OPT_WRITE` to 0 (Figure 8). This event is intended to start writing to file when the vehicle has reached a condition where we want to see cross-plots as shown earlier (Figure 1 and Figure 2). We define the condition to be when the absolute value of lateral acceleration has increased to some threshold `AY_START` (6), given a value of 0.02 g's (4). (The value of the parameter can be reset elsewhere, such as on the miscellaneous yellow field on the **Run Control** screen.) When lateral acceleration has reached the level needed to trigger this event, another dataset is loaded (9) that sets `OPT_WRITE` to 1 so results can be plotted.

A second pending event is added with the intent to stop the simulation when the steering wheel angle changes drastically. To determine what a "drastic" change is, we define some events based on the steering wheel angle when the lateral acceleration reaches the value of a parameter `AY_REF`. This parameter is defined and given a value of 0.3 g (4). When the vehicle absolute lateral acceleration hits this level (7), another dataset is loaded (11) to set limits for stopping the run. (These limits will be described in the section **Stopping the Run**.)

This dataset also resets the speed control for the remainder of the run. A link is made to a dataset for the closed-loop speed controller, specifying target speed as a function of time (2). This dataset (Figure 11) shows target speed starting at 6 km/h and increasing to 160 km/h at almost 2000 seconds.

Settings for the closed-loop speed controller gains are left blank (2). This means that when the VS solver reads the dataset during the run, the target speed is redefined, but the controller settings retain their earlier values set from a speed control dataset linked from the **Procedure** screen (2, Figure 7). This is done in case special values might have been set elsewhere.

The notes for the target speed (1) indicate that the curve was created using the **Calculator: Symbolic** screen, accessible with the **Libraries** menu or the **Tools** menu. This library is handy for generating tables from functions and keeping the formulas in the database for future use and revision. Figure 12 shows the **Calculator: Symbolic** dataset used to generate the table of speed vs. time.

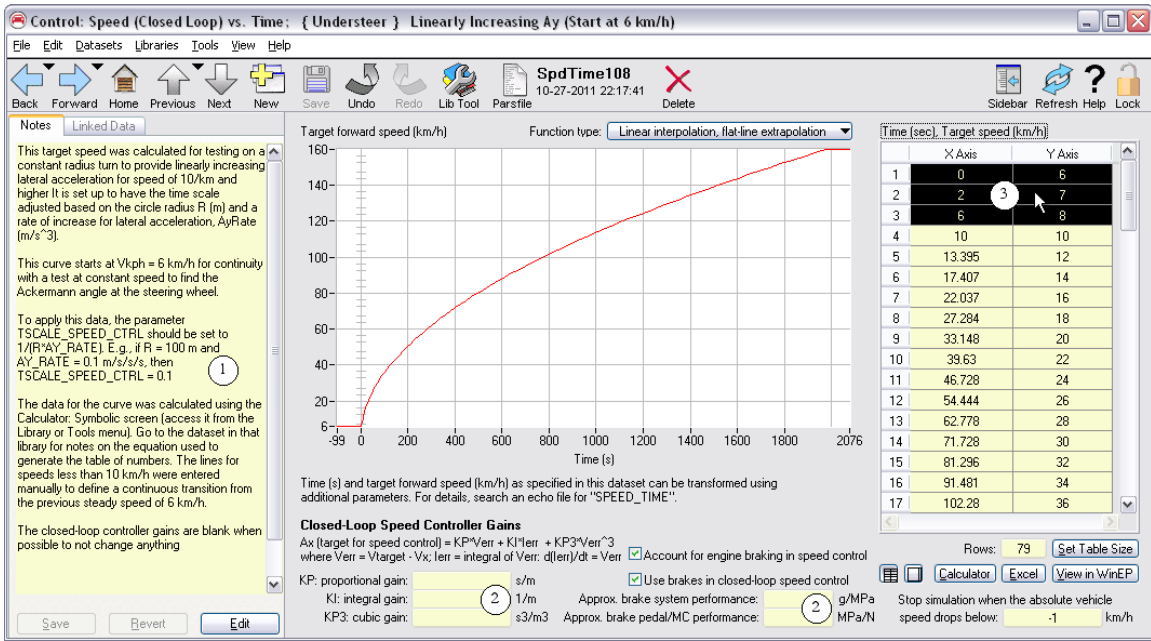


Figure 11. Target speed for linearly increasing Ay.

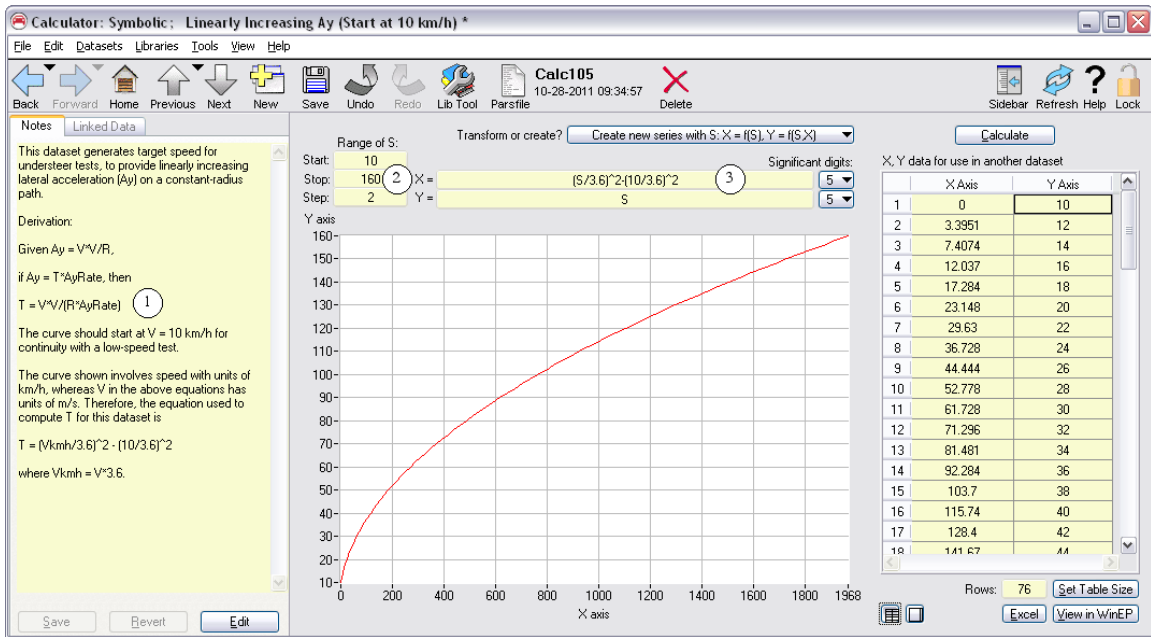


Figure 12. Using the calculator to apply a formula for target speed.

In Figure 12 we can see that the independent variable of the calculator, S, is the target speed in km/h (3), and that it ranges from a starting value of 10 up to a maximum of 160 with increments of 2 (2).

The calculator notes (1) explain the formula used to calculate time as a function of speed (3). This is a normalized curve that is intended to be rescaled by the factor $1 / (R \cdot AY_RATE)$, where R (radius of path) and AY_RATE (rate of increase of lateral acceleration) are parameters that were

defined earlier with the `DEFINE_PARAMETER` command. The scaling formula is applied on the event screen for the parameter `TSCALE_SPEED_CTRL` (3) (Figure 10, page 14).

By using the built-in scale factor for target speed, it is possible to reuse the same normalized curve for different turn radii and different rates of increasing acceleration.

With a scale factor of $1/10$ ($1 / (R * AY_RATE)$) where $R = 100$ m and $AY_RATE = 0.1$ m/s²/s, the time scale for the plot is reduced to about 200 sec. (The example vehicle cannot reach a speed of 160 km/h on a 100-m radius path; therefore the time needed for the test is significantly less than 200 sec).

Additional rows for the table were entered manually for the table in the **Control: Speed (Closed Loop) vs. Time** screen to provide a smooth transition from 6 to 10 km/h (3), Figure 11).

The timeline for the increasing target speed starts at zero (Figure 11). However, our intention is to have the target speed start increasing at the time that the dataset is loaded, when the simulation time has reached `T_WAIT`. To do this, the box on the **Events** screen is checked to **specify initialization details** (1), Figure 10) revealing another box that is checked to **reset all control clocks**. (Other boxes involving resetting of vehicle motion variables are not checked.) Resetting the control clocks shifts the time scale for the target speed to the time when the event is triggered.

Stopping the Run

The **Procedures** dataset specified a stop time of 300 sec as a backup in case other conditions for stopping the run are not met. Overall, four other conditions for stopping the run are set in this example.

Lateral Tracking Error

ISO 4138 specifies that lateral tracking error should be limited to 0.5 m; therefore, the run should stop if this limit is reached. A link was made to an **Events** dataset from the **Procedures** screen (Figure 7) to add a pending event. This **Events** screen is shown in Figure 13.

Note This lateral tracking stop condition can be used for runs other than understeer tests. Any simulations that require the vehicle to follow a path using the closed-loop steer controller might benefit from having this check installed, to stop the run if the lateral tracking error becomes excessive.

Recall that when the Boolean formula used to define an event includes functions or more than one variable, the expression must be placed into the yellow field on the right-hand side of the conditional function. In this case, the expression is `abs(Lat_Veh - Lat_Targ)` (4). The parameter to check (`Lat_Tolerance`) is not built into CarSim or TruckSim, so it is defined here with the `DEFINE_PARAMETER` command (1). As with any other parameter, the value of `Lat_Tolerance` can be changed by any dataset that is read by the VS solver after this one (e.g., it could be set using a miscellaneous yellow field on the **Run Control** screen).

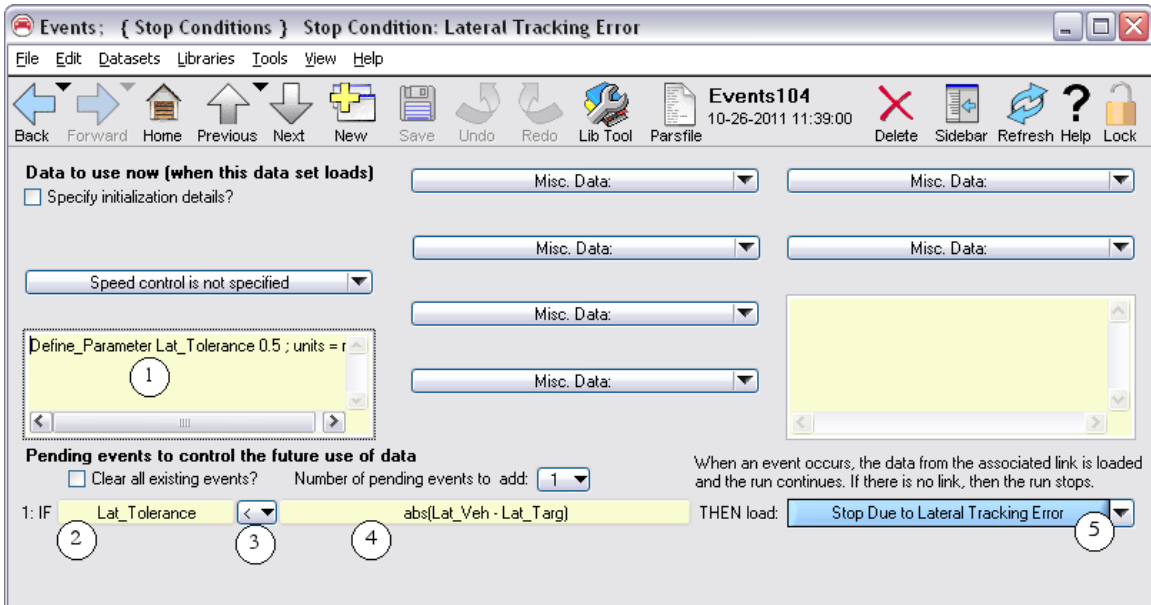


Figure 13. Event for stopping the run if lateral tracking exceeds a limit.

Although a VS solver will terminate the run if an event is triggered that does not include an associated link (e.g., ⑤) most of the examples shipped with CarSim and TruckSim do not use this feature. Instead, they explicitly link to a dataset that terminate the run. This is done to confirm that the termination is intentional and not an error in which a link was accidentally broken.

Figure 14 shows the dataset that stops the run when loaded due to the event for lateral tracking error being triggered. It has a single setting: the VS command `STOP_RUN_NOW` followed by a message that is written into the Log file to document the action (①, Figure 15).

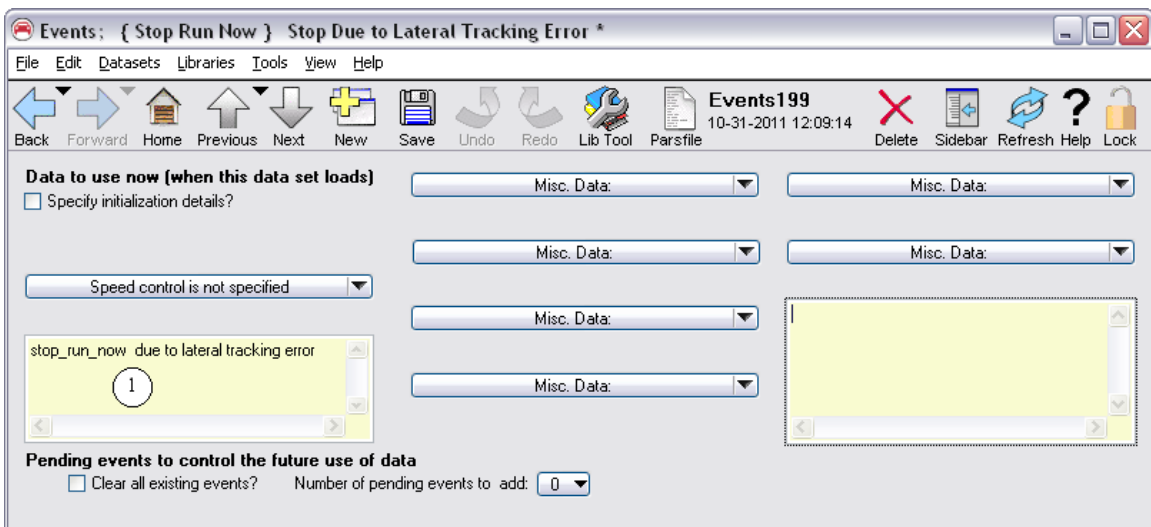


Figure 14. Dataset to terminate a run using the `STOP_RUN_NOW` command.

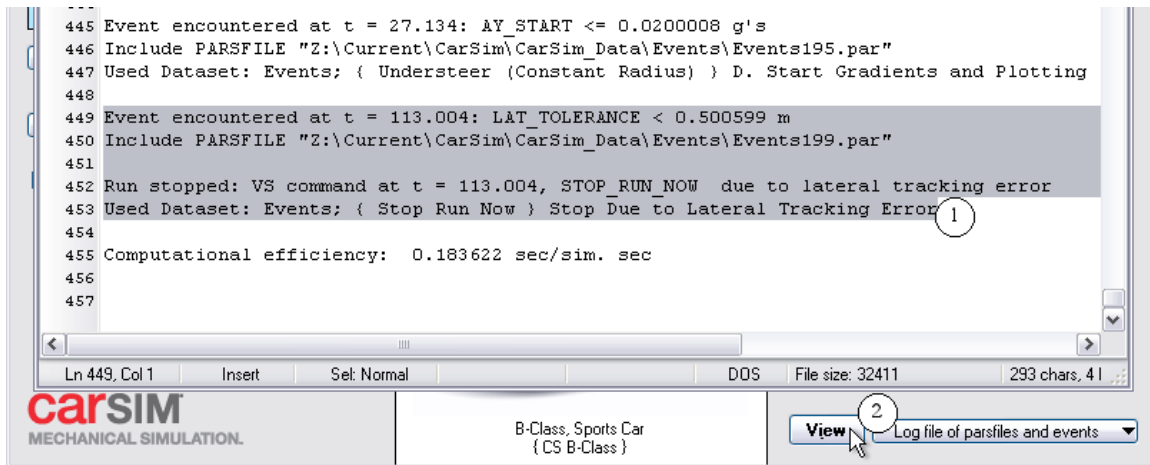


Figure 15. End of Log file, indicating the reason a run was terminated.

Note The Log file can be viewed from the **Run Control** screen by choosing the file type and clicking the **View** button (2).

Speed Error

There are situations where the vehicle reaches a limit in speed. For example, when heavy trucks with high centers of gravity make circular turns with slowly increasing speed, vehicle roll can cause the inside drive wheels (relative to the turn) to lift, reducing the capability of the vehicle to increase speed. To check for this possibility, the **Procedures** dataset has a link to another **Events** dataset that adds a pending event based on the absolute difference between the target speed and the actual vehicle speed: $\text{abs}(Vx_Err)$ (11) (Figure 7). The dataset is similar to the one used for lateral tracking, except the tolerance is named $Vx_Tolerance$ with a value of 5 km/h.

Steering Limits

The limits in tracking error and speed error are sufficient to stop the simulation run when the vehicle reaches the limits allowed for the test in ISO 4138. However, at the very limit, large steering wheel angles are generated by the built-in closed-loop path following. For example, Figure 16 overlays results for the testing shown earlier in Figure 1 with the difference that the simulation runs lasted just a little bit longer until the either the lateral tracking or the speed limit was reached.

In the case of the baseline vehicle (blue line), the steering wheel angle increases to 475° , when the run is stopped due to tracking error exceeding 0.5 m; in the case of the vehicle with different tires, the steering wheel angle goes in the opposite direction to a value of -174° , when the run is stopped due to excessive speed error.

After viewing plots such as those shown in Figure 16, two pending events were scripted to stop the run when the steering wheel angle from the closed-loop controller reaches large values. One event handles the case when steering increases excessively because the vehicle is no longer responsive to steering (too much understeer). Another handles the case where steering decreases excessively because the vehicle is too responsive to steering (too much oversteer).

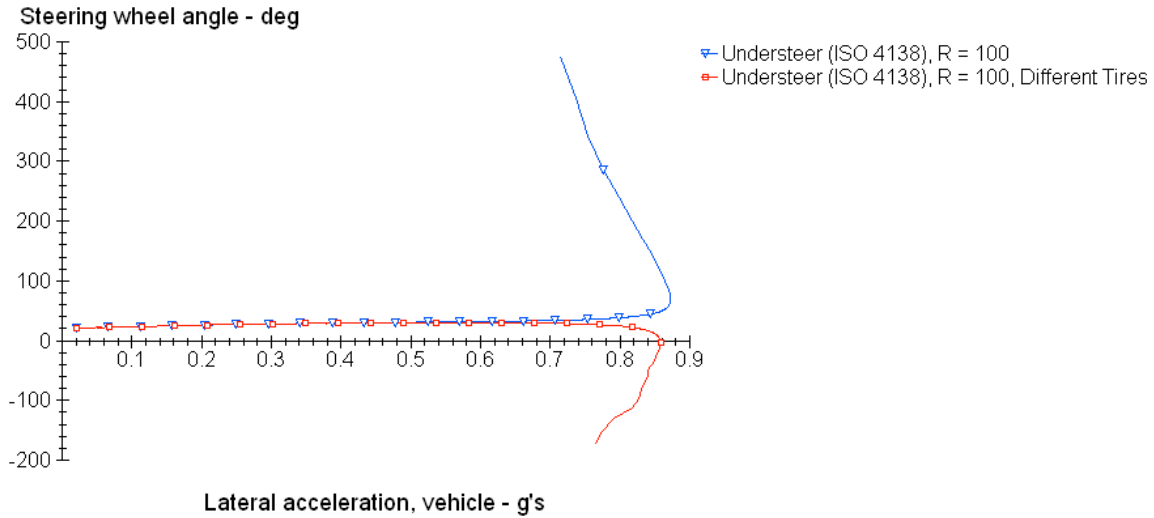


Figure 16. Steering wheel angle going to the frictional limits of the testing.

The amount of steering associated with this test depends on properties of the vehicle. Therefore, the limits are defined using the general sensitivity of the vehicle during the test. When the gradually increasing speed part of the test started, two pending events were added: one to start recording data for plots, and one to set steering limits for stopping (9) and (11), Figure 10, page 14). Sensitivity of the vehicle is determined by looking at steering wheel angles at a reference level of lateral acceleration (AY_REF , given a default value of 0.3 g's). When the absolute value of lateral acceleration reaches that reference level, then the **Events** dataset shown in Figure 17 is loaded.

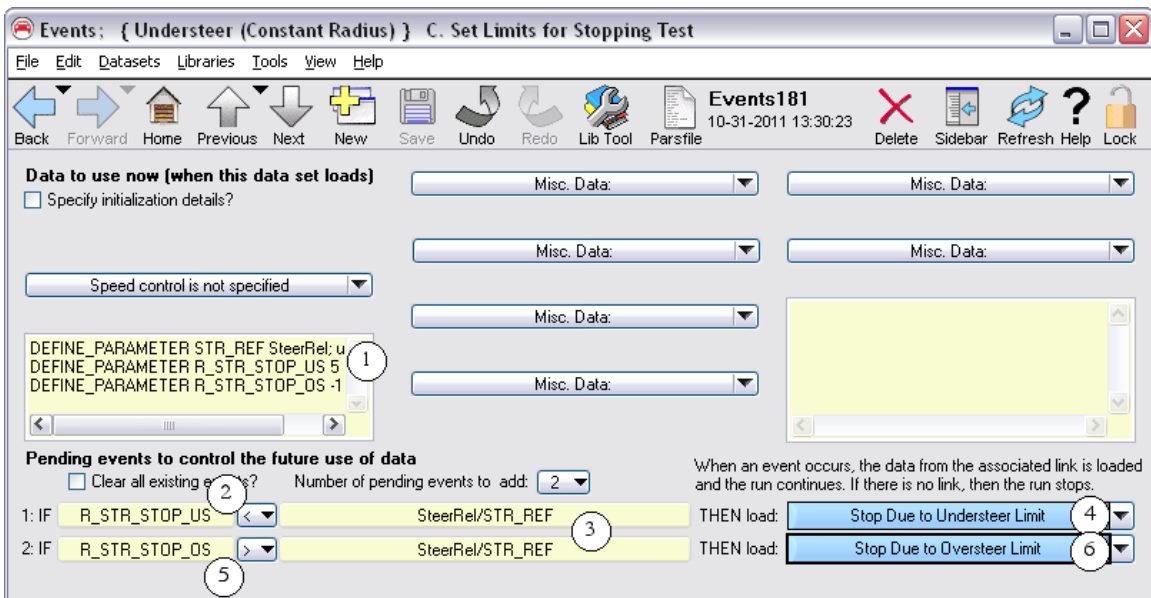


Figure 17. Setting up stopping conditions based on steering wheel angle.

The full steering wheel angle depends on the radius of the path, with shorter radii requiring more steering. Therefore, a reference angle is defined and set to the relative steering wheel angle $SteerRel$ (with Ackermann steer removed) (1), reducing the dependence on the turn radius.

After having looked at several plots such as those shown in Figure 16, a plan was made to stop the test if the relative steering wheel angle exceeds the reference by a factor of 5. For example, the plots of relative steering wheel angle shown in Figure 2 (page 5) shows the relative angle at about 8° with lateral acceleration of 0.3 g's, and the test plot ends with a relative angle of about 40°.

This condition is reached when the following Boolean formula is true:

$$\text{SteerRel}/\text{Str_Ref} > 5 \quad (\text{understeer limit})$$

As was done in other example **Events** datasets, the formula (an algebraic ratio) goes into the threshold field (3) and a new parameter is defined for the limit: R_STR_STOP_US (2) with a default value of 5 (1).

In the case of an oversteer limit, the plan is to stop the test when the relative steering angle has reversed direction and reached the same relative steer as needed to for the 0.3 g's reference.

$$\text{SteerRel}/\text{Str_Ref} < -1 \quad (\text{oversteer limit})$$

In this case, the same ratio goes into the threshold field (3) but a different parameter is used for the limit: R_STR_STOP_OS (5) with a default value of -1 (1). Also, the conditional operator is changed to trigger when the ratio is less than the threshold.

As with the other stopping conditions, datasets are linked that stop the run and write a message indicating the reason (understeer limit (4) or oversteer limit (6)).

The Programming Process

Now that we have reviewed the example procedure, consider the steps in creating a new procedure from scratch.

The discussions and figures in this memo were taken from the final version of the procedure. Not all parts are absolutely necessary. When developing a comprehensive test procedure such as this example, it is typical to work in stages.

Simulate the Basic Test

Start by simulating the most important part of the test, setting controls as needed to see the behavior of interest.

For the understeer example, the most essential part of the procedure is to have a slowly increasing speed while following a constant-radius path. This can be simulated with no events or VS commands with a few basic control options:

1. Define a circular path with the desired radius and use the closed-loop steering controller to follow that path.
2. Use the closed-loop speed controller and define a target speed where the speed slowly increases
3. Turn off open-loop braking and use closed-loop shifting for the powertrain.
4. Set up plots showing the variables of interest (e.g., steering vs. lateral acceleration).

Automate a Series of Tests with Events

Once it has been determined that the behavior of interest can be simulated, the procedure can be extended to match custom requirements.

In this example, the Ackermann steering angle is needed to show relative angle at the steering wheel and the front road wheels. Two new parameters were added to define Ackermann angles for the steering wheel and the average of the front road wheels, and new output variables were added for relative steering wheel angle, average front wheel steering, and average relative front wheel steering. Two tests were defined and linked in sequence using an event based on simply waiting some period of time (`T_WAIT`) for the vehicle and controller to settle into a steady-state turning condition at low speed.

When adding parameters, variables, and pending events via VS commands, we recommend viewing the end of the Echo files where these VS commands are echoed. (View an echo file with the **View** button in the lower-right part of the **Run Control** screen.) Check to make sure they appear as intended. Note that an echo file is generated at the start of each run and another is generated at the termination. For example, Figure 18 shows the end of the echo file made at the start of the run for the 100-m radius turn. It lists new parameters defined before the run started, equations for three new output variables (shown with the command `EQ_OUT`), and three pending events.

Figure 19 shows similar information, written at the end of the run. Comparing the two listings shows that more parameters were defined by the end of the run (these were added with an **Events** dataset that was processed after the simulation had run for `T_WAIT` seconds). The event checking the Boolean condition `T > T_WAIT` (line 4153 in Figure 18) is not shown in Figure 19 because this event was triggered and removed before the run ended.

Refine and Extend the Procedure

After the procedure is working for one specific scenario, you might want to cover similar scenarios. For example, the first version of this steady-turning example (not documented in this memo) had a link to the 100-m radius road dataset from the **Procedures** screen, and had a slowly increasing speed suitable for the 100-m radius. Later, when considering possible tests with other radii, the link was moved from the **Procedures** screen to enable the single **Procedures** datasets and all of the linked **Events** datasets to work with an arbitrary radius. Also, parameters `R` and `AY_RATE` were introduced to make the procedure parametric.

Control of writing to file was added to remove motions during initializations that are not of primary interest.

The time step in the file was specifically set to 0.1 sec (10 Hz) to reduce the size of output files.

The events used to stop the testing when steering reached high levels were added to improve the initial appearances of the cross-plots and avoid the need to manually edit the plots.

```

4125 ! -----
4126 ! NEW VARIABLES DEFINED AT RUN TIME
4127 ! -----
4128 DEFINE_PARAMETER T_WAIT = 25;
4129 DEFINE_PARAMETER STR_ACK = 0; UNITS = deg;
4130 DEFINE_PARAMETER STR_ACK1 = 0; UNITS = deg;
4131 DEFINE_PARAMETER LAT_TOLERANCE = 0.5; UNITS = m;
4132 DEFINE_PARAMETER VX_TOLERANCE = 5; UNITS = km/h;
4133 DEFINE_PARAMETER AY_RATE = 0.1;
4134 DEFINE_PARAMETER R = 100; UNITS = m;
4135 DEFINE_OUTPUT SteerRel = 0; UNITS = deg;
4136 DEFINE_OUTPUT Steer_F = 0; UNITS = deg;
4137 DEFINE_OUTPUT StrF_Rel = 0; UNITS = deg;
4138
4139 ! -----
4140 ! EQUATIONS OUT (AT END OF EACH TIME STEP)
4141 ! -----
4142 EQ_OUT STEERREL = STEER_SW -STR_ACK ;
4143 EQ_OUT STEER_F = (STEER_L1 + STEER_R1)/2 ;
4144 EQ_OUT STRF_REL = STEER_F -STR_ACK1 ;
4145
4146 ! -----
4147 ! EVENTS
4148 ! -----
4149 ! Each event is defined with the name of a variable, a comparison operator (==,
4150 ! ~=, <=, >=, <, or >), a comparison expression, and an optional pathname for a
4151 ! parsfile to read if the specified condition occurs. If no pathname is specified
4152 ! and the specified condition occurs, then the run stops.
4153 DEFINE_EVENT T > T_WAIT ; Events\Events180.par
4154 DEFINE_EVENT LAT_TOLERANCE < ABS(LAT_VEH-LAT_TARG) ; Events\Events199.par
4155 DEFINE_EVENT VX_TOLERANCE < ABS(VX_ERR) ; Events\Events200.par
4156
4157
4158 END

```

Figure 18. Echo file generated at the start of a run.

Review of the Methods Used

This procedure illustrated several methods that can be used routinely to define procedures based on standard tests. Although the example was for a specific handling test, the basic programming approach is recommended for many other applications.

Programming Approach

The example showed the following concepts:

1. Control of the procedure is handled using built-in options for open-loop control and closed-loop control. In this case, steering is handled with the closed-loop path preview controller that follows the road centerline; speed is handled with the closed-loop controller following a target speed that is adjusted at different parts of the procedure

```

ConTEXT - [Z:\Current\CarSim\CarSim_Data\Runs\Run267_END.PAR *]
File Edit View Format Project Tools Options Window Help
Run267_END.PAR *
4211 ! -----
4212 ! NEW VARIABLES DEFINED AT RUN TIME
4213 ! -----
4214 DEFINE_PARAMETER T_WAIT = 25;
4215 DEFINE_PARAMETER STR_ACK = 19.64; UNITS = deg;
4216 DEFINE_PARAMETER STR_ACK1 = 1.33597; UNITS = deg;
4217 DEFINE_PARAMETER LAT_TOLERANCE = 0.5; UNITS = m;
4218 DEFINE_PARAMETER VX_TOLERANCE = 5; UNITS = km/h;
4219 DEFINE_PARAMETER AY_RATE = 0.1;
4220 DEFINE_PARAMETER R = 100; UNITS = m;
4221 DEFINE_PARAMETER AY_START = 0.02; UNITS = g's;
4222 DEFINE_PARAMETER AY_REF = 0.3; UNITS = g's;
4223 DEFINE_PARAMETER STR_REF = 8.37748; UNITS = deg;
4224 DEFINE_PARAMETER R_STR_STOP_US = 5;
4225 DEFINE_PARAMETER R_STR_STOP_OS = -1;
4226 DEFINE_OUTPUT SteerRel = 41.8965; UNITS = deg;
4227 DEFINE_OUTPUT Steer_F = 3.98453; UNITS = deg;
4228 DEFINE_OUTPUT StrF_Rel = 2.64856; UNITS = deg;
4229
4230 ! -----
4231 ! EQUATIONS OUT (AT END OF EACH TIME STEP)
4232 ! -----
4233 EQ_OUT STEERREL = STEER_SW -STR_ACK ;
4234 EQ_OUT STEER_F = (STEER_L1 + STEER_R1)/2 ;
4235 EQ_OUT STRF_REL = STEER_F -STR_ACK1 ;
4236
4237 ! -----
4238 ! EVENTS
4239 ! -----
4240 ! Each event is defined with the name of a variable, a comparison operator (==,
4241 ! ~=, <=, >=, <, or >), a comparison expression, and an optional pathname for a
4242 ! parsfile to read if the specified condition occurs. If no pathname is specified
4243 ! and the specified condition occurs, then the run stops.
4244 DEFINE_EVENT LAT_TOLERANCE < ABS(LAT_VEH-LAT_TARG) ; Events\Events199.par
4245 DEFINE_EVENT VX_TOLERANCE < ABS(VX_ERR) ; Events\Events200.par
4246 DEFINE_EVENT R_STR_STOP_OS > STEERREL/STR_REF ; Events\Events201.par
4247
Ln 4225, Col 37      Insert      Set: Normal      Modified      DOS      File size: 242613

```

Figure 19. Data file generated at the end of a run.

2. Going from one stage to another is controlled with events, triggered when a Boolean expression comparing a variable to a formula becomes true. In this example, the first part of the run involves a low-speed turn on the circular track that goes for about 25 seconds; the remainder of the run has the speed increasing until the run terminates.
3. Multiple pending events can be defined at different times for different reasons, and may remain in effect until either they are triggered or the run ends. In this example, four events are defined to end the run. Of those four, the first one triggered has the effect of stopping the run.
4. Configurable functions can be created to match the requirements of specific tests. In this case, target speed is defined as a function of time such that lateral acceleration increases at a constant rate on the circular track.
5. New parameters are defined as needed to simplify the procedure or make it more configurable. In this example, parameters were defined for the turn radius (R), rate of increase of lateral acceleration (AY_RATE), reference lateral acceleration (AY_REF),

reference steering at the reference lateral acceleration (STR_REF), and several tolerances (LAT_TOLERANCE, VX_TOLDERANCE, etc.).

6. New parameters and output variables are defined as needed to view variables of interest. In this example, parameters were defined for Ackermann steering wheel angle (STR_ACK) and Ackermann angle for the average steer at the front axle (STR_ACK1). New output variables were defined based in part on those parameters.
7. Writing to the ERD/BIN output file was controlled via the parameter OPT_WRITE. It was initially set to 0 (don't write) and then set to 1 when the vehicle speed reached 10 km/h.
8. The frequency of writing to the output ERD/BIN file was set with a formula to obtain a specific frequency (e.g., 10 Hz), regardless of the time step used in the internal simulation.
9. When possible, datasets are defined such that they can be applied in different contexts. In this example, the definitions were valid for any turn radius (characterized by the new parameter R and in either direction (steer can be positive or negative).

Use of Support Tools in the Browser

The VS Browser (e.g., carsim.exe) has tools to help prepare inputs for the vehicle math models run by the VS solvers. This example showed how a few of these tools are used.

1. The road-builder tool is used to create custom tracks for specialized tests, such as the constant-radius track used here.
2. Sets of VS commands are often placed in datasets in the **Generic VS Commands** library, which has a large yellow field that can display longer commands.
3. VS commands can sometimes be viewed and edited more easily using a text editor, accessed by clicking the **Parsfile** button in the button bar of the screen.
4. Configurable functions can be made with the Calculator Tool (use the **Calculator** button in the lower-right corner of a screen with tabular data), or with the **Calculator: Symbolic** screen. If the **Calculator: Symbolic** screen is used, as in the example described in this memo, then the formula can be kept in the database for later use and easy modification.