

VS Math Models

Reference Manual



Table of Contents

1. Introduction.....	3
Building and Running VS Math Models	3
Who Should Read this Manual.....	4
Notational Conventions	4
Revision History.....	4
2. Overview of Program Operation.....	7
Parameters, State Variables, and Degrees of Freedom.....	7
Modules and Commands to Construct the Model	8
Running Simulations with VS Math Models	10
Parsfiles (Parameter Files).....	12
Echo and End Files.....	16
Machine-Generated Documentation for Model Variables	22
Simulation Output Files.....	28
Log Files.....	30
Simfile (Simulation Control File).....	30
3. Configuring the Math Model	34
Reading a line from a Parsfile	34
Unit Systems: User Display Units and Internal SI Units	36
Indexed Parameters	38
Configurable Functions (Tables).....	41
Initial Conditions of State Variables	54
4. Extending the Math Models.....	55
Communicating with Import and Export Arrays.....	56
Import Variables.....	57
Export and Output Variables	59
Multiple Ports.....	61
5. Time Steps During Simulation.....	62
VS Numerical Integration Methods	62
Other State Variables.....	63
Simulation Variables for Time	64
Timing for Import and Export Variables.....	66
6. The Simulation Process	69
Initialize.....	69
Run	70
Terminate	72
7. More Information.....	73

NOTICE

This manual describes software that is furnished under a license agreement and may be used or copied only in accordance with the terms of such agreement. BikeSim, CarSim, SuspensionSim, TruckSim, and VehicleSim are registered trademarks of Mechanical Simulation Corporation.

© 1996 – 2022, Mechanical Simulation Corporation.

Last updated May 2022.

1. Introduction

Mechanical Simulation Corporation produces and distributes software tools for simulating and analyzing the dynamic behavior of motor vehicles in response to inputs from steering, braking, throttle, road, and aerodynamics. The simulation packages are organized into families of products named BikeSim[®], CarSim[®], and TruckSim[®]. Another product, SuspensionSim[®], simulates quasi-static equilibrium conditions for suspensions and other multibody systems. All are based on the simulation architecture named VehicleSim[®].

These products contain programs that construct optimized math models for vehicles or other mechanical systems, solve the equations of motion for the model, and generate output variables of interest. VehicleSim (VS) Solvers are dynamic libraries with functions that are used to construct VS Math Models, which then run a simulation. VS Math Models calculate variables that are written into output files for viewing with plots and animation, or exported to other software for additional analysis. Results from simulated tests are typically analyzed using the same methods as would be used for physical test measurements.

Besides the built-in features of a VS Math Model, advanced users can extend the model using several methods. When extending the model, it is helpful to understand how VS Math Models work, in terms of input files, output files, and calculation methods.

Building and Running VS Math Models

VehicleSim products are available for Windows, Linux, and several real-time (RT) platforms. The Windows versions include 32-bit and 64-bit versions of the VS Solver libraries for the product and a database with example vehicles, procedures, and simulations. The main program (e.g., `carsim.exe`) is a database manager with a GUI for setting up the simulations, running the simulations, and viewing results with a video and plotting post-processing tool named VS Visualizer. This main program is called the VS Browser.

As an alternative, command-line wrapper programs (`VS_SolverWrapper_CLI_32.exe` and `VS_SolverWrapper_CLI_64.exe` on Windows, and `VS_SolverWrapper_CLI.bin` on Linux) are available to load a VS Solver library and run a simulation. (Documentation for the command-line wrapper program is in the Help technical memo *VS Solver Wrapper*.)

The RT versions usually have software installed on a Windows host machine that contains the normal Windows installation (database, VS Browser, VS Visualizer, VS Solver libraries for Windows, and other utilities). They also include a VS Solver library to construct VS Math Models for the target RT machine.

The non-RT Linux products include the VS Solver library as a binary library file (`.so`), the database, VS Visualizer, and a command-line wrapper (e.g., for CarSim it is `carsim-cli`). However, there is not yet a Linux VS Browser.

Who Should Read this Manual

This manual describes the fundamental operation of VS Math Models that are constructed in a VehicleSim product: the types of files they read and write, the format of the input files, and how variables in the VS Math Models are shared with Simulink and some other external software.

This manual is mainly intended for those who wish to construct VS Math Models to work with other software or use capabilities beyond those that are obvious from the GUI. Given that the Linux versions do not include a native VS Browser, this document should also be read by Linux users. (It should also be read by users of the Windows command-line based VS solver wrapper programs.)

Occasional users of a VehicleSim product who set up all runs from the Windows GUI might not need the information in this manual.

This manual assumes some familiarity with a VehicleSim product. If you are new to the software, you should first complete the Quick Start Guide for your product. We also recommend that you read the beginning of the **Help** menu document **Reference Manuals > VS Browser (GUI and Database)** to learn how the database is managed and how the GUI works.

Notational Conventions

Dynamic library is a binary library file that is dynamically linked. On Windows OS, dynamically linked library files are called DLLs and often have the `.dll` extension. On Linux, these are typically called *shared objects* and are identified with the `.so` extension.

VS Solver is a dynamic library within a VehicleSim product (such as CarSim) that has VS Modularity functions to construct an internal VS Math Model that can run a simulation.

VS Math Model is a set of variables and functions constructed for a specific vehicle configuration or other system of interest, along with equations to calculate the variables. It is used to run a simulation calculating values for variables of interest and writing results into files, and possibly sharing results with other software during the simulation.

VS Browser is the main Windows program of the product (e.g., `carsim.exe`). It provides the GUI to manage the database, use VS Modularity functions to construct and run VS Math Models, and view results with VS Visualizer.

When showing the syntax for data, words shown in the `Courier` font indicate specific literal keywords or file names. Words in *italics* indicate the name of the item whose value can vary in different situations or indicate the introduction of term with a specific meaning in this document, such as *VS Math Model*. Square brackets [like this] are used in documenting lines of input to indicate that the content between the brackets is optional. An asterisk before braces `*[]` indicates that whatever is between the braces may be repeated.

Revision History

This manual applies to the VS Math Models used in CarSim, TruckSim, BikeSim, and SuspensionSim. The version of the manual is indicated by the date shown at the bottom of page ii.

The manual is updated when new features are added to the VehicleSim architecture, when errors are found in the current version, or when improved descriptions have been written.

- The May 2022 revision added information about how Import variables might interact with native values and values set with VS Command equations. Hyperlinks to local PDF files were removed to improve security.
- The October 2021 revision added more information about indexed parameters and how they are handled in VS Browsers and in symbolic formulas.
- The June 2021 revision added the variables `T_Real_Last`, `T_Real_Step`, and `T_Real_Elapsed`, and had some minor edits.
- The August 2020 revision changed the title of this manual from “VS Solver Programs” to “VS Math Models.” It clarifies the role of the VS Solver library in constructing an internal VS Math Model via `MODEL_LAYOUT` and other commands. It changed the terminology for VS Extended Assignment commands, such as those used to set Configurable Function properties. It clarified the use of index lists in setting properties of indexed parameters. A section on Multiple Ports was added to Chapter 4.
- The April 2020 revision added the `MODEL_LAYOUT` command and revised the discussion of the Simfile. References were included for running multiple vehicles from a single solver. The section on indexed parameters was revised.
- The November 2019 revision had minor edits involving the VS Solver Wrapper and CSV files.
- The April 2019 revision provided more information about documentation files for state variables, import variables, and output variables. It also mentions the User ID option available for some Configurable Functions.
- The October 2018 revision updated information about Echo files.
- The May 2018 revision describes new timing options involving import and export variables to match improvements made in version 2018.1 (Chapters 4 and 5), adds information about Configurable Functions (Chapter 3), and updates the descriptions of calculation sequences (Chapter 6). It also changes recommendations for numerical integration options (Chapter 5).
- The November 2017 revision removed details about multiple custom solvers that are obsolete after the introduction of modular solvers in version 2018.0.
- The October 2016 revision updated the information for the 2017 release involving Simfiles (with macros), the Results folder, the Echo file organization, support for writing CSV spreadsheet files, and some clarifications when running on Linux. The document was reorganized to better cover the additional information.
- The September 2015 revision updated the information in chapter 5 regarding time steps and added information about the three-dot continuation and options with second ‘;’ delimiter used for reading statements from a Parsfile.

- The February 2015 revision provides more information about the Simfile and the specification of the VS Solver pathname.
- The September 2014 revision was updated to describe new features added to the VehicleSim architecture with CarSim 9, such as generation of documentation files, the VS output file format, and organization of Echo files.
- The June 2013 revision expanded the manual with formatting changes and more information about timing, state variables, and the simulation process.
- The January 2013 revision added information about conversion between the user unit system and the internal SI unit system.
- The December 2012 revision was reorganized to provide more information about options for extending the VS Math Model built into a VS Solver library. It added information about setting units for parameters, variables, and tables, documented the `IMPORT`, `EXPORT`, `WRITE`, and `ANIMATE` commands for activating import and output variables, and documented options for splitting 2D tables into two 1D tables.
- The July 2011 revision added more detail about input files and Configurable Functions, including the `_EQUATION` option and parameters for transforming the independent variables. It also added detail about state variables, degrees of freedom, and the machine-generated model description file.
- The June 2010 revision described two new options added to Configurable Functions and added information about extra state variables and the saving of some variable values between time steps during a run.
- The May 2009 revision replaced a single manual with two: *VS Math Models* (this document) and *VS Commands*. This version also introduces new table interpolation options to include variable-width tables, 2D splines, and step tables.
- The May 2008 revision introduced the VS Command scripting language that is now described in the *VS Commands* manual.
- The January 2008 revision added more detail and improved some descriptions of existing features.
- The April 2007 revision described a few more functions that had been added to VS Math Models.
- The first version (February 2007) was prepared for the release of CarSim 7.0.

2. Overview of Program Operation

The purpose of a VS Math Model is to calculate time histories of variables in a simulated test of a vehicle or some other system of interest. The variables of interest include motions (coordinates, angles, speeds, accelerations), forces and moments, controls, and many other types of information.

Parameters, State Variables, and Degrees of Freedom

A VS Math Model is *parametric*; it has built-in equations that perform calculations using parameters and tables as placeholders in those equations. Values are provided to the VS Math Model at runtime for hundreds of parameters. Tables of numbers are provided to define potentially nonlinear relationships between variables in the VS Math Model. The data provided at runtime cause the VS Math Model to simulate a specific vehicle under specific test conditions.

A VS Math Model typically applies tens of thousands of equations to calculate thousands of outputs that might be of interest. In the context of VehicleSim software, specific terms are used for groups of the variables that are updated in a run.

- Variables that are set at runtime but typically remain constant during a run are called *parameters*. Parameters such as dimensions, masses, friction coefficients, etc. are used to define a specific vehicle and test condition.
- Variables that are marked as “read only” are listed in the Echo files and are available for use in VS Command formulas, but they cannot be assigned values directly.
- Fixed relationships between model variables that are potentially nonlinear are specified at runtime with *Configurable Functions* that can be configured to use constants, linear coefficients, or values interpolated from tables of numbers.
- Variables that are needed to define the state of a VS Math Model and which normally change during the simulation run are called *state variables*. For example, a CarSim vehicle model with two independent suspensions contains about 240 state variables. The initial values of the state variables are set at runtime and updated by the VS Math Model as the run proceeds.

Each state variable is necessary to fully define the state of the system. These variables all have names that begin with the prefix “SV_” and are listed in an Echo file made at the end of each run. They are also listed in machine-generated files, as described later in this chapter.

Many of the state variables (SVs) are defined within the VS Math Model with ordinary differential equations (ODEs). The term *equation of motion* is sometimes used to describe an ODE in a multibody model. For example, a basic CarSim vehicle model has about 80 ODEs. The ODE SVs are then calculated from their derivatives using numerical integration methods as described in the technical memo *Numerical Integration Methods in VS Math Models*, available from the **Help** menu. The other SVs (those that are not defined with differential equations) are needed to define some condition of interest in the model, such as whether a clutch is locked or slipping, or the current value of a force that is affected by friction. These are variables that are saved for use in the next time step of the simulation, or to set proper initial conditions for a new run, or to support advanced features in a VS Math Model such as linearization.

All SVs are listed and described in a machine-generated text file, as shown later (Figure 9, page 24). Whether listed in an Echo file or machine-generated text file, the descriptions indicate whether the SV is associated with an ODE. There are a few cases where the old value of an ODE SV is needed, usually to apply friction. In these cases, the non-ODE version typically has a suffix `_OLD` in the name. The `_OLD` suffix is not used for other non-ODE variables, because they are all “old” (calculated at the previous time step).

The term *degree of freedom* (DOF) is sometimes used to describe the complexity of a math model. However, different definitions of DOF are used in different fields of interest (mechanics, dynamics, statistical analyses, etc.).

When describing a math model of a 3D mechanical system composed of rigid bodies, the number of DOF is typically the number of independent generalized coordinates needed to define the state of the system. For example, a CarSim vehicle model without a powertrain or power steering has 14 explicit rigid-body mechanical DOF: 6 for the sprung mass, 2 for each suspension, and 4 for the wheel spin rotations. A rigid-body mechanical DOF usually has two differential equations with corresponding state variables; hence, 28 of the CarSim state variables handle the rigid-body DOFs.

When describing a generic dynamic model that is not limited to rigid bodies, the term DOF is typically applied for each SV. A basic CarSim model has about 80 ODE SVs and a total of about 250 SVs

Some analysts use the term DOF to mean the number of variables that can be adjusted in a math model; by this standard, a CarSim model has thousands of DOF.

Modules and Commands to Construct the Model

Some VS Solver libraries have a single built-in VS Math Model. The VS Math Model used to generate shapes for road surfaces is limited to reading data to describe a road surface (on Windows this solver is named `road_32.dll`). The VS Math Model used to provide a tire tester is limited to tire model options (on Windows this is `tire_32.dll`).

On the other hand, the VS Math Models used to simulate vehicle behavior are constructed with VS Solver libraries that contain many modules. In these cases, the internal database for model variables and keywords contains only a few system-level variables that exist in all VS Commands, such as a start time, stop time, etc. The VS Math Model itself is constructed by processing specific commands that make use of modular options in the VS Solver libraries provided with the product.

The `MODEL_LAYOUT` Command

CarSim, TruckSim, and BikeSim VS Math Models support a command `MODEL_LAYOUT`. This command provides a simple description of the vehicle model layout using a code such as `I_I` for CarSim and TruckSim (vehicle with independent front and rear suspensions), or `FIXED_CASTER` for a BikeSim model with a front suspension that has fixed caster. (In version 2020.0 and a few previous versions, the layout was provided in the Simfile using the keyword `VEHICLE_CODE`.)

CarSim and TruckSim

In CarSim and TruckSim, the `MODEL_LAYOUT` command specifies the number of vehicles, the number of vehicle units, the number of suspensions per unit, and the types of each suspension (solid axle, independent, virtual steering axis, or legacy twist-beam). Based on this information, a VS Math Model is constructed using instances of internal structures for vehicle parts, such as sprung masses, suspensions, wheels, tires, etc. Each internal structure has a set of associated variables (parameters, state variables, output variables, etc.). Along with the structure and associated variables, the VS Math Model is provided with an internal database of keywords that are associated with all variables added in each module.

Because the `MODEL_LAYOUT` statement creates the internal database needed to handle additional information from the input Parsfiles, it must be one of the first lines in the input Parsfile. (Otherwise, keywords for parameters that were not yet installed would be ignored.)

The layout text is generated automatically from the VS Browser for simulations with a single vehicle. For simulations with multiple vehicles simulated in a single instance of the VS Math Model, it is necessary to specify the layout explicitly. Table 1 lists the codes that are used in CarSim and TruckSim.

Table 1. Codes used to build a CarSim or TruckSim layout for the `MODEL_LAYOUT` command.

Code	Example	Description	C/T
I	I_S	Independent suspension	Both
S		Solid-axle suspension	
_		Separator for suspensions on same unit that are not tandem	
__	I_S__SS	Hitch between vehicle units	
<i>space</i>	I_I I_S	Space: Separator between multiple vehicle layout codes	
F_	F_S_S	Prefix for vehicle with sprung masses with frame flex	CarSim
E_	E_I_I	Prefix for vehicle with suspended engine	
V	V_I	Virtual steer axis involving tables for jounce and steer	

More information about the vehicle modules that support these layout codes is provided in the tech memo *Modular Vehicle Models*.

BikeSim

BikeSim 2020.1 also uses the `MODEL_LAYOUT` command, but with different codes that describe one of six front suspensions and whether the rear suspension is rigid or flexible laterally. These codes are provided in the tech memo *Modular Vehicle Models*, along with other information about the BikeSim vehicle modules.

Commands to Extend the Model

The `MODEL_LAYOUT` command defines the layout of the vehicle in terms of types of sprung masses, number of suspensions, etc. But there are still many options that might be added, such as a powertrain, a speed controller, payloads, moving objects, sensors, etc.

These optional features are also added to the VS Math Model with commands. The convention is that the parameters and variables associated with an optional part of the model do not exist until the part is installed with a command. Therefore, it is essential that the command is read by the VS Math Model before any data that would set values for parameters or variables created by the command.

In the cases of the vehicle models, some optional commands for parts are `OPT_PT` (powertrain), `OPT_HEV` (hybrid-electric power), `INSTALL_ENGINE` (internal combustion engine), `DEFINE_PATHS` (reference paths), `INSTALL_SPEED_CONTROLLER`, etc.

In SuspensionSim, there is no `MODEL_LAYOUT` command. The entire model is always assembled with repeated commands such as `SS_DEFINE_POINTS`, `SS_DEFINE_JOINTS`, `SS_DEFINE_VECTORS`, etc. The same convention holds for defining parts before they can be referenced; for example, parameters for a Point cannot be specified until the Point is defined with a `SS_DEFINE_POINTS` command.

Running Simulations with VS Math Models

A VS Solver is a dynamic library file with a set of functions that are used to build a VS Math Model and interact with that model. These functions are documented in the VS API Manual, which is provided with the VehicleSim SDK.

When working with a VS Browser in Windows, you might run a simulation by clicking the **Run Math Model** button from the **Run Control** screen in the Windows VS Browser. When you do so, the Browser loads a VS Solver library (e.g., `carsim_32.dll`), reads a Simfile and Parsfile to obtain a `MODEL_LAYOUT` command to construct a VS Math Model, which then processes more commands and sets values for parameters, variables, and data for configurable functions. The VS Math Model then runs a simulation.

When a VS Math Model runs a simulation, several output files are generated. Figure 1 shows that the VS Browser has a menu item **View > Open Results Folder for this Run in Windows** ①. When selected, Windows will show the folder containing the simulation results ②. Note that most of the files in the figure begin with a common base name `LastRun`. Table 2 summarizes the file types.

The base name (indicated in the table as *base*) is generated by the program controlling the simulation, typically the VS Browser. The names are all passed to the initial VS Math Model by way of the simulation control file, called a *Simfile*, as described in a later section (page 30).

The same VS Math Models can be constructed and run from software other than a VS Browser using the command-line wrapper, or other software created to use the VS API as described in the *VehicleSim API Manual*. When running from other software, the names can be anything permitted by the OS under which the VS Math Model is running.

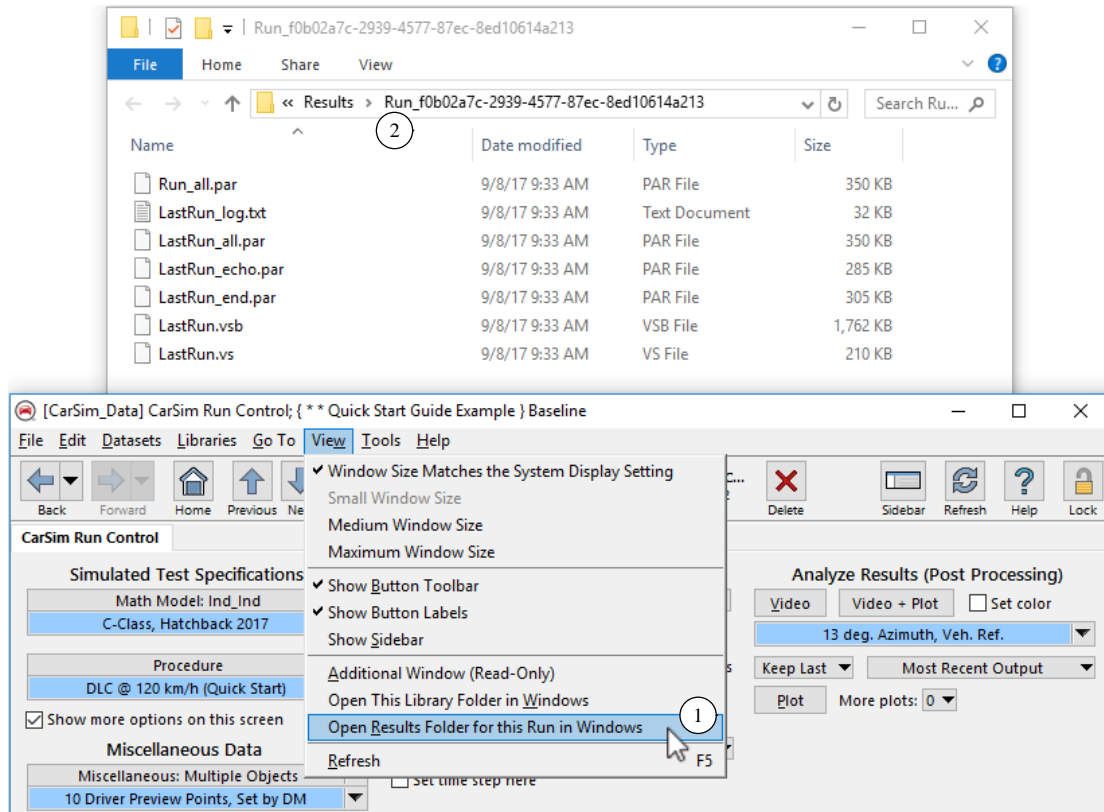


Figure 1. Simulation files generated by a VS Math Model.

Table 2. Standard VS files created when a simulation is run.

Name	Description	Creator
simfile.sim	Simulation control file (located in working directory).	Browser
run_all.par	A single Parsfile containing the contents of the Parsfile associated with the Run Control dataset, plus the contents of all linked files. This file is generated whenever a button on the Run Control screen is clicked to make a run, transfer files, or view results via video and/or plotting.	
base_all.par	Copy of run_all.par made when new simulation is run.	
base_echo.par	List of parameters and Configurable Functions. May also list initial conditions.	VS Math Model
base_end.par	List of parameters, Configurable Functions, and final conditions. May be used to continue a run.	
base.erd or base.vsb or base.csv	Header for output binary file with simulation results. The header has variable names, units, and other labels. The CSV file has name in the first line, numbers in all other lines	
base.bin or base.vsb	Binary file with time histories of variables calculated in the simulation. There is no binary associated with a CSV file.	
base_log.txt	Log of all input files, events, errors, and warnings made when a simulation is run.	

Parsfiles (Parameter Files)

The database, the VS Math Model, and VS Visualizer all use Parsfiles to receive inputs. For example, Figure 2 shows a **Tire** screen from a VS Browser and the associated Parsfile with the data.

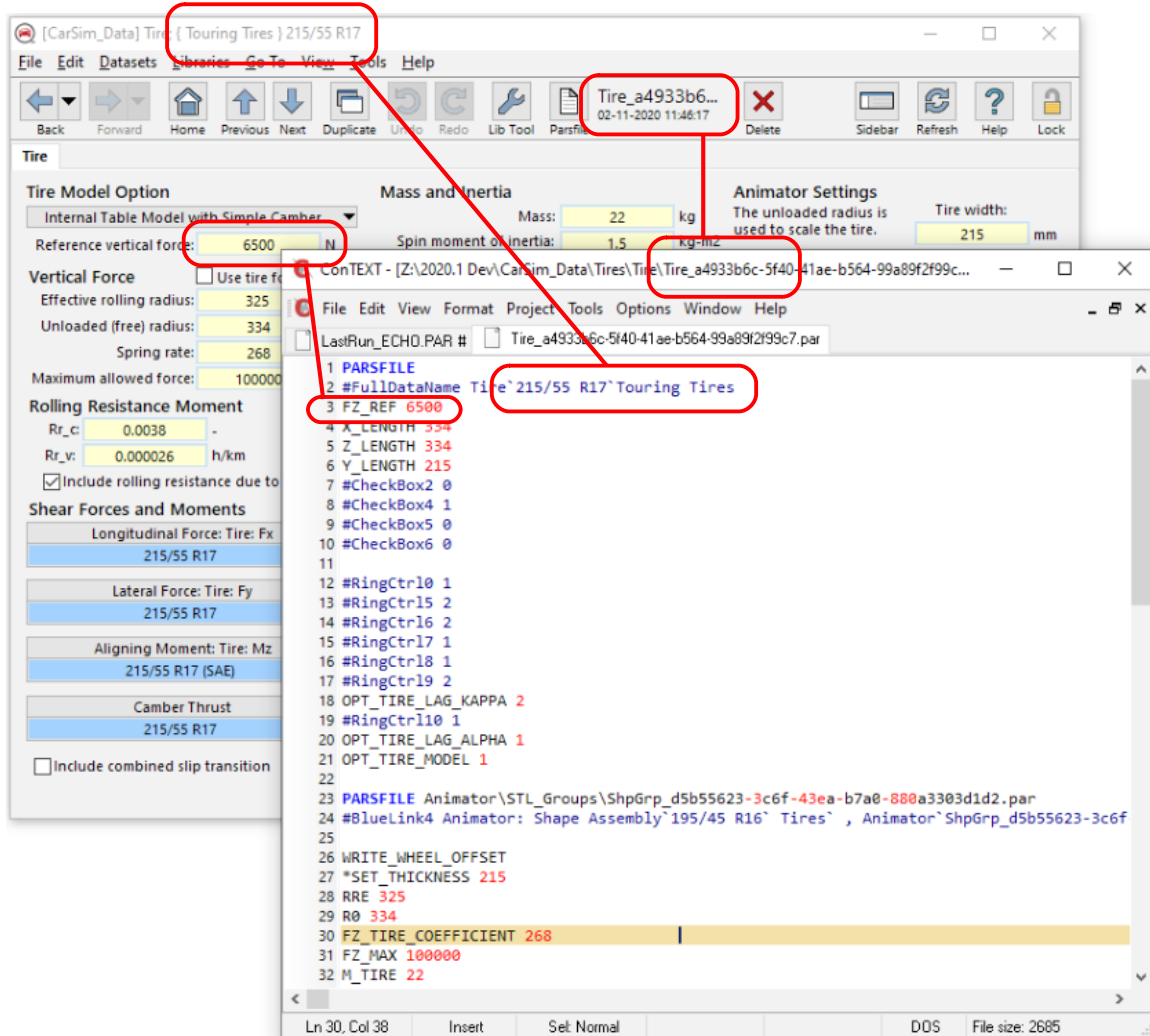


Figure 2. Data screen and associated text Parsfile.

Notice that most information from the screen is also shown in the Parsfile: the title (Touring Tires: 215/55 R17) and values such as reference force (6500). The Parsfile name (Tire_a4933...) is shown in the top of the Browser screen.

Parsfile Content

Most lines of text in a Parsfile have keywords followed by values. Here is the fundamental layout of a Parsfile:

1. The first line is the keyword PARFILE.

2. Other lines that begin with the word `PARSFILE` have one or more spaces followed by a pathname. Programs reading the Parsfile open the file associated with the pathname, read its contents, then resume reading from the original file. (The processing is the same as the `#include` instruction in C.)
3. When a line reading `END` is reached, processing of the file ceases. If the end of the file is reached without an `END` line, processing also ceases. (The `END` line is optional. It offers clarity and the option to put extra information in the file that will not be seen by VS Math Models reading the Parsfile.)
4. All other lines are scanned to see if they begin with a keyword that is recognized. A keyword is an alphanumeric expression that cannot contain spaces, tabs, control (invisible) characters, arithmetic and Boolean operators `^`, `*`, `/`, `+`, `-`, `&`, `|`, `~`, quotes, punctuation marks, or the characters: `!` or `#`. Keywords are not case sensitive. In parsing the line of text, the VS Math Model searches for a space, tab, `'=`' sign, or opening parenthesis `'(`. Some keywords are standard in all VS Math Models; others are specific to the VS Math Model that was constructed for the current run.

If the keyword is recognized, then the effect depends on what the keyword represents. There are just a few possibilities:

- a. The keyword is the name of a parameter or variable in the model. In this case, whatever follows the keyword on that line of text is processed to obtain a numerical value for the parameter or variable, as described in the next chapter (page 34).
- b. The keyword is followed by an index list, made of parentheses enclosing indices, e.g., `A_KPI(1,1)`. In this case, the index list is used to identify an indexed parameter, applied to a part of the model that is repeated. If the keyword and index list identify a valid existing parameter, then whatever follows on that line of text is handled the same as described above for a non-indexed parameter.
- c. The keyword is the root name for an indexed parameter, applied to a part of the model that is repeated, e.g., `A_KPI`. This assigns a value to an indexed parameter, with the indexing obtained using the current value(s) of *index parameter(s)*, as described later (page 38).
- d. The keyword is a VS Extended Assignment command that assigns a value and performs other actions, such as indicating that tabular data associated with a Configurable Function will follow (this is described later, on page 41).
- e. The keyword is a VS Command to perform an action other than assigning a value to a parameter or variable or reading tabular data. VS Math Models support a scripting language called VS Command that can be used to extend the model by adding program modules, defining new variables, adding equations, and more. VS Commands are documented in *VS Commands Manual*.

Comments and Line Continuation Indicators

When scanning a line of text from a Parsfile, the VS Math Model always checks for two special indicators:

1. The character '!' is always interpreted as a comment indicator. If found, the line of text is terminated at the position where the '!' character was found before any other processing.
2. A sequence of three periods '...' is interpreted as an indicator that a self-contained statement continues on the next line. If found, the line of text stops at the position just before the first of the three periods, and continues with the first non-blank character of the next line from the Parsfile.

For example, the following statement is written over three lines using the three-dot continuation indicator.

```
EQ_OUT SPEED_TARGET_CONSTANT(1) = IF(MagS1_1 > 0, ...  
    MIN(speed_limit, vx + SpdS1_1*Distance_ACC/DisS1_1), ...  
    speed_limit)
```

This convention is intentionally like the continuation method used in MATLAB.

Here are a few details about special cases:

1. If a line of text contains three consecutive dots, any content after the three dots is never seen.
2. If the three dots appear after a '!' comment indicator, they are not seen; the comment processing occurs before the line continuation processing.
3. The continuation processing does not occur when reading tabular data from multiple lines from the Parsfile.
4. Although three consecutive periods are commonly used to represent the *ellipsis* character, the ellipsis is not a standard character in ASCII text. Do not use a single ellipsis if generating a Parsfile with an external text editor; always use three consecutive periods to continue a statement.

Comments are commonly used in miscellaneous yellow fields in the VS Browser to document the nonstandard settings made in the field.

Continuations are mainly used for VS Commands, which can sometimes be lengthy.

Links to other Parsfiles

Each screen display in a VS Browser has an associated Parsfile that contains only information related to the contents of the screen. However, screens that have links to other datasets use the PARSEFILE keyword, such that the information read by the VS Math Model can have the contents of hundreds of small Parsfiles.

The titles of linked Parsfiles can be seen in a VS Browser using the **Linked Data** Sidebar view, as shown in Figure 3 for the top-level dataset for a simulation.

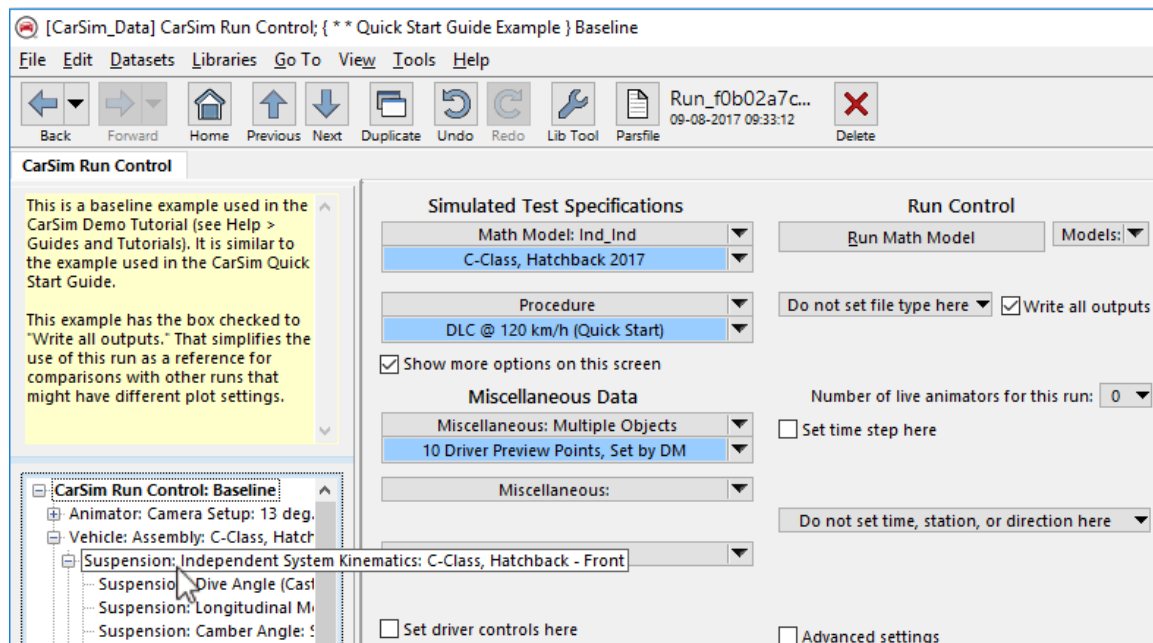


Figure 3. The sidebar displays linked Parsfiles.

In typical use, a single Parsfile is assembled from all the linked Parsfiles and sent to construct a VS Math Model when you click the **Run Math Model** button or VS Visualizer when you click a **Video** or **Plot** button. The single file, with suffix `_ALL.PAR`, contains the information in all files referenced with the keyword `PARSFILE`. Although this *All Parsfile* contains everything needed for the construction and initialization of the VS Math Model, it is still possible that more Parsfiles will be read during the run because of VS Events (see *VS Commands Manual* for details).

The name of each Parsfile is written by the VS Math Model into the Log file to help keep track of the sources of the data used in a run.

Warning If you are running from the Windows VS Browser, never use the keyword `PARSFILE` as a command in a miscellaneous yellow field. See the next subsection for the safe alternative: the `INCLUDE` keyword.

The keyword `PARSFILE` is used extensively by the VS Browser to manage links to datasets and ensure that the necessary datasets are included when Expanded Parsfiles and Consolidated Parsfiles are generated for archiving or transferring data. If the keyword appears in unexpected places, the export files are corrupted and cannot be used as intended.

Using INCLUDE to Support Multiple Input Files

Another keyword with action like `PARSFILE` is `INCLUDE`. The keyword `INCLUDE` is not recognized by the VS Browser, so it is safe for you to use as you see fit.

For example, when the VS Browser **Run Math Model** button is clicked and the All Parsfile is created, any line beginning with `INCLUDE` is written into the expanded file, but the contents of the referenced file are not automatically inserted. Thus, the All Parsfile contains everything except the

information that will be obtained by the VS Math Model using the `INCLUDE` commands to open other files.

This capability is helpful if you are running VS Math Models from external software and modifying part of the data with external software or by hand editing. Some examples might be advanced Simulink models or optimization software. The single All Parsfile can be left intact, and multiple runs can be made by modifying the content in files specified by `INCLUDE` commands.

If runs are controlled completely by external software, another option is to make a top-level Parsfile that in turn references other Parsfiles with settings for the controls, road, vehicle, etc. In this case, the other files can be specified using `INCLUDE`. The Simfile would reference the top-level file using the `INPUT` keyword, and the top-level Parsfile would be rewritten as needed to reference different datasets.

Echo and End Files

When a VS Math Model runs, it creates summary files that list every parameter used. The VS Echo file (whose name ends with `_echo.par`) is created just before the run starts, and the VS End file (whose name ends with `_end.par`) is created at the end of the run. As indicated by the `PAR` extension, the Echo and End files follow the Parsfile format.

The Echo file may be configured to fully describe the state of the model (all parameters, tables, and state variables) at the start of the simulation. The End file is similar, except that it always fully describes the state of the model at the end of the simulation.

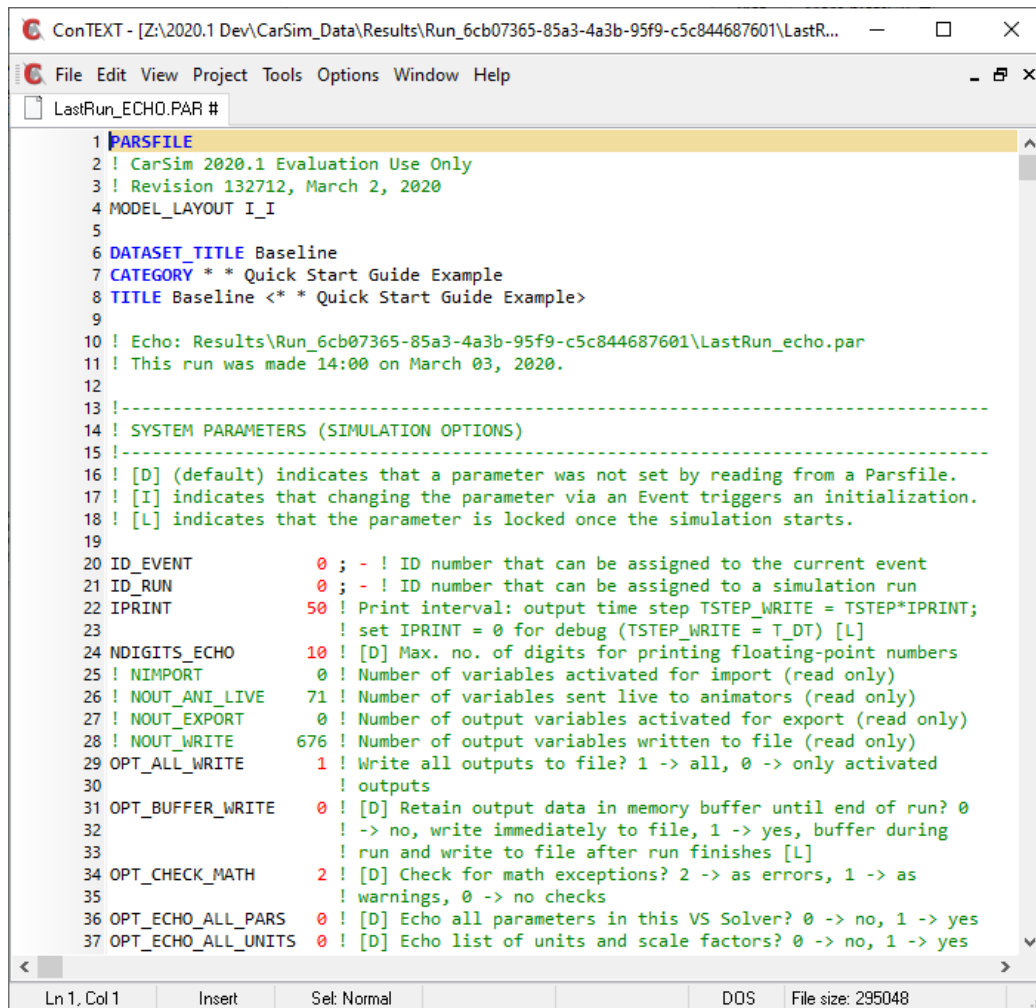
The VS Echo and End files both describe the options that are available in the VS Math Model as constructed for the run. Every parameter or Configurable Function that influenced a specific run is included in the VS Echo file. Additional information is provided using comments; the `'!'` character indicates that everything that follows on that line is to be ignored by the VS Math Model. The ConTEXT text editor installed with a VehicleSim product uses syntax coloring, such that comments are easily seen by color (Figure 4).

Most lines have a keyword for a parameter, followed by the numerical value, the units, and then a comment describing the parameter.

The top of the file identifies the VehicleSim product and version. For vehicle VS Math Models, the first non-comment line has the keyword `MODEL_LAYOUT` followed by a layout code for the vehicle. (In the example, the layout is text is `I_I`, indicating a vehicle with two independent suspensions.) Next are the title of the dataset used to set up the simulation, and the time and date when the run was made.

If an error was encountered before the Echo file was written, a message to that effect is written in the top section.

Some of the inputs to the VS Math Model must be made in the proper sequence; for example, the command `MODEL_LAYOUT` must be first for a Parsfile used for a vehicle model. Also, any command to add an optional feature must be provided before properties of the feature can be specified. The Echo files are organized with two objectives:



```

1 PARSFILE
2 ! CarSim 2020.1 Evaluation Use Only
3 ! Revision 132712, March 2, 2020
4 MODEL_LAYOUT I_I
5
6 DATASET_TITLE Baseline
7 CATEGORY * * Quick Start Guide Example
8 TITLE Baseline <* * Quick Start Guide Example>
9
10 ! Echo: Results\Run_6cb07365-85a3-4a3b-95f9-c5c844687601\LastRun_echo.par
11 ! This run was made 14:00 on March 03, 2020.
12
13 !-----
14 ! SYSTEM PARAMETERS (SIMULATION OPTIONS)
15 !-----
16 ! [D] (default) indicates that a parameter was not set by reading from a Parsfile.
17 ! [I] indicates that changing the parameter via an Event triggers an initialization.
18 ! [L] indicates that the parameter is locked once the simulation starts.
19
20 ID_EVENT          0 ; - ! ID number that can be assigned to the current event
21 ID_RUN            0 ; - ! ID number that can be assigned to a simulation run
22 IPRINT            50 ! Print interval: output time step TSTEP_WRITE = TSTEP*IPRINT;
23                  ! set IPRINT = 0 for debug (TSTEP_WRITE = T_DT) [L]
24 NDIGITS_ECHO      10 ! [D] Max. no. of digits for printing floating-point numbers
25 ! NIMPORT          0 ! Number of variables activated for import (read only)
26 ! NOUT_ANI_LIVE    71 ! Number of variables sent live to animators (read only)
27 ! NOUT_EXPORT      0 ! Number of output variables activated for export (read only)
28 ! NOUT_WRITE       676 ! Number of output variables written to file (read only)
29 OPT_ALL_WRITE     1 ! Write all outputs to file? 1 -> all, 0 -> only activated
30                  ! outputs
31 OPT_BUFFER_WRITE  0 ! [D] Retain output data in memory buffer until end of run? 0
32                  ! -> no, write immediately to file, 1 -> yes, buffer during
33                  ! run and write to file after run finishes [L]
34 OPT_CHECK_MATH    2 ! [D] Check for math exceptions? 2 -> as errors, 1 -> as
35                  ! warnings, 0 -> no checks
36 OPT_ECHO_ALL_PARS 0 ! [D] Echo all parameters in this VS Solver? 0 -> no, 1 -> yes
37 OPT_ECHO_ALL_UNITS 0 ! [D] Echo list of units and scale factors? 0 -> no, 1 -> yes

```

Figure 4. The top part of an example VS Echo file.

1. Provide all parameters and tables used in the simulation, organized to help experienced users find any information of interest.
2. Ensure that the file will work as input to the VS Math Model and reproduce the state of the model.

Default Values and Special Parameters

All parameters in the VS Math Model have default values that can be reset by reading from the input Parsfile(s). If the parameter is not assigned a value, then the description begins with [D] and the value shown is the default (e.g., see NDIGITS_ECHO in Figure 4).

Note The [D] indicator is never shown if the parameter is assigned a value from a Parsfile, even if the assigned value is the same as the original default value. The purpose of the indicator is simply to confirm that a value was assigned from an input Parsfile(s).

Most integer parameters have only a few valid values. For example, the valid values for `OPT_ECHO_ALL_PARS` are 0 and 1. In these cases, the VS Math Model checks the value set in the Parsfile and generate an error message if the value is outside the allowable range of values.

A convention used for describing integer options is that the default value is always listed first. For example, the system parameter `OPT_ALL_WRITE` (line 29) has the description “Write all outputs to file? 1 -> all, 0 -> only activated”, which indicates that the default value that will be used if the parameter is not set in a Parsfile is 1.

As also noted at the top of Figure 4, the Echo files have an indicator to identify parameters whose values influence the initialization of the vehicle model: `[I]`. The initialization (described later) occurs after all the Parsfiles have been read, but before the simulation starts. If a parameter is marked with the `[I]` indicator, it means that if a new value is set during a simulation via a VS Event, then a new initialization will be performed. (See *VS Commands Manual* for details about VS Events.)

A third indicator identifies parameters whose values are locked when the simulation is running: `[L]`. Any parameter that involves the organization of the output file(s) cannot be changed once the file has been created during the initialization, e.g., `OPT_BUFFER_WRITE` (line 31).

Along with parameters that can be assigned values from Parsfiles, there are also internal system parameters whose values are calculated automatically. These are identified by the VS Math Model as “read only.” They are written as comments such that they are not processed when the Echo file is used as an input to a VS Math Model. Figure 4 shows calculated system parameters for the number of optional variables in several categories (e.g., `NOUT_WRITE` is the number of output variables that will be written to file, `NOUT_EXPORT` is the number of output variables activated for export, etc.). Attempts to assign values to read-only variables will generate an error and stop the run.

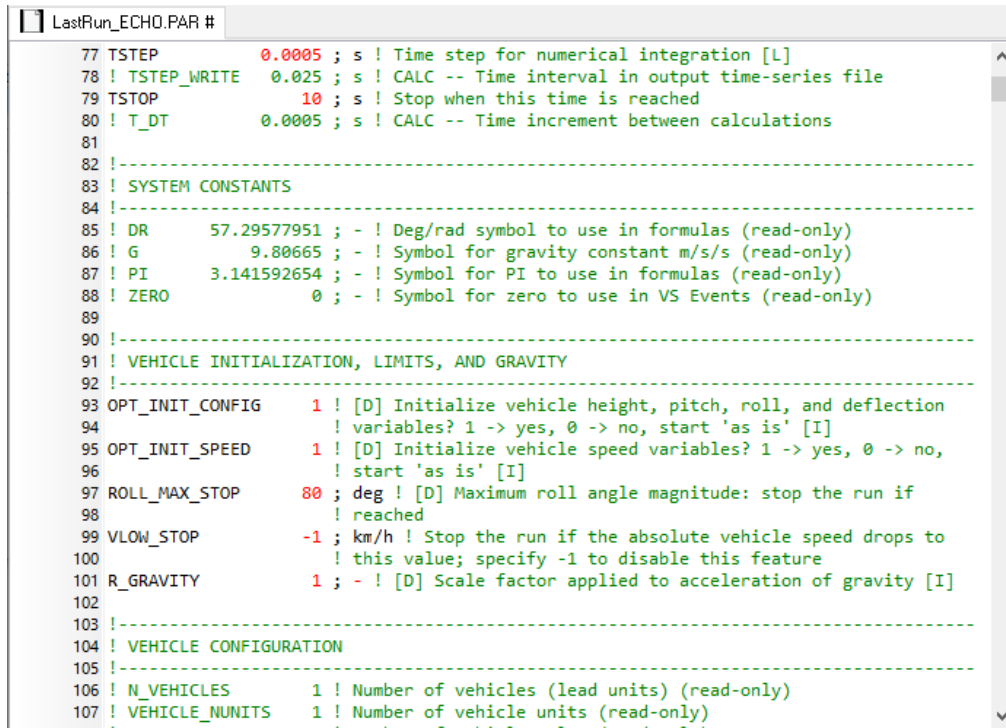
System Parameters

The first section of the Echo file has an alphabetical listing of system parameters that exist in all VehicleSim products, such as the start time, stop time, etc. Additional information about these parameters is provided in the *VS System Parameters* reference manual.

Figure 5 shows the very end of the System Parameters section, with time step `TSTEP`, and possibly the stopping time `TSTOP`. The calculated read-only parameter `TSTEP_WRITE` is the time interval between outputs written to file and the read-only parameter `T_DT` is the time increment between calculations done internally during the simulation. Both calculated parameters are sometimes helpful for advanced users adding equations to the model with VS Commands.

Note Calculated parameters such as `TSTEP_WRITE` and `T_DT` are shown as comments to prevent conflicts in case an Echo file is used later as an input file.

Next is a section showing four read-only parameters that are available for advanced users to include in VS Commands.



```

77 TSTEP      0.0005 ; s ! Time step for numerical integration [L]
78 ! TSTEP_WRITE 0.025 ; s ! CALC -- Time interval in output time-series file
79 TSTOP      10 ; s ! Stop when this time is reached
80 ! T_DT       0.0005 ; s ! CALC -- Time increment between calculations
81
82 !-----
83 ! SYSTEM CONSTANTS
84 !-----
85 ! DR         57.29577951 ; - ! Deg/rad symbol to use in formulas (read-only)
86 ! G          9.80665 ; - ! Symbol for gravity constant m/s/s (read-only)
87 ! PI         3.141592654 ; - ! Symbol for PI to use in formulas (read-only)
88 ! ZERO       0 ; - ! Symbol for zero to use in VS Events (read-only)
89
90 !-----
91 ! VEHICLE INITIALIZATION, LIMITS, AND GRAVITY
92 !-----
93 OPT_INIT_CONFIG 1 ! [D] Initialize vehicle height, pitch, roll, and deflection
94 ! variables? 1 -> yes, 0 -> no, start 'as is' [I]
95 OPT_INIT_SPEED  1 ! [D] Initialize vehicle speed variables? 1 -> yes, 0 -> no,
96 ! start 'as is' [I]
97 ROLL_MAX_STOP   80 ; deg ! [D] Maximum roll angle magnitude: stop the run if
98 ! reached
99 VLOW_STOP       -1 ; km/h ! Stop the run if the absolute vehicle speed drops to
100 ! this value; specify -1 to disable this feature
101 R_GRAVITY       1 ; - ! [D] Scale factor applied to acceleration of gravity [I]
102
103 !-----
104 ! VEHICLE CONFIGURATION
105 !-----
106 ! N_VEHICLES   1 ! Number of vehicles (lead units) (read-only)
107 ! VEHICLE_NUNITS 1 ! Number of vehicle units (read-only)

```

Figure 5. Part of Echo file with simulation time parameters and vehicle global settings.

Model Parameters

More sections in the Echo file follow with model-specific parameters. The first are global settings, shown in Figure 5 for CarSim (starting with line 93). These include the initialization options, stop options, and a gravity scale factor.

The next section — also shown in in Figure 5 — has a summary of the vehicle configuration, including the number of vehicles, vehicle units, axles, etc.

Figure 6 shows another part of the Echo file, with parameters specific to the vehicle model, along with comments that provide more documentation. For example, the Aerodynamics section mentions the names of related Configurable Functions that are part of the aerodynamics model. As shown in Figure 6, some of the function names include the text `_AERO_SHAPING`. To see the information for this function, you will typically select text and the search function (Ctrl+F) in the text editor to quickly find the tabular data (Figure 7).

Configurable Function Data (Tables)

The largest part of the Echo file lists all data for VS Configurable Functions. This follows all the data for scalar parameters. Regardless of how they were defined or what module they might be associated with, data for these functions are listed alphabetically by the function name. For example, data for the aerodynamic shaping function `FX_AERO_SHAPING` follow data for the function `FUEL_RATE` and precede data for the function `FX_AERO_SHAPING_2` (Figure 7).

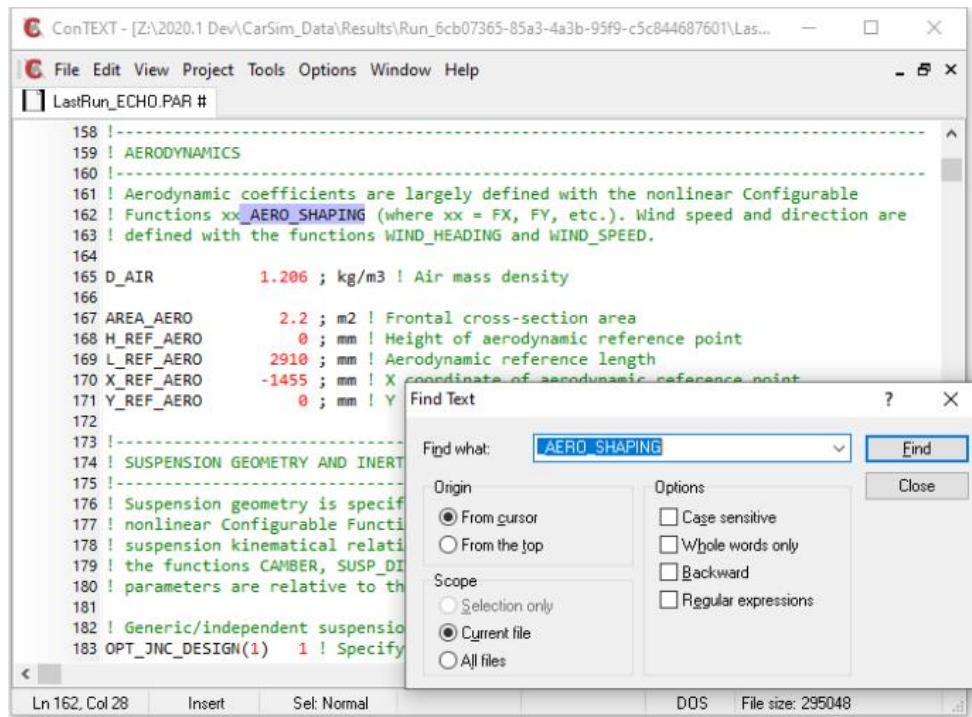


Figure 6. Part of Echo file with more documentation and model parameters.

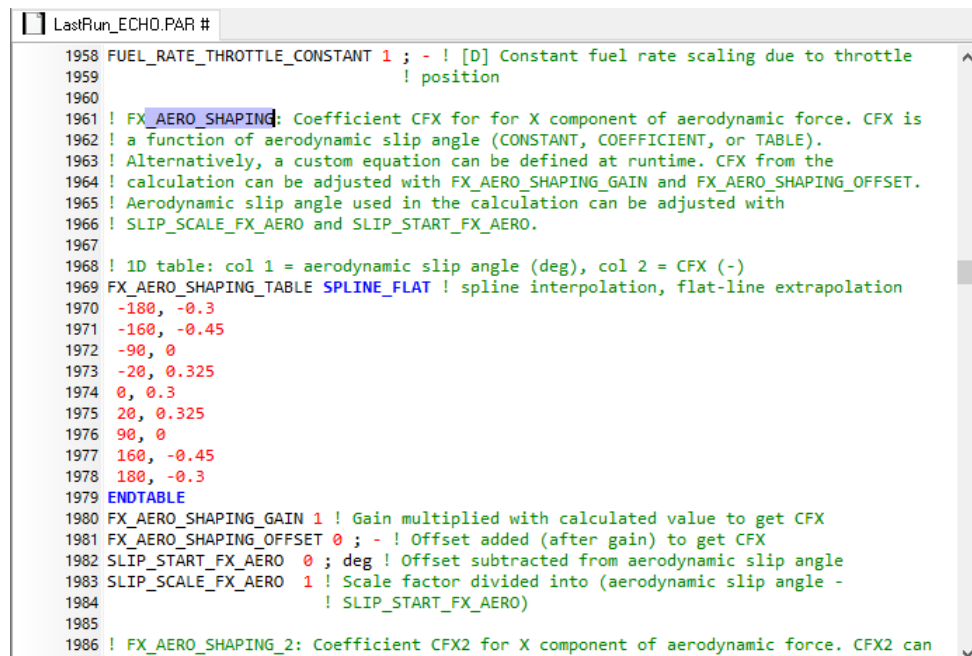


Figure 7. Configurable Functions used in the Aerodynamics part of the model.

Nearly all the Configurable Function root names are associated with preceding sections, as shown for the aerodynamic shaping functions mentioned in comments in the parameter section (Figure 6).

Remainder of the Echo File

The Echo file has a few more sections after the Configurable Functions.

1. If new variables were defined with VS Commands, these are listed next.
2. If equations were added to the model with VS Commands, they are listed.
3. The `_end.par` file lists all the state variables and their values at the end of the run. If the system parameter `OPT_ECHO_ICS` has a nonzero value, then the state variables and their initial values are also listed in the `_echo.par` file.
4. If any variables were activated for import or export, the `IMPORT` and `EXPORT` commands are written.
5. If the system parameter `OPT_ECHO_WRITE` has a nonzero value, then output variables activated for writing are written last with explicit `WRITE` commands. The variables are listed along with their values at the time the Echo file is written.

Repeat and Continue Options

Advanced users may use an Echo or End file to repeat or continue a reference simulation, with some limits. The Echo file made at the beginning of the run should be able to reproduce the results, with a few recommendations:

1. If the initial position of the vehicle is set explicitly with state variables, the parameter `OPT_ECHO_ICS` should be set to 1 so the state variables are listed along with their values.
2. The parameter `OPT_ECHO_WRITE` should be set to 1 so `WRITE` commands are written in the Echo file for all output variables that were activated for writing to file. Alternatively, the `OPT_ALL_WRITE` parameter may be set to 1 to unconditionally activate all outputs.
3. If VS Events are defined that reference additional Parsfiles, then those Parsfiles must be available if the simulation is run from the Echo file. (See *VS Commands Manual* for details about VS Events).

The Echo file may be used to reproduce a simulation. It can be used to recreate the data for the reference simulation, and then other Parsfiles can be used to modify some parameters and/or conditions.

The End file is designed to allow the run to be continued. The three recommendations listed above also apply for setting up a run that will be continued.

If used alone, the End file will continue the reference simulation with the same vehicle properties and controls. However, you can use other Parsfiles to modify parameters and/or conditions. Be aware that the start time (`TSTART`) in the End file is the time when the first simulation stopped, and the stop time (`TSTOP`) is the new `TSTART` plus the duration of the reference run. It is usually necessary to modify `TSTOP`. Also, depending on the use of Paths, it might be necessary to modify `SSTART`, `SSTOP`, and `OPT_DIRECTION`.

If working from the Windows Browser for your product (e.g., `carsim.exe`), the **Run Control** screen has controls to simplify the continuation of an existing run. See the **Help** menu item **Home: The Run Screen** for details.

Care must be taken if the information from an Echo or End file is combined with information from other Parsfiles. The Echo/End files may include commands that add features to the model (Payloads, Moving Objects, etc.). Be careful when continuing a run in Simulink or other external software involving a transfer of information using arrays of Import and Export variables. The End file will include the `IMPORT` and `EXPORT` commands that were used for the reference run. (`IMPORT` and `EXPORT` commands are described in Chapter 4.) Be sure to avoid duplicating those commands for a continuation run.

Machine-Generated Documentation for Model Variables

Most VS Math Models recognize thousands of keywords. Some are specific to the VS Math Model in the Solver, while others are shared in all VS Math Models. Most of the keywords used for a specific VS Math Model are documented in files that are themselves generated by the VS Math Model.

Model Parameters and Configurable Functions

The Echo file is the ultimate reference document for the parameter and table names recognized by the VS Math Model for a specific vehicle configuration. One of the system keywords recognized by all VS Math Models is `OPT_ECHO_ALL_PARS` (Figure 4, page 17). Enter this keyword followed by a 1 in a miscellaneous data field, and after the run is made, the Echo file will show other keywords that were not used in the current run or would normally be hidden. For example, parameters related to steering on a rear axle are normally hidden if the axle is not steered. Running with `OPT_ECHO_ALL_PARS` set to 1 will reveal the other keywords that could have been used. This setting will also reveal the existence of Configurable Functions that were not used, but are available for other situations.

The Echo file generated at the end of the simulation also lists all state variables for the model and their values at the end of the run. This information is provided to support continuation of the run, but also serves to document all state variables of the VS Math Model as assembled for the run.

Note	A Parsfile that is referenced with a VS Event is not read until the Event is triggered (see <i>VS Commands Manual</i> for information about Events). Values that are set in a Parsfile that was loaded when an Event triggered were not in effect when the run started and will not appear in the Echo file. Furthermore, values that were changed during the simulation will not be in effect at the end of the run, so the early values will not appear in the End file.
-------------	--

Machine-Generated Documentation Files for Variables

Besides solving equations of motion, VS Math Models are used to generate documentation about the model, including options that were installed in the input Parsfiles. This capability is used in VehicleSim products to generate five types of text and spreadsheet files that are viewed using the **View** button in the lower-right corner of the **Run Control** screen (Figure 8). VS Math Models also generate the tabbed-text and spreadsheet files used to browse the variables that can be specified for plotting from the **Plot: Setup** screen, specified for import from the **I/O Channels: Import** screen,

specified for export from the **I/O Channels: Export** screen, and specified for writing from the **I/O Channels: Write** screen.

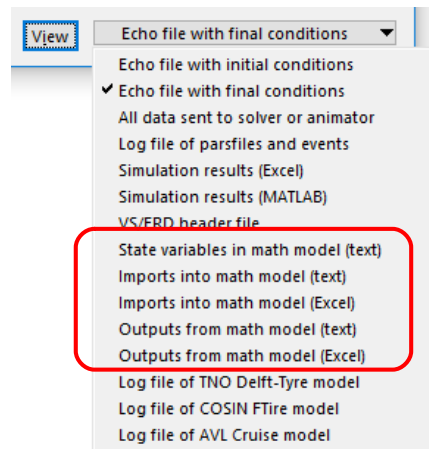


Figure 8. Drop-down menu of file types that are available for a specific simulation setup.

If you click the **View** button with the file type being State variables, Import variables, or Output variables, the current **Run Control** dataset (and all linked datasets) are sent to the VS Math Model, which generates a documentation file. Note that files with Imports and Outputs can be generated in either text or spreadsheet (.xls) form.

These documentation files are used elsewhere in the Browser. Table 3 lists seven types of documentation files that can be written by a VS Math Model from a VS Browser. In the table, *base* is a base name, typically shared by all files associated with a given dataset. In addition to using the **View** button on the **Run Control** screen, the documentation files can be generated and viewed from other locations within the Browser as listed in the **Where** column.

The tabbed text files with extension .txt are used to support built-in browsing for the I/O screens and Plot Setup (for outputs). The same files are written with the extension .xls to load easily into Microsoft Excel and other spreadsheet applications.

State Variables

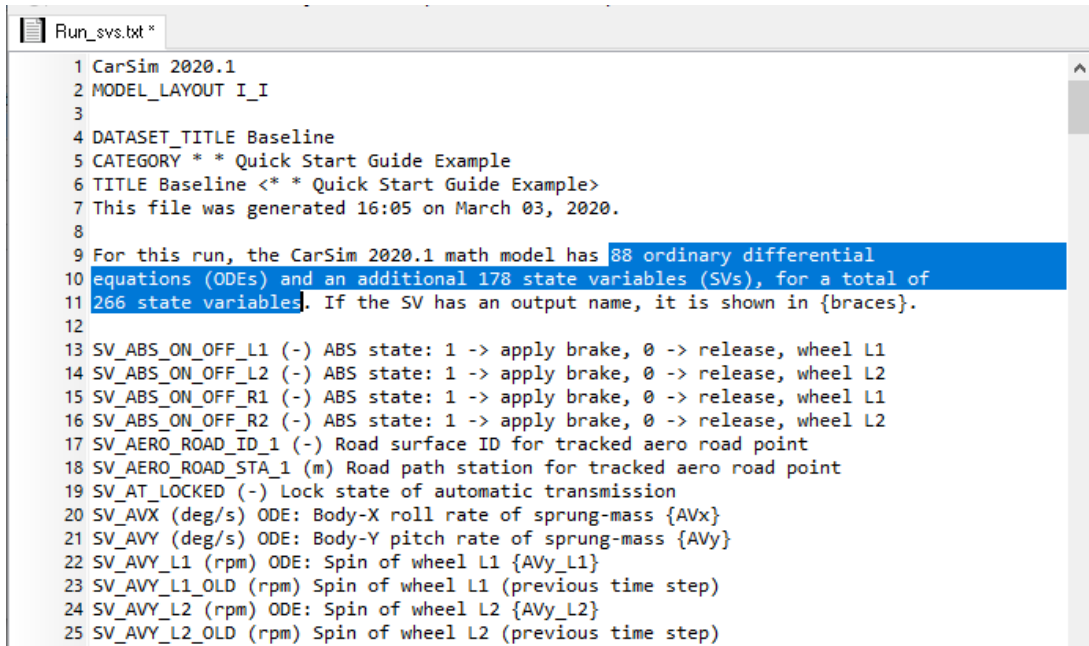
If you click the **View** button with the file type being **State variables**, the Browser and VS Math Model will generate a text file listing all the state variables for the model as specified with the current **Run Control** dataset and all linked data (Figure 9).

Recall from an earlier section that the state of the VS Math Model is defined by the values of Parameters, Configurable Functions, and State Variables. The Parameters and Configurable Functions settings are expected to mostly remain constant during a simulation, whereas the state variables are expected to change as the simulation proceeds. Also, recall that some of the state variables are calculated by numerically integrating ODEs, while the others are calculated as needed to describe some property of the model that must be preserved for use in the next time step.

The top of the text file summarizes the types of state variables. In this example, there are 88 ODEs and an additional 178 state variables, for a total of 266 (lines 9 - 11).

Table 3. Standard machine-generated documentation files for a VS Math Model.

Name	Where	Contents
<i>base_svs.txt</i>	Run Control	A list of all state variables, with keyword and short description, whether each is an ODE variable. Viewable with any text editor.
<i>base_outputs.txt</i>	Run Control	A list of all output variables, with keywords, units, and labels used for plotting. Viewable with any text editor.
<i>base_outputs_tab.txt</i>	I/O Export + I/O Write + Plot Setup	A tabbed-text spreadsheet with each row describing an output variable, and columns giving the keyword, category, component name, long name, and units. Viewable with Excel and other spreadsheet programs.
<i>base_outputs.xls</i>	Run Control + Plot + I/O	
<i>base_imports.txt</i>	Run Control	A list of all Import variables, with keyword, units, category, and short description. Viewable with any text editor.
<i>base_imports_tab.txt</i>	I/O Import	A tabbed-text spreadsheet with each row describing an Import variable, and columns giving the keyword, category, description, units, and an indicator of whether the import can be combined with an internal variable using the REPLACE and MULTIPLY modes. Viewable with Excel.
<i>base_imports.xls</i>	Run Control I/O Import	



```

1 CarSim 2020.1
2 MODEL_LAYOUT I_I
3
4 DATASET_TITLE Baseline
5 CATEGORY * * Quick Start Guide Example
6 TITLE Baseline <* * Quick Start Guide Example>
7 This file was generated 16:05 on March 03, 2020.
8
9 For this run, the CarSim 2020.1 math model has 88 ordinary differential
10 equations (ODEs) and an additional 178 state variables (SVs), for a total of
11 266 state variables. If the SV has an output name, it is shown in {braces}.
12
13 SV_ABS_ON_OFF_L1 (-) ABS state: 1 -> apply brake, 0 -> release, wheel L1
14 SV_ABS_ON_OFF_L2 (-) ABS state: 1 -> apply brake, 0 -> release, wheel L2
15 SV_ABS_ON_OFF_R1 (-) ABS state: 1 -> apply brake, 0 -> release, wheel L1
16 SV_ABS_ON_OFF_R2 (-) ABS state: 1 -> apply brake, 0 -> release, wheel L2
17 SV_AERO_ROAD_ID_1 (-) Road surface ID for tracked aero road point
18 SV_AERO_ROAD_STA_1 (m) Road path station for tracked aero road point
19 SV_AT_LOCKED (-) Lock state of automatic transmission
20 SV_AVX (deg/s) ODE: Body-X roll rate of sprung-mass {AVx}
21 SV_AVY (deg/s) ODE: Body-Y pitch rate of sprung-mass {AVy}
22 SV_AVY_L1 (rpm) ODE: Spin of wheel L1 {AVy_L1}
23 SV_AVY_L1_OLD (rpm) Spin of wheel L1 (previous time step)
24 SV_AVY_L2 (rpm) ODE: Spin of wheel L2 {AVy_L2}
25 SV_AVY_L2_OLD (rpm) Spin of wheel L2 (previous time step)

```

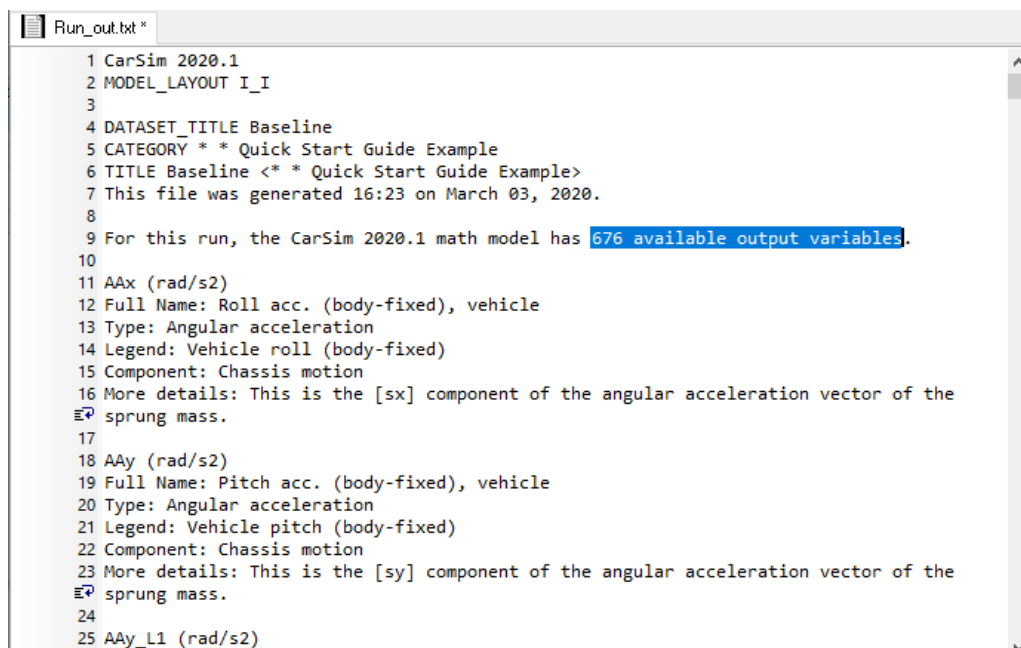
Figure 9. Text file generated with list of all state variables for the current Run data.

The remainder of the file lists every existing state variable sorted alphabetically by keyword, then the units, then a designation ODE if it is an ODE state variable, and then a description. Note that all keywords begin with a prefix `SV_`, so they can be easily identified by advanced users when setting initial conditions or creating formulas for VS Commands.

Most of the state variables exist for internal use in the VS Math Model; however, some are also available as output variables for plotting and supporting animation. When a state variable is also an output, the name used for accessing the variable as an output is shown in {braces} (see lines 20, 21, 22, and 24).

Output Variables

Simulation results are largely viewed using output variables that support animation, plots, and post-processing analyses with third-party tools. The same **View** button and drop-down menu (Figure 8) can be used to view a list of all output variables available from the VS Math Model, as set up with the data defined with the current **Run Control** data. Each variable has several associated labels to support automated plot generation. For example, Figure 10 shows the top of a text file listing all the outputs available from the same example used for the previous figure.



```

1 CarSim 2020.1
2 MODEL_LAYOUT I_I
3
4 DATASET_TITLE Baseline
5 CATEGORY * * Quick Start Guide Example
6 TITLE Baseline <* * Quick Start Guide Example>
7 This file was generated 16:23 on March 03, 2020.
8
9 For this run, the CarSim 2020.1 math model has 676 available output variables.
10
11 AAx (rad/s2)
12 Full Name: Roll acc. (body-fixed), vehicle
13 Type: Angular acceleration
14 Legend: Vehicle roll (body-fixed)
15 Component: Chassis motion
16 More details: This is the [sx] component of the angular acceleration vector of the
17 sprung mass.
18
19 AAy (rad/s2)
20 Full Name: Pitch acc. (body-fixed), vehicle
21 Type: Angular acceleration
22 Legend: Vehicle pitch (body-fixed)
23 Component: Chassis motion
24 More details: This is the [sy] component of the angular acceleration vector of the
25 sprung mass.
26
27 AAL1 (rad/s2)

```

Figure 10. Text file listing all available output variables.

In this case, the file indicates that there are 676 available outputs (line 8). The file shows all outputs in alphabetical order, with the short name (which will not contain any blanks) and units, then a longer name that may have separate words, a “type” sometimes used to label axes with overlay plots, a “legend” label used to generate legends for overlay plots, a “Component” name used to support interactive browsing and sorting in spreadsheets, and sometimes, additional information.

In older versions of the software, output variable names were limited to eight characters. The limit was increased to 32 characters in current versions. In support of legacy utility programs for managing ERD format output files (described later, page 29), 8-character alias names are provided

for any outputs whose names exceed 8 characters in length. Figure 11 shows both names are listed in the documentation file.

The screenshot shows a text editor window with two tabs: 'LastRun_ECHO.PAR #' and 'Run_out.txt #'. The text in the window is as follows:

```

1484
1485 GPS_Altitude (m)
1486 8-Char Alias: GPSAltid
1487 Full Name: GPS altitude (Zo + GPS_REF_ALT)
1488 Type: GPS altitude
1489 Legend: Vehicle
1490 Component: Chassis motion
1491
1492 GPS_Lat (deg)
1493 Full Name: GPS relative latitude
1494 Type: GPS latitude
1495 Legend: Vehicle
1496 Component: Chassis motion
1497
1498 GPS_LatA (deg)
1499 Full Name: GPS absolute latitude
1500 Type: GPS latitude
1501 Legend: Vehicle
1502 Component: Chassis motion
1503
1504 GPS_LatO_1 (deg)
1505 8-Char Alias: GPS_Lat1
1506 Full Name: GPS latitude, object 1
1507 Type: GPS latitude
1508 Legend: Object 1
1509 Component: Targets
  
```

Figure 11. Outputs with names longer than 8 characters have 8-char aliases.

The VS Math Model can also write the same information in a CSV file, for use in a spreadsheet application such as Microsoft Excel. As shown in Figure 12, the outputs may easily be organized by other labels, such as the Component Type.

Import Variables

VS Math Models may be connected to external models made with third-party tools such as Simulink, from The MathWorks. As described later in Chapter 4, they communicate with arrays of Import and Export variables. All Output variables are available for export. VS Math Models also include variables that can be imported.

The same **View** button and drop-down menu (Figure 8, page 23) can be used to view a list of all import variables available from the VS Math Model, as set up with the data defined with the current **Run Control** data. As with the outputs, the imports can be obtained in a text file (Figure 13) or spreadsheet.

As with the other machine-generated documents, this gives the number of available import variables (line 9), followed by an alphabetical list of all imports. Note that all keyword names begin with the prefix `IMP_`. Each entry has the variable name/keyword, then the units, then a component shown in {braces}, and then a description.

Keyword	Units	Units Type	Component Type	Full Label	Specific Component	Extra Info
1 Yaw	deg	Angle	Chassis motion	Yaw, vehicle	Vehicle yaw	
119 Ycg_SM	m	Global Y coord	Chassis motion	Y coordinate, CG, sprung mass	CG, sprung mass	
120 Ycg_TM	m	Global Y coord	Chassis motion	Y coordinate, inst. CG, vehicle	Instant CG, vehicle	
						This is the local Y coordinate of the instant CG of the vehicle unit in the sprung mass coordinate system. Be aware that the motion of the instant CG cannot be measured directly in physical vehicles.
121 Ylcm	mm	Local Y coordin	Chassis motion	Local Y, inst. CG, vehicle	Instant CG, vehicle	
122 Yo	m	Global Y coord	Chassis motion	Y coordinate, vehicle origin	Vehicle	
123 Zcg_SM	m	Global Z coord	Chassis motion	Z coordinate, CG, sprung mass	CG, sprung mass	
124 Zcg_TM	m	Global Z coord	Chassis motion	Z coordinate, inst. CG, vehicle	Instant CG, vehicle	
125 Zo	m	Global Z coord	Chassis motion	Z coordinate, vehicle origin	Vehicle	
						This output gives the value of the system parameter ID_EVENT. The parameter will keep its default value of 0 unless it is changed by a VS Event or other VS Command.
126 ID_Event	-	ID number	Database	ID number of event	ID of event	
						This output gives the value of the system parameter ID_RUN. The parameter will keep its default value of 0 unless it is changed by a VS Event or other VS Command.
127 ID_Run	-	ID number	Database	ID number of run	ID of run	
128 Lat_Targ	m	Lateral distanc	Driver model path	Target lateral offset from path	Target	
129 Lat_Veh	m	Lateral distanc	Driver model path	Vehicle lateral distance to path	Vehicle	
130 Ltrgldm	-	ID of LTARG da	Driver model path	ID of LTARG dataset used by DM	Target	
131 Pathldm	-	ID of path data	Driver model path	ID of path dataset used by DM	Target	
						Global pitch of the ground is calculated by the closed-loop speed controller when using path

Figure 12. Spreadsheet with outputs, sorted by component type and keyword.

```

Run_imp.txt *
1 CarSim 2020.1
2 MODEL_LAYOUT I_I
3
4 DATASET_TITLE Baseline
5 CATEGORY * * Quick Start Guide Example
6 TITLE Baseline <* * Quick Start Guide Example>
7 This file was generated 16:26 on March 03, 2020.
8
9 For this run, the CarSim 2020.1 math model has 326 available import variables.
10
11 Import variables that are combined with internal math model variables (ADD,
12 MULTIPLY, or REPLACE) are identified below with "[V]".
13
14 IMP_AV_D3_F (rpm) {Powertrain} Output shaft speed of transfer case to front
15 differential
16 IMP_AV_ENG (rpm) {Powertrain} Engine speed (external engine only)
17 IMP_AV_TC (rpm) {Powertrain} Torque converter speed (external transmission only)
18 IMP_AV_TRANS (rpm) {Powertrain} Transmission output speed [V]
19 IMP_AX_FIELD (m/s2) {Environment} Field acceleration in global X direction
20 IMP_AX_SC (g) {Speed controller} Acceleration command for speed controller [V]
21 IMP_AY_FIELD (m/s2) {Environment} Field acceleration in global Y direction
22 IMP_AZ_FIELD (m/s2) {Environment} Field acceleration in global Z direction
23 IMP_BK_STAT (-) {Brakes} Brake apply status: 0 or 1 (based on control pressure) [V]
24 IMP_CLT_D1_2 (-) {Powertrain} Clutch control for front differential (second clutch)
25 IMP_CLUTCH (-) {Powertrain} Clutch control for transmission [V]
26 IMP_CLUTCH_D1 (-) {Powertrain} Clutch control for front differential

```

Figure 13. Text file listing all available Import variables.

Simulation Output Files

The main purpose of the VS Math Model is to calculate time histories of variables of interest. Those time histories are stored in either an output text file or a binary file with an associated text header file that describes the layout of the binary file.

The file with simulation time histories can either be written as the run proceeds, or, the information can be saved in memory and written to file when the run terminates. The option for writing at termination requires sufficient memory to store all of the data until the run ends. It is mainly used for real-time applications with hardware in the loop.

VS Math Models can output thousands of variables into the output file. A later subsection *Writing Variables to File for Post Processing* (page 60) describes how to specify which variables are written to file.

The time histories written to file consist of values of the selected output variables, sampled at a constant interval of time called a *time step*. The internal time step is specified with the VS system parameter named `TSTEP`. Recall that the output interval is a calculated system parameter `TSTEP_WRITE`, listed in the Echo file (Figure 5, page 19). Internally, calculations are made with a small time step that is sufficient to solve the internal equations of the VS Math Model. For more information about the various time steps used by a VS Browser, please see Chapter 5 (page 62).

Simfiles generated by the VS Browser always specify an output file name. However, the output file is optional from the point of view of the VS Math Model. If you are running VS Math Models from other software, you may omit the output files.

VS Math Models support three formats for these output files: VS, ERD, and CSV. The type of file is specified with the system parameter `OPT_VS_FILETYPE`.

VS Files

The VS file format has been use in VehicleSim products starting with Version 9 (2015). VS files support binary files with either 32-bit or 64-bit floating-point numbers, and names and labels that support automated plotting in VS Visualizer.

When the VS format is used (`OPT_VS_FILETYPE` = 1 for 64-bit; 2 for 32-bit), the binary file has the extension `VS.B`.

CSV Spreadsheet Files

Comma-separated-variable (CSV) files are text files supported by Microsoft Excel and many other programs that handle tabular data. VS Math Models can generate the files directly, and VS Visualizer can read CSV files to provide animation and plotting. Along with the CSV files, the VS Math Model also generates a VS file. The VS Visualizer reads the generated VS file in conjunction with the CSV file to get the time stamp and header information from the VS file, as long as they have the same number of variables. As mentioned in subsequent sections, there are other methods to generate CSV files along with VS or ERDS files, where the CSV file may be a subset of the VS or ERD file. In such cases, the VS Visualizer reads from the VS or ERDS file.

The first row of the CSV spreadsheet has names of the variables, including the time stamp (`T_Stamp`) as the first variable. All other rows have samples of the variables.

When compared to binary files, the CSV format takes more space and is less efficient for reading and writing. However, with modern PCs, these technical disadvantages are often negligible. The main limitation is that there is no standard means to provide auxiliary labeling information, so plots will not show units or any identifying information other than the short name of the output variable. Also, plots with more than one pair of variables will have a generic label on the Y axis.

ERD and BIN Files

The ERD file format was developed in 1984 for use within the Engineering Research Division (ERD) at The University of Michigan Transportation Research Institute (UMTRI). Binary time-history data are recorded using 32-bit floating-point numbers. The text header has the extension ERD and contains short names and unit names that contain eight characters, and longer names with 32 characters. When the ERD format is used, the binary file has the extension BIN.

Starting with version 2018, ERD files generated by VS Math Models have alternative short names that do not have the eight-character limit. VS Visualizer and the VS File utility both recognize the new short names and will use them to identify channels.

Note	The original ERD file header uses the keyword <code>SHORTNAM</code> to identify unique short names that are ≤ 8 characters in length. The ERD files generated with VS Math Models 2018.0 and newer use the keyword <code>NAME</code> to identify unique names that are ≤ 32 characters in length. Utility programs written by users prior to 2018 will need to be modified to make use of the new <code>NAME</code> keyword. However, the ERD files still contain the original short names, so existing data processing tools should still work.
-------------	--

Future developments of the VehicleSim technology will extend the use of VS files but not ERD files. However, there are no plans to discontinue the ERD format as an output option.

Excel and MATLAB Files with Copied Data

The VS Browser has the capability to read the VS/VSB or ERD/BIN files produced by a VS Math Model, extract data for variables of interest, and write the data in a new file for Excel (type = CSV) or MATLAB (type = MAT). The conversion is so fast that it might seem that the VS Math Models generate these files.

MAT files are always generated by extracting data from a VS or ERD file.

CSV files can be generated directly, as described in the previous section, or they can be generated using the same method as the MAT files. The option for extracting data from a VS or ERD file to create a CSV file can be useful if the CSV file will have a subset of the data produced for the full VS/ERD output file.

If a CSV file is generated directly by the VS Math Model, there is no option to create a second CSV file with a subset of the data. If a CSV file is desired with a subset, then the VS Math Model must generate a VS or ERD file. (This avoids potential name conflicts.)

Log Files

The VS Math Model writes a text log file for each run. During the initialization, it can echo every line that begins with the keyword `LOG_ENTRY`. It also can log every Parsfile that is referenced with an `INCLUDE` or `PARSFILE` keyword. It logs the end of the run at termination. During the run, it can log the processing of VS Events (described in *VS Commands Manual*) if any are triggered. If any errors or warnings are reported during the run, these are also logged.

VS Math Models have options to control the level of detail in a log file, to reduce the size when very long runs are made, or when many Events are being processed. The options are listed and defined in the Echo file made by the VS Math Model and in the *VS System Parameters* reference manual.

Simfile (Simulation Control File)

When the VS Browser runs a VS Math Model, it automatically creates a simulation control file (Simfile). This file has the top-level information for use by the VS Math Model to identify an input file, create output files, and other information.

In some cases, the Simfile is also used by an external program that will control the simulation (e.g., Simulink). For this role, it also has information to identify the VS Math Model and possibly other information (e.g., define I/O ports).

By default, it is given the name `simfile.sim`.

Note Although the names “Simfile” and “`simfile.sim`” are used throughout this document, the VS Browser can generate control files with arbitrary names in support of some advanced applications. Regardless of the name, the VS Browser generates the Simfile, loads the VS Solver library, and passes the name of the Simfile to a function in the library that results in the construction of VS Math Model that performs the simulation.

One situation that requires an alternate name to be assigned to a Simfile is when multiple VS Math Models are run in parallel under the control of external software such as Simulink or LabVIEW. Each VS Math Model must have different inputs. This is accomplished by typing a distinct name in the datasets for the separate vehicles. Each model reads its own Simfile, gets distinct information as input, and generates properly named output files.

Whenever you click the **Run Math Model** button in a VS Browser, a new Simfile is created, typically in the root folder of the current database (this is also the Windows working directory when you are interacting with the VS Browser). If a file with the specified name already exists, it is overwritten. Because the Simfile is automatically re-generated each time a run is made, it can be deleted at any time without loss of information.

Listing 1 shows an example Simfile (this was made for the example shown in the earlier examples (Figure 1, page 11). When the function from the VS Solver library reads a Simfile, the only keywords recognized are `SIMFILE`, `SET_MACRO`, `FILEBASE`, `INPUT`, `INPUTARCHIVE`,

ECHO, FINAL, LOGFILE, ERDFILE, DATADIR, GUI_REFRESH_V, VEHICLE_CODE, and END. Other keywords are written for possible use by other programs (e.g., the VS S-Function wrapper program used in Simulink models).

Listing 1. Contents of an example Simfile.

```
SIMFILE

SET_MACRO $(ROOT_FILE_NAME)$ Run_f0b02a7c-2939-4577-87ec-8ed10614a213
SET_MACRO $(OUTPUT_PATH)$ Results
SET_MACRO $(WORK_DIR)$ .\
SET_MACRO $(OUTPUT_FILE_PREFIX)$ Results\Run_f0b02a7c-2939-4577-87ec-
8ed10614a213\LastRun

FILEBASE $(OUTPUT_FILE_PREFIX)$
INPUT Results\$(ROOT_FILE_NAME)$\Run_all.par
INPUTARCHIVE $(OUTPUT_FILE_PREFIX)$_all.par
ECHO $(OUTPUT_FILE_PREFIX)$_echo.par
FINAL $(OUTPUT_FILE_PREFIX)$_end.par
LOGFILE $(OUTPUT_FILE_PREFIX)$_log.txt
ERDFILE $(OUTPUT_FILE_PREFIX)$_erd
PROGDIR Z:\Current\CarSim_2018\CarSim_Prog\
DATADIR Z:\Current\CarSim_2018\CarSim_Data\
GUI_REFRESH_V CarSim_RefreshEvent_3340
RESOURCEDIR Z:\Current\CarSim_2018\CarSim_Prog\Resources\
PRODUCT_ID CarSim
PRODUCT_VER 2018.0
ANIFILE Z:\Current\CarSim_2018\CarSim_Data\runs\animator.par
EXT_MODEL_STEP 0.00050000
PORTS_IMP 0
PORTS_EXP 0

DLLFILE Z:\Current\CarSim_2018\CarSim_Prog\Programs\solvers\carsim_32.dll
END
```

The VS Math Model deals with only one file of each type. For example, if a Simfile specifies several input file names using the **INPUT** keyword, the current pathname is updated each time a new line is read that contains the **INPUT** keyword. The last name specified in the Simfile is the only one used. If you want to have the VS Math Model to read from several input files you should create a top-level Parsfile that in turn specifies other Parsfiles with the datasets of interest, as described in the next section. The optional **INPUTARCHIVE** parameter will create a copy of the input file for keeping alongside the outputs.

Simfiles generated by the VS Browser always specify Input Archive, Echo, and End files (keywords **INPUTARCHIVE**, **ECHO**, and **FINAL**). However, these files are optional from the point of view of the VS Math Model. If you are running VS Math Models from other software, you may omit any of these parameters.

Macros can be used in Simfiles, which start with “\$(” and end with “)\$”. Several macros are defined toward the top of the file with **SET_MACRO** and used beneath.

There are also built-in macros available for use in the Simfile that are evaluated and substituted by the VS Math Model (Table 4).

Table 4. Built-in Macro keywords available for use in the VS Math Models.

Macro Keyword	Description
TIMESTAMP	The system time in the format: YYYY-MM-DD_HH.MM.SS.
TIMESTAMP_ISO8601	The system time represented in ISO 8601 basic format: YYYYMMDDTHHMMSS.SSSZ. The T indicates where the time of day begins, and the Z at the end indicates it is a UTC timestamp.
PROCESS_ID	The integer process ID of the program running the solver.
THREAD_ID	The integer thread ID of the running solver.
SIMFILE_FILENAME_RAW	The Simfile name that was passed to the solver.
SIMFILE_FILENAME_LOCATION	The full path of the folder containing the Simfile.
SIMFILE_FILENAME_ABSOLUTE	The absolute pathname of the Simfile.
SIMFILE_FILENAME_BASE	The base Simfile name, without path and extension.
SIMFILE_FILENAME_EXTENSION	The extension of the Simfile, after the dot.

The VS Browser makes extensive use of macros to enable the accumulation of multiple runs. For example, Figure 14 shows a **Run Control** dataset where the option is selected to keep multiple simulations (2). The five most recent runs all have names based on a timestamp with the date and time (3); the most recent has the date 2016-10-20 (4). All of the files are in a folder that is named to match the dataset for the run (1). Listing 2 shows the contents of the Simfile used to generate the files in this example. The macro **OUTPUT_FILE_PREFIX** is set using the **TIMESTAMP** macro, causing the VS Math Model to generate a unique base name every time a new simulation is run.

Listing 2. Simfile used to generate file names using the current date and time.

```

SIMFILE

SET_MACRO $(ROOT_FILE_NAME)$ Run_1078cca4-7e91-46cf-aef4-b668c96ab733
SET_MACRO $(OUTPUT_PATH)$ Z:\Current\CarSim_2017_Beta\CarSim_Data\Results
SET_MACRO $(WORK_DIR)$ Z:\Current\CarSim_2017_Beta\CarSim_Data\
SET_MACRO $(OUTPUT_FILE_PREFIX)$ $(OUTPUT_PATH)\$(ROOT_FILE_NAME)\$(TIMESTAMP)$

FILEBASE $(OUTPUT_FILE_PREFIX)$
INPUT $(WORK_DIR)\Results\$(ROOT_FILE_NAME)\Run_all.par
INPUTARCHIVE $(OUTPUT_FILE_PREFIX)_all.par
ECHO $(OUTPUT_FILE_PREFIX)_echo.par
FINAL $(OUTPUT_FILE_PREFIX)_end.par
LOGFILE $(OUTPUT_FILE_PREFIX)_log.txt
ERDFILE $(OUTPUT_FILE_PREFIX).erd
PROGDIR Z:\Current\CarSim_2017_Beta\CarSim_Prog\
...
END

```

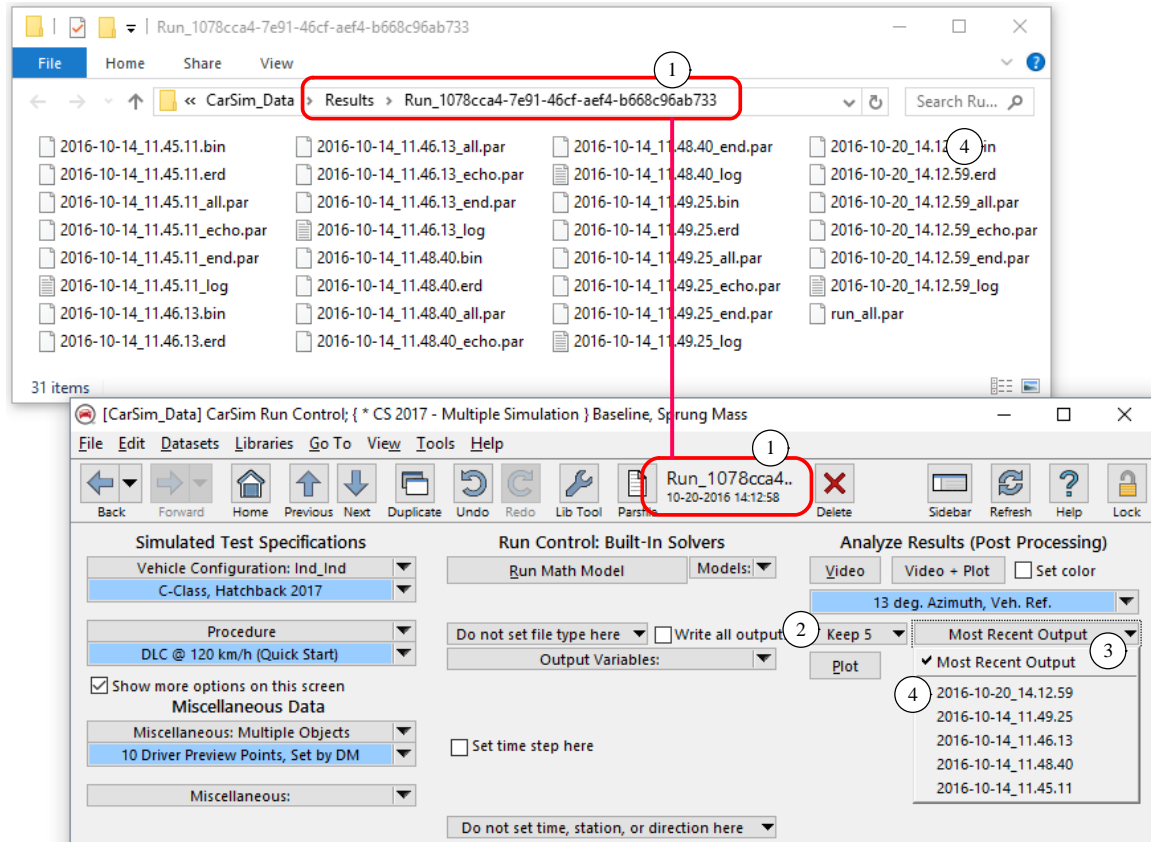



Figure 14. Multiple files made from a single Run Control dataset, using time stamp naming.

If you're generating your own Simfiles you do not have to use macros — you can specify your own names. Of course, you can also use any of the built-in keywords shown in Table 4.

3. Configuring the Math Model

The VS Math Model is often constructed by reading a Simfile and the `MODEL_LAYOUT` command in a Parsfile. The Simfile specifies the names of input and output text files and sometimes additional information for RT systems. All other information comes from the text Parsfiles.

Note SuspensionSim does not use a `MODEL_LAYOUT` command. All models are built using the fundamental SuspensionSim commands.

Reading a line from a Parsfile

Most VS Math Model parameters and state variables can be assigned new values at runtime, just before the simulation starts. Each line of the Parsfile is read and scanned to see if it begins with a keyword associated with a variable. If there is a recognized keyword, more information is taken from the line of text, and possibly, the following lines in the file (as in the case of tabular data).

Note Variables that are internal to a VS Math Model might be identified in the context of the VS Math Model as parameters, state variables, output variables, etc. When descriptions in this document refer to a *variable*, the assumption is that the description is generic with respect to whether the variable represents a parameter, a state variable, an output variable, etc.

A line of data that sets a value for a variable in the VS Math Model has the syntax:

keyword [=] *expression* [; [[UNITS] [=] *units*] [; [*description*]]]

where *keyword* is the name of a variable, *expression* is either a number or an algebraic expression, *units* is an optional keyword used to specify the units associated with the variable specified by *keyword*, and *description* is a new text description of the variable.

In the case of an indexed parameter, the syntax may be extended as:

keyword [*index_list*] [=] *expression* [; [[UNITS] [=] *units*] [; [*description*]]]

where *index_list* has the form:

(*index* *[, *index*])

where each *index* is evaluated to obtain a number. If there are multiple indices, they are separated by commas. Usually, the indices are numbers, e.g., `A_KPI(1,1)`.

Lines of data in this form are assignment statements and are processed with a generic VS Assignment Command if *keyword* (possibly with an index-list) is recognized as the name of a variable. In this case the assignment command potentially performs three actions:

1. Assign a value to the identified variable.
2. Optionally change the units for the variable.

3. Optionally change the text description for the variable that will be printed in Echo files and other documentation.

Every variable installed in the VS Math Model already has a value, assigned units, and a text description. The units and description are optional; most lines of input from a Parsfile exist only to change the values of existing variable. The option to change units is covered in the next section. The option to change the descriptive text is not common in basic use of the software and is not covered further in this document (for more information about setting descriptions, please see *VS Commands Manual*).

As noted earlier, if a comment character ‘!’ is found anywhere in the input line of text, all text starting from that position is removed and will not be processed by the VS Math Model.

Each VS Math Model has an internal database of keywords that it recognizes. As noted earlier, a keyword is an alphanumeric expression that cannot contain spaces, tabs, control (invisible) characters, the semicolon ‘;’, the exclamation mark ‘!’, or three consecutive periods ‘...’. Keywords in a Parsfile are never case sensitive.

If *keyword* is not recognized, the VS Math Model simply skips the line of input.

Note	Lines that begin with unrecognized keywords are ignored to allow the same Parsfile to be used with VS Visualizer and the VS Browser, and possibly alternate VS Math Models.
-------------	---

You can add new variables and associated keywords to the internal database of the VS Math Model using VS Commands as described in *VS Commands Manual* and API functions as described in *VehicleSim API Manual*.

The Echo file is the main reference document for the list of keywords recognized by an assembled VS Math Model. When viewing an Echo file, you will see that most lines involving a parameter use the syntax shown above, with unneeded delimiters (‘=’ and the second ‘;’) omitted along with the `UNITS` keyword for brevity. For example, nearly all VS Math Models have a time step parameter identified in an Echo file with a line such as:

```
TSTEP    0.0005 ; s ! Time step for numerical integration [L]
```

where the keyword is "TSTEP", the expression is the number 0.0005, and the units are seconds.

If the text description is to be changed, then the comment indicator ‘!’ would be replaced with a ‘;’ to indicate that the descriptive text is customized (and to use the description if the Echo file is used to repeat or continue the run).

Most inputs in a typical dataset just assign a number, e.g.,

```
TSTEP 0.0005
```

The value assigned to the parameter can be a number, an algebraic expression involving numbers (e.g., $1/2000$), or an algebraic expression involving numbers and symbolic names of parameters and variables.

Note	When a keyword associated with a parameter or variable appears in an algebraic expression or the right-hand side of the (optional) '=' sign, it is processed by the VS Math Model as a function that provides the current value of the linked internal variable. When it appears on the left-hand side, the generic VS Assignment Command checks to see if it is linked to an internal variable; if it is, the line of input is processed to set the value and possibly units and description.
-------------	--

VS Commands Manual describes more advanced features, such as the use of VS symbolic expressions and the specification of custom text descriptions.

Unit Systems: User Display Units and Internal SI Units

A VS Math Model maintains two major sets of units:

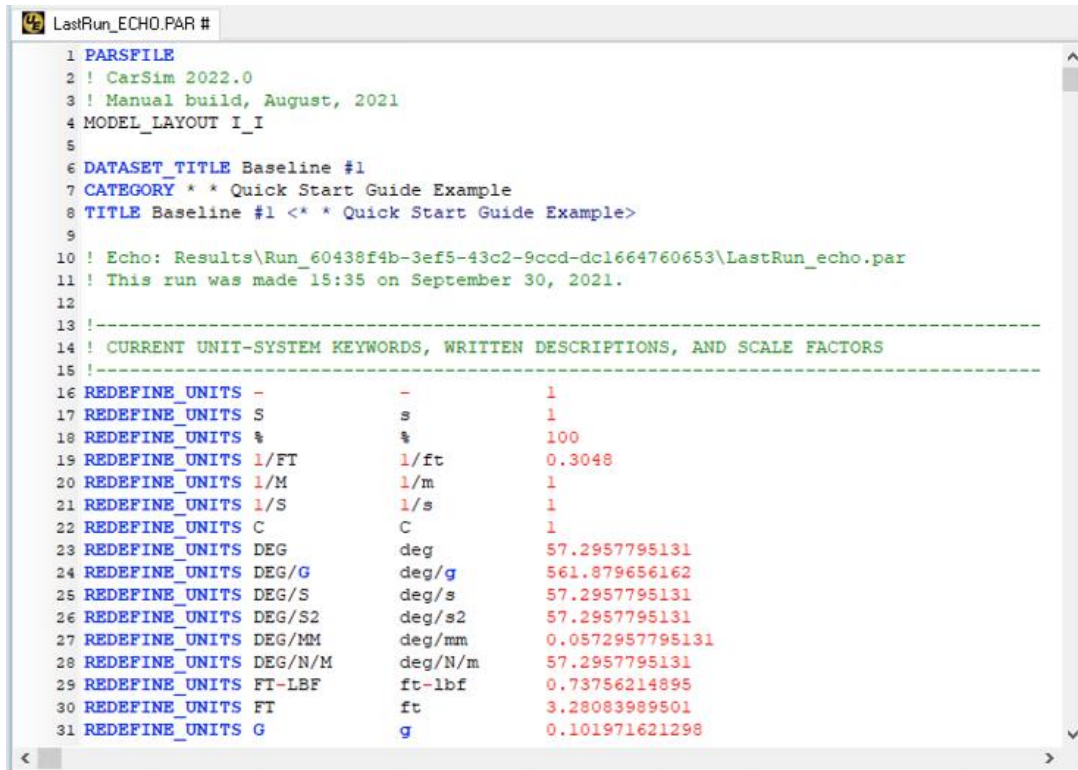
1. All calculations during a run assume all user input parameters and variables are scaled according to the *International System of Units* (SI). This means that no scale factors are needed in the internal equations of motion. Length and distance variables use meters, mass variables use kilograms, force variables use Newtons, angular variables use radians, acceleration variables use m/s^2 , pressure variables use Pascals, etc.
2. *User display units* are accepted as inputs and shown for outputs. All machine-generated documentation files (Echo files, etc.) show values of variables and parameters using user display units. The VS Math Model automatically scales values as needed to convert to and from internal SI units during input and output.

VS Math Models support user display units to allow inputs to be provided with convenient units such as mm, degrees, g, etc. Furthermore, new display units can be provided at runtime (e.g., feet, inches, etc.) to support inputs with unit systems that are not built in. The handling of units is as follows:

- All calculations are done with internal SI units.
- Any numerical values read from files or imported from other software (e.g., Simulink) are assumed to have user display units; therefore, they are divided by a gain associated with the user display units at the time they are read or imported.
- Any numerical values with internal SI units written to file or exported to other software are multiplied by the gain associated with the user display units at the time they are written or exported.

For example, consider a wheelbase parameter `LX_AXLE` (longitudinal axle location) with user display units of millimeters. The value provided is 3000 (mm). When read by the VS Math Model, it is divided by 1000 such that the value used in all calculations during the simulation is 3.000 m. When written in an Echo file for user documentation, the internal value of 3.000 is multiplied by 1000 to write the value 3000 mm.

The Echo file can display the complete set of units used for any run. To specify this option, enter `OPT_ECHO_ALL_UNITS = 1` in a miscellaneous data field, and run a simulation. The list of units appears at the beginning of the new Echo file (Figure 15).



```

1 PARSFILE
2 ! CarSim 2022.0
3 ! Manual build, August, 2021
4 MODEL_LAYOUT I_I
5
6 DATASET_TITLE Baseline #1
7 CATEGORY * * Quick Start Guide Example
8 TITLE Baseline #1 <* * Quick Start Guide Example>
9
10 ! Echo: Results\Run_60438f4b-3ef5-43c2-9ccd-dc1664760653\LastRun_echo.par
11 ! This run was made 15:35 on September 30, 2021.
12
13 !-----
14 ! CURRENT UNIT-SYSTEM KEYWORDS, WRITTEN DESCRIPTIONS, AND SCALE FACTORS
15 !-----
16 REDEFINE_UNITS - - 1
17 REDEFINE_UNITS S s 1
18 REDEFINE_UNITS % % 100
19 REDEFINE_UNITS 1/FT 1/ft 0.3048
20 REDEFINE_UNITS 1/M 1/m 1
21 REDEFINE_UNITS 1/S 1/s 1
22 REDEFINE_UNITS C C 1
23 REDEFINE_UNITS DEG deg 57.2957795131
24 REDEFINE_UNITS DEG/G deg/g 561.879656162
25 REDEFINE_UNITS DEG/S deg/s 57.2957795131
26 REDEFINE_UNITS DEG/S2 deg/s2 57.2957795131
27 REDEFINE_UNITS DEG/MM deg/mm 0.0572957795131
28 REDEFINE_UNITS DEG/N/M deg/N/m 57.2957795131
29 REDEFINE_UNITS FT-LBF ft-lbf 0.73756214895
30 REDEFINE_UNITS FT ft 3.28083989501
31 REDEFINE_UNITS G g 0.101971621298

```

Figure 15. An Echo file can optionally show all units definitions.

When a variable is assigned a numerical value, the value should be expressed in the user display units associated with the variable. The VS Math Model will perform conversions as needed. For example, the 8° kingpin inclination angle is automatically converted by the solver to 0.139626 radian for use in the internal equations. On the other hand, if a math expression is used to assign a value, a unit conversion may or may not be applied depending on the content of the math expression. *VS Commands Manual* describes the conditions that cause the units conversion to be skipped. (*VS Commands Manual* deals with more advanced inputs such as symbolic expressions that can be used to calculate a value for one parameter based on current values of other parameters.)

The line of input used to specify a value for a variable can also be used to specify the units. For example, all of the following lines of input have exactly the same effect with respect to the units and value of the indexed parameter `A_KPI(1,1)` :

```

A_KPI(1,1) 8
A_KPI(1,1) = 8 ;
A_KPI(1,1) = 8 ! Here is a comment. The units are not changed.
A_KPI(1,1) = 8 ; units = deg ! Units are explicitly degrees
A_KPI(1,1) 8 ; deg ! Same thing, more compact
A_KPI(1,1) 8 ; deg ; Reference kingpin inc. angle ! new description
A_KPI(1,1) 8 ;; Reference kingpin inc. angle ! new description

```

Suppose you want to use different units. The keyword for dimensionless units (e.g., radians and other ratios) is '-'. Here is how the parameter could be specified in radians:

```
A_KPI(1,1) 0.139626 ; - ! '-' is the symbol for dimensionless
```

When the entry is used to specify units, the units associated with the variable are changed to the specified *units*. If a numerical value is also provided, the associated expression is converted using the new *units*. Because the variable is updated to match *units*, the numerical value shown in the Echo file will use the new units. Later, if the variable is given a new value and units are not specified, then the new units still apply.

If you want to use units that are not already available (e.g., specify a dimension in furlongs), there is a VS Command `DEFINE_UNITS` to define new units that is described in *VS Commands Manual*.

Indexed Parameters

Given that a vehicle has many features that are repeated (axles, tires, etc.), the VS Math Models have parameters that are indexed relative to repeated components in the vehicle.

There are two ways to set values for an indexed parameter:

1. As shown in the previous examples, each indexed parameter in the VS Math Model can be specified using a unique keyword with an associated index list, e.g.:

```
A_KPI(1,1) 8
```

In this case, the two indices indicate that this is for axle 1, side 1 (left).

2. Alternatively, the value can be specified without the index list:

```
A_KPI 8
```

where the value is applied for the “current axle” and “current side.”

Setting the Context for Indexed Parameters

Every indexed parameter in a VS Math Model is associated with *index parameters* that define the “current” object of interest. For example, standard indices are used in CarSim and TruckSim to indicate unit number, axle number, side, payload number, gear number, etc. These indices are identified with index keywords `IUNIT`, `IAXLE`, `ISIDE`, `ILOAD`, and `IGEAR`, respectively.

This allows the same root parameter name to be used for a property that is applied to multiple components, as was seen earlier in the example Parsfile for a tire (Figure 2, page 12). In fact, the Parsfile datasets in all products rely on this method of specifying values for indexed parameters.

In the case of the tire data, the index parameters `IUNIT` (vehicle unit number), `IAXLE` (axle number on the specified unit), `ISIDE` (which side of the specified axle), and possible `ITIRE` (if there are duals, which tire on the wheel: inner or outer) are specified in other Parsfiles. They define the context before the tire Parsfile is read by the VS Math Model.

The VS Browsers for each product are designed to set the context for indexed parameters in a component, such as a sprung mass, a suspension, a tire, etc. For example, the main vehicle index keywords in the CarSim and TruckSim Browsers are set as follows:

1. IVEHICLE is set to 0 near the top of all **Run Control** Parsfiles. It is incremented at the top of the Parsfiles for any lead unit (e.g., **Vehicle Assembly** in CarSim).
2. IUNIT is also set to 0 near the top of all **Run Control** Parsfiles. It is incremented at the top of the Parsfiles for the **Vehicle Assembly** library (in CarSim) or any of the Trailer libraries. Therefore, it has a value of 1 for the first vehicle unit, a value of 2 for the second, etc.
3. IAXLE is set in the Parsfile for a vehicle unit screen to 1 before the link to axle 1 data (suspension, brakes, tires, etc.), then 2 for the link to axle 2 data, etc. IAXLE is reset to start again at 1 for each vehicle unit.
4. ISIDE is set to 1 or 2 within the Parsfiles for axle sub-systems that have separate parameters or tables for the left and right sides.

For example, Figure 16 shows part of a Parsfile for a front **Suspension: Springs, Dampers, and Compliance** dataset in CarSim. The values for IVEHICLE and IUNIT were incremented in the **Vehicle Assembly** dataset for the sedan, which also set IAXLE to 1 before the link to the Suspension dataset shown in the figure. Notice that the Suspension data sets ISIDE to 1 for the left side (line 37) before lines of text that assign values using keywords OPT_EXT_SP, FS_COMP_COEFFICIENT, etc. Two more Parsfiles are read in the same context for shock absorber data (line 49) and Jounce and Rebound stops (line 53).

Later in the file, the ISIDE index is set to 2 (line 68), such that any of the following settings will apply to parameters and tables on the right side.

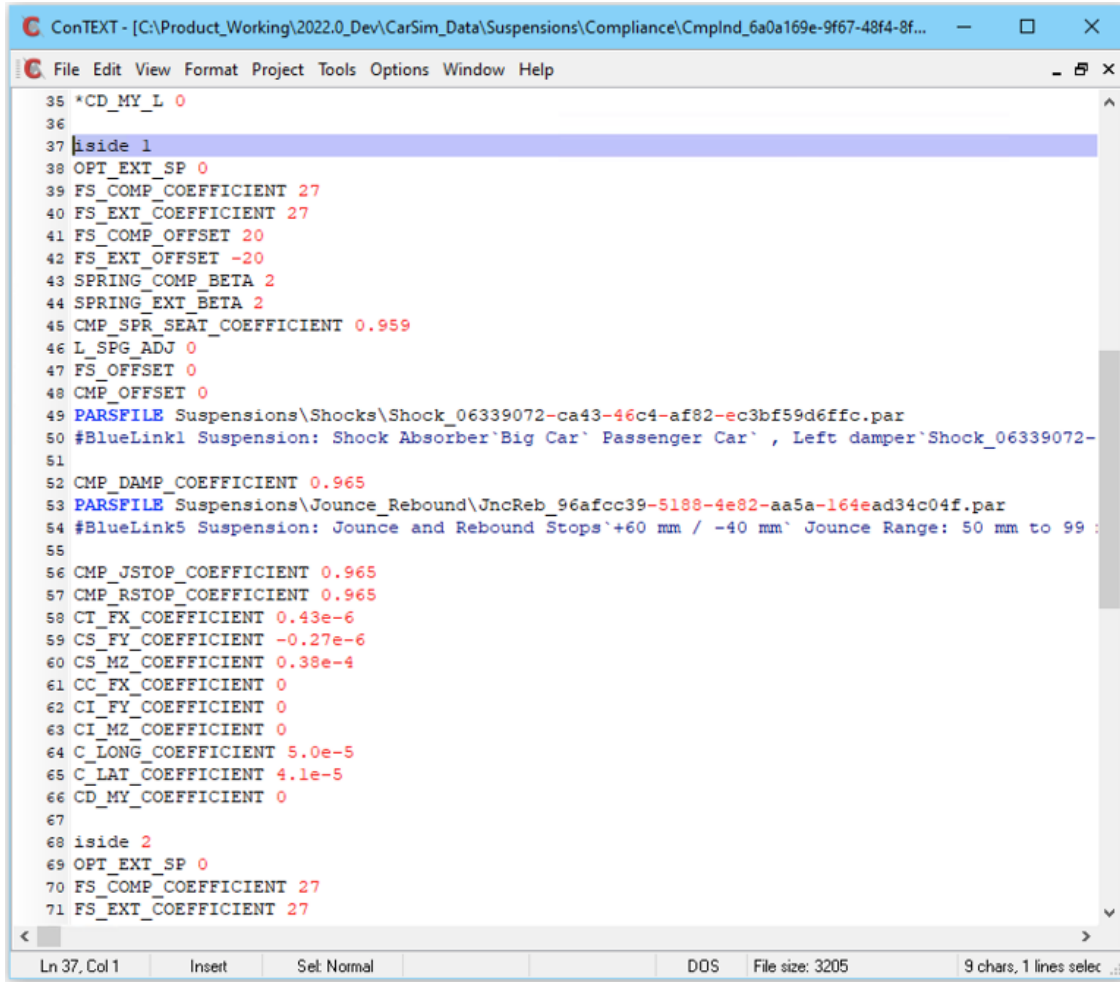
To see these index parameters and other hidden keywords, make a run with OPT_ECHO_ALL_PARS set to 1. This causes the VS Math Model to generate an Echo file that includes all the hidden keywords. For example, Figure 17 shows the section of the Echo file where values of OPT_EXT_SP are shown for two wheels in CarSim (lines 339 and 342). The preceding comments document the roles of (hidden) index parameters IAXLE and ISIDE.

Indexing Dimension Changes Due to Model Options

Notice that the comments in Figure 17 make no mention of the indices IUNIT and IVEHICLE. The IUNIT index is used only if the simulation has two or more units. In this example, the single vehicle was a sedan with no trailer, so there is only one vehicle and one unit. Had there been a trailer, the parameters with root name OPT_EXT_SP would have had three indices: unit, axle, side.

All wheels in TruckSim support dual tires. Accordingly, tire parameters and tables in TruckSim have an additional index ITIRE, set to 1 or 2 to indicate an inner or outer tire, respectively.

CarSim also supports dual tires as an option. Therefore, in CarSim, a tire parameter might have four indices (unit, axle, side, inner-outer) if duals are enabled and there is a trailer. On the other hand, the tire parameters will only have two indices (axle and side) if there is no trailer and duals are not enabled.

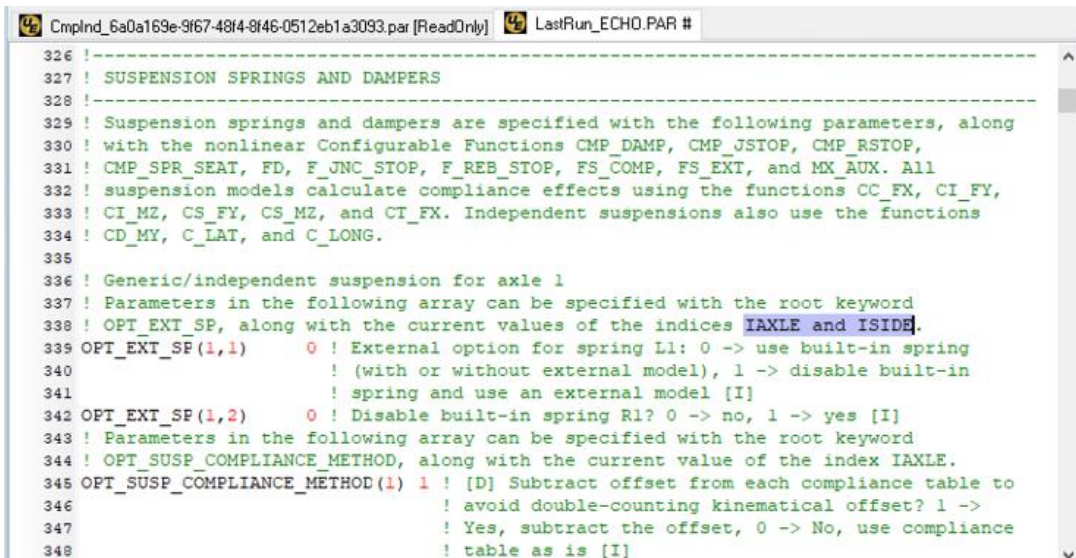


```

35 *CD_MY_L 0
36
37 iside 1
38 OPT_EXT_SP 0
39 FS_COMP_COEFFICIENT 27
40 FS_EXT_COEFFICIENT 27
41 FS_COMP_OFFSET 20
42 FS_EXT_OFFSET -20
43 SPRING_COMP_BETA 2
44 SPRING_EXT_BETA 2
45 CMP_SPR_SEAT_COEFFICIENT 0.959
46 L_SPG_ADJ 0
47 FS_OFFSET 0
48 CMP_OFFSET 0
49 PARSPFILE Suspensions\Shocks\Shock_06339072-ca43-46c4-af82-ec3bf59d6ffc.par
50 #BlueLink1 Suspension: Shock Absorber`Big Car` Passenger Car` , Left damper`Shock_06339072-
51
52 CMP_DAMP_COEFFICIENT 0.965
53 PARSPFILE Suspensions\Jounce_Rebound\JncReb_96afcc39-5188-4e82-aa5a-164ead34c04f.par
54 #BlueLink5 Suspension: Jounce and Rebound Stops`+60 mm / -40 mm` Jounce Range: 50 mm to 99
55
56 CMP_JSTOP_COEFFICIENT 0.965
57 CMP_RSTOP_COEFFICIENT 0.965
58 CT_FX_COEFFICIENT 0.43e-6
59 CS_FY_COEFFICIENT -0.27e-6
60 CS_MZ_COEFFICIENT 0.38e-4
61 CC_FX_COEFFICIENT 0
62 CI_FY_COEFFICIENT 0
63 CI_MZ_COEFFICIENT 0
64 C_LONG_COEFFICIENT 5.0e-5
65 C_LAT_COEFFICIENT 4.1e-5
66 CD_MY_COEFFICIENT 0
67
68 iside 2
69 OPT_EXT_SP 0
70 FS_COMP_COEFFICIENT 27
71 FS_EXT_COEFFICIENT 27

```

Figure 16. Parsfile for a CarSim dataset for suspension springs, dampers, and compliance.



```

326 !-----
327 ! SUSPENSION SPRINGS AND DAMPERS
328 !-----
329 ! Suspension springs and dampers are specified with the following parameters, along
330 ! with the nonlinear Configurable Functions CMP_DAMP, CMP_JSTOP, CMP_RSTOP,
331 ! CMP_SPR_SEAT, FD, F_JNC_STOP, F_REB_STOP, FS_COMP, FS_EXT, and MX_AUX. All
332 ! suspension models calculate compliance effects using the functions CC_FX, CI_FY,
333 ! CI_MZ, CS_FY, CS_MZ, and CT_FX. Independent suspensions also use the functions
334 ! CD_MY, C_LAT, and C_LONG.
335
336 ! Generic/independent suspension for axle 1
337 ! Parameters in the following array can be specified with the root keyword
338 ! OPT_EXT_SP, along with the current values of the indices IAXLE and ISIDE.
339 OPT_EXT_SP(1,1) 0 ! External option for spring 1: 0 -> use built-in spring
340 ! (with or without external model), 1 -> disable built-in
341 ! spring and use an external model [I]
342 OPT_EXT_SP(1,2) 0 ! Disable built-in spring R1? 0 -> no, 1 -> yes [I]
343 ! Parameters in the following array can be specified with the root keyword
344 ! OPT_SUSP_COMPLIANCE_METHOD, along with the current value of the index IAXLE.
345 OPT_SUSP_COMPLIANCE_METHOD(1) 1 ! [D] Subtract offset from each compliance table to
346 ! avoid double-counting kinematical offset? 1 ->
347 ! Yes, subtract the offset, 0 -> No, use compliance
348 ! table as is [I]

```

Figure 17. Part of Echo file generated with OPT_ECHO_ALL_PARS set to 1.

Indexed Parameters for Optional Parts in the Model

The VS Math Models might include optional parts that were added with `INSTALL` or `DEFINE` commands.

In `SuspensionSim`, nearly the whole model is added with commands such as `SS_DEFINE_POINTS`, `SS_DEFINE_JOINTS`, etc. There are index parameters such as `IPOINT` and `IJOINT` that are used to specify the current part of interest. The `SuspensionSim` GUI is designed such that these index parameters are typically set to the most recently added part. For example, if Point #15 was just added, `IPOINT` is set to 15, and root names for point parameters such as coordinates `SS_PT_X`, `SS_PT_Y`, etc. will apply for Point 15. All parameter root names for a point will apply for point 15, until `IPOINT` is changed; then the root names will all apply to the newly identified point.

The vehicle models in `CarSim`, `TruckSim`, and `BikeSim` have many parameters that are defined automatically at the start of the run by the `MODEL_LAYOUT` command. This command sets the number of units, number of axles on each unit, and so on. As noted earlier, the VS Browser screens are designed such that index parameters such as `IAXLE` and `IUNIT` are set outside the scope of an individual screen Parsfile, allowing a dataset to be used for any axle on any vehicle unit.

Other commands are used to add parts that are optional, such as payloads, sensors, and moving objects. For example, the command `DEFINE_MOVING_OBJECTS` will add a specified number of moving objects to the model. At the completion of the command, an index parameter `IOBJECT` is set to the number of the last one added. That is, the last object is the “current one,” and all parameters for that object are typically specified using the root keywords.

Configurable Functions (Tables)

Many of the equations in a VS Math Model involve algebraic relationships used to calculate a variable from values of one or two other variables in the model. These relationships are represented with *Configurable Functions* that can be set at runtime to use various calculation methods, such as table lookup, linear coefficients, constants, ore custom user-defined formulas. About half of the calculations performed in a simulation run are typically taken up by Configurable Functions.

These functions are configured with sets of commands, parameters, and possibly tables with columns and rows of numbers. All involve the calculation of a dependent variable from at least one independent variable; some support the calculation of the dependent variable from two independent variables. Calculations involving one independent variable are sometimes called 1D functions and those involving two independent variables are called 2D functions.

About Parameters and Commands for Configurable Functions

Most lines of input in a Parsfile begin with a keyword that is interpreted by the VS Math Model as it reads the Parsfile. When the VS Math Model reads a keyword at the beginning of a line of text, there are three cases:

1. The keyword identifies a parameter or variable in the VS Math Model, and whatever follows the keyword is treated as a number (or formula that is evaluated to provide a number) that is assigned to the parameter or variable.

2. Or, the keyword identifies another VS Command supported by the VS Math Model. The command causes the VS Math Model to do something, which might be simple or might be complicated. In the case of a Configurable Function, it might continue to read from the Parsfile to obtain tabular data.
3. Or, the keyword is not recognized. In this case, the VS Math Model proceeds to read the next line from the Parsfile.

If the generic VS Assignment Command recognizes a keyword as a parameter or variable, it changes the value (and possibly units and description) for the identified parameter or variable but does nothing else. As noted earlier, it's OK to assign a new value to an existing parameter or variable many times. The value is not used until the VS Math Model performs an initialization. On the other hand, other commands cause the VS Math Model to do something immediately, and possibly change the way it will respond to keywords in following lines of input. These are called VS Extended Assignment Commands. Many of the keywords described in this section are for VS Extended Assignment Commands that perform several actions when applied, as will be shown shortly.

Note	Documentation for most of the VS Commands supported by a VS Math Model is beyond the scope of this reference Manual. Please see <i>VS Commands Manual</i> for details about commands that are not related to using existing Configurable Functions.
-------------	---

Calculating the Value of a Configurable Function from One Variable

There are four options available to calculate the dependent variable F from a single independent variable X (defined in a later section, page 52):

1. Constant: $F = \text{constant}$
2. Linear relationship: $F = \text{coefficient} \cdot X$
3. 1D table-lookup: $F = f_1(X)$
4. Custom user-defined equation: $F = f_1(X)$

The table-lookup option has many variations involving interpolation and extrapolation calculation details. Table 5 shows commands used to specify *constant*, *coefficient*, and calculation options for the table-lookup function f_1 for the specific example of a shock absorber.

Each command type listed in Table 5 is a VS Extended Assignment Command that has two effects:

1. It sets the method of calculation. For example, the command `FD_COEFFICIENT` sets the calculation method for the FD function to use equation 2 above. The command `FD_TABLE` sets the calculation method to use equation 3.
2. It provides the essential data needed to apply the calculation method. For example, the command `FD_COEFFICIENT 100` not only sets the calculation method, it gets a value for the coefficient needed to apply equation 2.

Table 5. Configurable function commands and options when there is one independent variable.

Command Suffix	2nd Keyword	Example: FD	Calculation Method
<code>_CONSTANT</code>	<i>expression</i>	<code>FD_CONSTANT 0.0</code>	Constant
<code>_COEFFICIENT</code>	<i>expression</i>	<code>FD_COEFFICIENT 100.0</code>	Coefficient multiplied by independent variable
<code>_TABLE</code>	<code>STEP</code>	<code>FD_TABLE STEP</code>	Table lookup using one independent variable.
	<code>STEP_LOOP</code>	<code>FD_TABLE STEP_LOOP</code>	
	<code>LINEAR</code>	<code>FD_TABLE LINEAR</code>	
	<code>LINEAR_LOOP</code>	<code>FD_TABLE LINEAR_LOOP</code>	
	<code>LINEAR_FLAT</code>	<code>FD_TABLE LINEAR_FLAT</code>	
	<code>SPLINE</code>	<code>FD_TABLE SPLINE</code>	
	<code>SPLINE_LOOP</code>	<code>FD_TABLE SPLINE_LOOP</code>	
	<code>SPLINE_FLAT</code>	<code>FD_TABLE SPLINE_FLAT</code>	
<code>_EQUATION</code>	<i>formula</i>	<code>FD_EQUATION 250*X^3</code>	Evaluation of formula

The `_CONSTANT` and `_COEFFICIENT` types of commands only need one numerical value for Equations 1 and 2, respectively, and that value is simply provided on the same line of text.

When the suffix of a command is `_TABLE`, the VS Math Model checks the rest of the line to see if the type of table is specified with a second keyword. The table type determines the interpolation method that will be used to calculate the output variable from the tabular data. If a second keyword is provided, the type of the table is set accordingly. If not, the VS Math Model uses a default option built into the VS Math Model. The command then causes the VS Math Model to continue reading from the Parsfile to obtain tabular data, reading until the keyword `ENDTABLE` is found.

Data for tables are shown with a sequence of lines of text:

```
keyword [specification]
<numerical data>
...
ENDTABLE
```

The calculation options are typically set using the GUI of the VehicleSim product. For example, in Figure 18, the selected method (spline interpolation & extrapolation) causes the GUI to show a spreadsheet for entering tabular data and generates the appropriate keywords in the Parsfile (`FD_TABLE` and `SPLINE`). The lower part of the GUI screen has a note indicating that more information is available in an Echo file, and that you can find the section of interest by searching for the root keyword “FD.” The Echo file (Figure 19, line 1546) shows the *keyword* is `FD_TABLE (1, 1)`, the *specification* is `SPLINE`, and there is a comment following *specification*.

Tabular data must be sorted with values of the independent variable(s) in increasing order. In this case, a single independent variable (compression speed) is given a value in each row of the table, followed by the associated value of the output variable (damper force).

The contents of a table must be numbers, not symbols; VS symbolic processing is not applied to these lines of input.

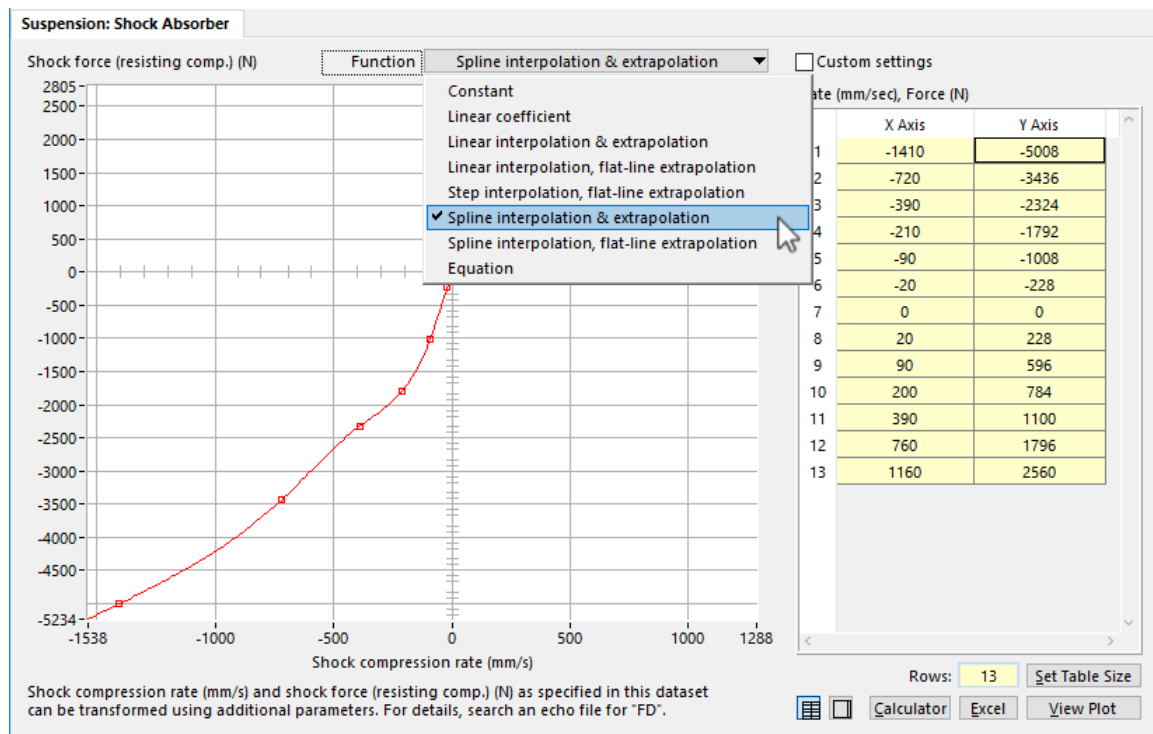


Figure 18. CarSim screen for shock absorber data.

```

ConTEXT - [Z:\Current\CarSim_2018.1\CarSim_Data\Results\Run_963baaf9-0eae-44ca-b359-c11a...
File Edit View Project Tools Options Window Help
1537 ! FD: Suspension damper force (1 side). Damper force is a function of compression
1538 ! speed (CONSTANT, COEFFICIENT, or TABLE). Alternatively, a custom equation can be
1539 ! defined at runtime. Damper force from the calculation can be adjusted with FD_GAIN
1540 ! and FD_OFFSET. Compression speed used in the calculation can be adjusted with
1541 ! CMP_R_SCALE_FD and CMP_R_START_FD. This configurable function supports 4 datasets;
1542 ! if indices shown below are not used, e.g., (1,2), the current values of the indices
1543 ! IAXLE and ISIDE are used to identify the dataset when reading data.
1544
1545 ! 1D table: col 1 = compression speed (mm/s), col 2 = damper force (N)
1546 FD_TABLE(1,1) SPLINE ! axle 1, left side
1547 -1410, -5008
1548 -720, -3436
1549 -390, -2324
1550 -210, -1792
1551 -90, -1008
1552 -20, -228
1553 0, 0
1554 20, 228
1555 90, 596
1556 200, 784
1557 390, 1100
1558 760, 1796
1559 1160, 2560
1560 ENDTABLE
Ln 1541, Col 38 Insert Sel: Normal DOS File size: 277019

```

Figure 19. Data for a table from a group, specified with a unique keyword.

The `_EQUATION` suffix indicates that a formula will be provided that involves a dummy variable `X`. For the shock absorber example, `X` would be the compression rate. When an equation is used, all units are internal. Therefore, the formula should produce force in Newtons as a function of `X` in m/s.

The `_EQUATION` option is not available for Configurable Functions if the VS Math Model uses the inverse or derivative internally. For example, the comments for the `ROAD_DZ` function indicate that the derivative is used internally (see lines 4736 and 4737, Figure 21, page 48). Therefore, the `_EQUATION` option is not available for this function. For information about using the `_EQUATION` option to add a symbolic equation, please see *VS Commands Manual*.

Examples for each type of calculation method are shown in the document *VS Browser (GUI and Database)*.

Indexing of Configurable Function Datasets

Many Configurable Functions have multiple datasets that are applied in several places in the VS Math Model. The syntax is like that used for parameter arrays. As shown by the highlighted text in Figure 19, comments preceding the Configurable Function data specify the number of datasets that can be used (e.g., 4) and names of index variables (e.g., `IAXLE`, and `ISIDE`).

As with indexed parameters, datasets for Configurable Functions that are used for different components in the VS Math Model can be identified with a keyword followed by an index list, e.g., `FD_TABLE (1, 1)`, or with a root keyword, e.g., `FD_TABLE`, within the context of the current values of index variables, e.g., `IAXLE`, and `ISIDE`. For example, the shock absorber table from Figure 19 could also be specified with the lines of text shown in Listing 3 (edited for length).

Listing 3. Data for a Configurable Function from a group, specified with a group keyword.

```
IAXLE 1
ISIDE 1
FD_TABLE SPLINE
-1410, -5008
-720, -3436
...
1160, 2560
ENDTABLE
```

User IDs and Optional Datasets

Most Configurable Functions are associated with a specific part of the VS Math Model, such as a vehicle tire, suspension, steering system, steering controller, etc. On the other hand, multiple instances of some parts may be added as needed in some scenarios, including moving objects, road surfaces, etc. Some of these parts make use of Configurable Functions, requiring that those Configurable Functions support datasets for every part that has been added. For example, the Configurable Function `SPEED_TARGET` defines a target speed as a function of time and location. It is used to provide a target for the speed controller available in CarSim, TruckSim, and BikeSim. It may also be used in ADAS scenarios to control the speed of traffic vehicles represented with VS Moving Objects. As shown in the comments in the Echo file, it supports up to 200 datasets (line 5568, Figure 20). It also includes a user-defined ID identified with the keyword `SPEED_TARGET_ID` (lines 5571 and 5587).

```

5559
5560 ! SPEED_TARGET: Speed controller target. Speed can be a nonlinear CARPET function of
5561 ! station and time or a function of time (CONSTANT, COEFFICIENT, or TABLE) combined
5562 ! with a function SPEED_TARGET_S of station (CONSTANT, COEFFICIENT, or TABLE).
5563 ! Alternatively, a custom equation can be defined at runtime. Speed from the
5564 ! calculation can be adjusted with SPEED_TARGET_GAIN and SPEED_TARGET_OFFSET. Time
5565 ! used in the calculation can be adjusted with TSCALE_SPEED_TARGET and
5566 ! TSTART_SPEED_TARGET. Station used in the calculation can be adjusted with
5567 ! SSCALE_SPEED_TARGET and SSTART_SPEED_TARGET. This configurable function supports
5568 ! 200 datasets; if indices shown below are not used, e.g., (2), the current value of
5569 ! the index ISPEED is used to identify the dataset when reading data.
5570
5571 SPEED_TARGET_ID(1) 2001 ! Need to Stop
5572 SPEED_TARGET_CONSTANT(1) 0 ; km/h ! Constant speed component due to time
5573 SPEED_TARGET_GAIN(1) 1 ! Gain multiplied with calculated value to get speed
5574 SPEED_TARGET_OFFSET(1) 0 ; km/h ! Offset added (after gain) to get speed
5575 SPEED_TARGET_COMBINE(1) ADD ! How to combine the two components
5576 SET_UNITS SPEED_TARGET_S_TABLE(1) km/h ;
5577
5578 ! ID table: col 1 = station (m), col 2 = speed component due to station (km/h)
5579 SPEED_TARGET_S_TABLE(1) STEP ! step interpolation, flat-line extrapolation
5580 -1, 3.6
5581 0, 0
5582 ENDTABLE
5583 SSTART_SPEED_TARGET(1) 0 ; m ! Offset subtracted from station
5584 SSCALE_SPEED_TARGET(1) 1 ! Scale factor divided into (station - SSTART_SPEED_TARGET)
5585 SET_UNITS SPEED_TARGET_CARPET(2) mi/h ;
5586
5587 SPEED_TARGET_ID(2) 2000 ! Path Preview: 0.15/0.2 G Ax/Ay, SPEED_LIMIT_ID, mi/h
5588 SPEED_TARGET_CONSTANT(2) 45 ; mi/h ! Constant speed component due to time

```

Figure 20. Some Configurable Functions include User IDs.

This example has two indices: the internal index shown in parentheses goes from 1 to the number of active Configurable Functions. The keywords shown in lines 5571 – 5584 include “(1)” in their names (index = 1). Lines 5585 – 5588 use “(2)” for index = 2. The second index is user-defined: 2001 (line 5571) and 2000 (line 5587).

Only three of the Configurable Functions in CarSim, TruckSim, and BikeSim have User IDs (Table 6), but these Functions are widely used.

Table 6. Properties of Configurable Functions with User IDs.

Function	No. of Datasets	Index	User ID
LTARG	N_LTARG	ILTARG	LTARG_ID
ROAD_DZ	NROAD_DZ	IROAD_DZ	ROAD_DZ_ID
SPEED_TARGET	N_SPEED_TARGET	ISPEED	SPEED_TARGET_ID

Note Three other special functions have user IDs: VS Paths, VS Roads, and VS XY table segments. These are described in the Help document *Paths and Road Surfaces*.

The purpose of the User ID is to allow a dataset to be reused in different scenarios. For example, moving objects and the speed controller for the ego vehicle have parameters that are set to the SPEED_TARGET_ID parameter for the SPEED_TARGET dataset that will be used.

The default value for a User ID is the internal index. For example, the default value of `SPEED_TARGET_ID(2)` is 2.

Note The User ID concept is supported in the graphical interface for the listed Configurable Functions. The same concept is also used for Paths, Roads, and Path segments in CarSim, TruckSim, and BikeSim.

Calculating the Value of a Configurable Function from Two Variables

Some Configurable Functions can calculate the dependent variable from two independent variables. For example, consider the Configurable Function `ROAD_DZ`, used to calculate an incremental road elevation as a function of longitudinal distance along the road (station) and lateral distance from a reference line.

Given the layout of 2D tabular data, the three variables are identified as:

- *output* variable F , calculated by the function (e.g., incremental elevation of the ground);
- *row* variable X , used to calculate the output (e.g., longitudinal position (station) along the road); and
- *column* variable $Xcol$, also used to calculate the output (e.g., lateral position on the road).

Table 7 lists commands that specify how an output variable is calculated from two independent variables for the example function `ROAD_DZ`.

Table 7. Configurable function commands and calculation methods for 2D tables.

Command Suffix	2nd Keyword	Example: <code>ROAD_DZ</code>	Calculation Method
<code>_CARPET</code>	<code>2D_STEP</code>	<code>ROAD_DZ_CARPET 2D_STEP</code>	Table lookup with both row and column independent variables
	<i>None</i>	<code>ROAD_DZ_CARPET</code>	
	<code>LOOP</code>	<code>ROAD_DZ_CARPET LOOP</code>	
	<code>VAR_WIDTH</code>	<code>ROAD_DZ_CARPET VAR_WIDTH</code>	
	<code>VAR_WIDTH_STEP</code>	<code>ROAD_DZ_CARPET VAR_WIDTH_STEP</code>	
	<code>2D_SPLINE</code>	<code>ROAD_DZ_CARPET 2D_SPLINE</code>	
<code>_COMBINE</code>	<code>MULTIPLY</code>	<code>ROAD_DZ_COMBINE MULTIPLY</code>	Multiply two 1D functions
	<code>ADD</code>	<code>ROAD_DZ_COMBINE ADD</code>	Add two 1D functions
<code>_EQUATION</code>	<i>formula</i>	N/A	Evaluation of formula

Up to four additional methods are available to calculate the output variable F from the row and column variables X and $Xcol$ (defined in a later section, page 52):

1. 2D table-lookup: $F = f_2(Xcol, X)$ — specify this option using a command with the `_CARPET` suffix, followed by lines of text with tabular data.

2. Sum of two single-variable functions: $F = f_r(X) + f_c(Xcol)$ — specify this option using a command with the `_COMBINE` suffix, followed by the keyword `ADD`.
3. Product of two single-variable functions: $F = f_r(X) \cdot f_c(Xcol)$ — specify this option using a command with the `_COMBINE` suffix, followed by the keyword `MULTIPLY`.
4. Custom user-defined equation: $F = f_{2c}(Xcol, X)$ — specify this option with the `_EQUATION` suffix, followed by a formula involving symbols `X` and `Xcol`.

In these definitions, f_2 is a 2D table-lookup method, f_r is any 1D calculation method from the previous subsection (Table 5, page 43) involving the row variable, f_c is also a 1D method from the previous subsection involving the column variable, and $f_{2c}(Xcol, X)$ is a custom expression involving the symbols `XCOL` and `X`, and possibly specific variables in the VS Math Model.

All Configurable Functions that use two independent variables support at least one 2D table lookup option (i.e., f_2). For example, Figure 21 shows part of an Echo file with documentation about the function `ROAD_DZ` and an example table with columns and rows of numbers following the keyword `ROAD_DZ_CARPET(1)` and specification `2D_STEP`.

```

4730 ! ROAD_DZ: Incremental elevation added to a road surface. DZ can be a nonlinear
4731 ! CARPET function of lateral position and station or a function of station (CONSTANT,
4732 ! COEFFICIENT, or TABLE) combined with a function ROAD_DZ_L of lateral position
4733 ! (CONSTANT, COEFFICIENT, or TABLE). DZ from the calculation can be adjusted with
4734 ! ROAD_DZ_GAIN and ROAD_DZ_OFFSET. Station used in the calculation can be adjusted
4735 ! with SSCALE_ROAD_DZ and SSTART_ROAD_DZ. Lateral position used in the calculation
4736 ! can be adjusted with L_SCALE_ROAD_DZ and L_START_ROAD_DZ. The derivative of this
4737 ! function is used internally. This configurable function supports 200 datasets; if
4738 ! indices shown below are not used, e.g., (2), the current value of the index
4739 ! IROAD_DZ is used to identify the dataset when reading data.
4740
4741 ROAD_DZ_ID(1)      1 ! 3.5 cm Bump (7 m Wide, 40 cm Long, Sharp)
4742
4743 ! 2D table: row 1 has "0" (place holder) followed by 6 values of lateral position
4744 ! (m). Other rows have station (m) followed by 6 values of dZ (m).
4745 ROAD_DZ_CARPET(1) 2D_STEP
4746 0, -10, -4, -3.5, 3.5, 4, 10
4747 105, 0, 0, 0, 0, 0, 0
4748 105.01, 0, 0, 0.035, 0.035, 0, 0
4749 105.39, 0, 0, 0.035, 0.035, 0, 0
4750 105.4, 0, 0, 0, 0, 0, 0
4751 ENDTABLE
4752 ROAD_DZ_GAIN(1)    1 ! Gain multiplied with calculated value to get dZ
4753 ROAD_DZ_OFFSET(1)  0 ; m ! Offset added (after gain) to get dZ
4754 SSTART_ROAD_DZ(1)  0 ; m ! Offset subtracted from station
4755 SSCALE_ROAD_DZ(1)  1 ! Scale factor divided into (station - SSTART_ROAD_DZ)
4756 L_START_ROAD_DZ(1) 0 ; m ! Offset subtracted from lateral position
4757 L_SCALE_ROAD_DZ(1) 1 ! Scale factor divided into (lateral position -
4758 ! L_START_ROAD_DZ)

```

Figure 21. Part of Echo file showing 2D tabular data.

Most Configurable Functions with two independent variables also support at least one of the methods for combining two 1D functions f_r and f_c . If so, the name of the row function f_r follows the convention for 1D functions using the root keyword (e.g., `ROAD_DZ`) and the name of the column function f_c will be mentioned in the comments, e.g., `ROAD_DZ_L`.

In the most general case, a Configurable Function will support all of the options listed in both Table 5 and Table 7. For example, the ROAD_DZ_TABLE command sets the calculation method to use table lookup using one of the interpolation methods listed in Table 5.

Figure 22 shows part of an Echo file from a different simulation, in which a dataset for the ROAD_DZ function is built from two 1D functions that are multiplied together.

```

4530 ! indices shown below are not used, e.g., (2), the current value of the index
4531 ! IROAD_DZ is used to identify the dataset when reading data.
4532 SET_UNITS ROAD_DZ_CARPET(1) in ;
4533 SET_UNITS_TABLE_ROW ROAD_DZ_CARPET(1) ft ;
4534
4535 ROAD_DZ_ID(1)      1 ! Small, Smooth Bump
4536
4537 ! 1D table: col 1 = station (ft), col 2 = dZ (in)
4538 ROAD_DZ_TABLE(1)  SPLINE_FLAT ! spline interpolation, flat-line extrapolation
4539 -0.8202099738, 0
4540 -0.3280839895, 0
4541 -0.1640419948, 0
4542 0, 0
4543 0.8202099738, 0.3167204724
4544 1.640419948, 0.9757086614
4545 2.460629921, 1.371062992
4546 3.280839895, 1.139330709
4547 4.101049869, 0.4935433071
4548 4.921259843, 0
4549 5.085301837, 0
4550 5.249343832, 0
4551 5.741469816, 0
4552 ENDTABLE
4553 ROAD_DZ_GAIN(1)    1 ! Gain multiplied with calculated value to get dZ
4554 ROAD_DZ_OFFSET(1)  0 ; in ! Offset added (after gain) to get dZ
4555 SSTART_ROAD_DZ(1)  0 ; ft ! Offset subtracted from station
4556 SSCALE_ROAD_DZ(1)  1 ! Scale factor divided into (station - SSTART_ROAD_DZ)
4557 ROAD_DZ_COMBINE(1) MULTIPLY ! How to combine the two components
4558 SET_UNITS_TABLE_ROW ROAD_DZ_L_TABLE(1) ft ;
4559
4560 ! 1D table: col 1 = lateral position (ft), col 2 = dZ component due to lateral position (-)
4561 ROAD_DZ_L_TABLE(1) LINEAR_FLAT ! linear interpolation, flat-line extrapolation
4562 -13.12335958, 0
4563 -11.48293963, 1
4564 11.48293963, 1
4565 13.12335958, 0
4566 ENDTABLE
4567 L_START_ROAD_DZ(1)  0 ; ft ! Offset subtracted from lateral position
4568 L_SCALE_ROAD_DZ(1)  1 ! Scale factor divided into (lateral position -
4569                      ! L_START_ROAD_DZ)

```

Figure 22. Part of Echo file showing two linked 1D functions.

When adding two single-variable functions, the units of the output from each function are those of the output variable. However, when multiplying the functions, the units of their product must be those of the output. To assure this, the convention in VS Math Models is that the 1D function of the row variable (e.g., ROAD_DZ_TABLE(1)) always has the same units as the output. The 1D function of the column variable (e.g., ROAD_DZ_L_TABLE(1)) has units that depend on how it is combined with the row variable 1D function:

1. If the two functions are added (e.g., with the command ROAD_DZ_COMBINE(1) ADD), both have units of the output variable.
2. If they are multiplied (e.g., ROAD_DZ_COMBINE(1) MULTIPLY, as in Figure 22), the units of the 1D function for the column variable are dimensionless.

Of course, if a 2D table is specified (e.g., ROAD_DZ_CARPET(1)), as in Figure 21), then f_c is not used and its units are irrelevant.

Alert Because the `_COMBINE` command affects units, it must be used before providing any data for the column function. For example, if the default combination method is `MULTIPLY`, any values provided for the second 1D function will be processed with dimensionless units. If the `_COMBINE` command is later used to change the combination method to `ADD`, the internal calculations will be incorrect because the scaling due to units was not made when the values were read.

If the `_EQUATION` option is available, the user-defined formula has two independent variables: `X` and `XCOL`. The symbol `X` represents the independent variable associated with rows in a 2D table; the symbol `XCOL` represents the independent variable associated with columns in a table. For more information about using the `_EQUATION` option to add a symbolic equation, please see *VS Commands Manual*.

Variable-Width Tables

Two of the 2D table types support variable-width tables. In these cases, the Configurable Function uses two 2D tables, as shown in Figure 23.

```

4555 ! can be adjusted with L_SCALE_ROAD_DZ and L_START_ROAD_DZ. The derivative of this
4556 ! function is used internally. This configurable function supports 200 datasets; if
4557 ! indices shown below are not used, e.g., (2), the current value of the index
4558 ! IROAD_DZ is used to identify the dataset when reading data.
4559
4560 ROAD_DZ_ID(1)      1 ! Highway Merge Lane
4561
4562 ! 2D variable width table: (data section) row 1 is a header for the 23 lanes of
4563 ! lateral position (m). Other rows have station (m) followed by 23 values of dZ (m).
4564 ! (column section) Row 1 is a header for the lanes. Other rows have station (m)
4565 ! followed by the values of lateral position (m).
4566 ROAD_DZ_CARPET(1) VAR_WIDTH
4567 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
4568 0, 0, 0, -0.5, -1, -0.5, 0, 0, 0.1, 0, 0, -0.5, -1, -0.5, 0, 0, 0.1, 0, 0, -0.5, -1,
4569 1000, 0, 0, -0.5, -1, -0.5, 0, 0, 0.1, 0, 0, -0.5, -1, -0.5, 0, 0, 0.1, 0, 0, -0.5,
4570 ENDDATA
4571 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
4572 0, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 33.9, 34.9, 35, 39, 40,
4573 100, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 33.9, 34.9, 35, 39, 40
4574 150, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 33.9, 34.9, 35, 39, 40
4575 200, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 33.9, 34.9, 35, 39, 40
4576 600, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 30, 31, 35, 39, 40, 42
4577 1000, -50, 2, 2.5, 5, 7.5, 10, 11, 15, 19, 20, 22.5, 25, 27.5, 30, 31, 35, 39, 40, 4
4578 ENDCOLS
4579 ENDTABLE
4580 ROAD_DZ_GAIN(1)    1 ! Gain multiplied with calculated value to get dZ
4581 ROAD_DZ_OFFSET(1)  0 ; m ! Offset added (after gain) to get dZ
4582 SSTART_ROAD_DZ(1)  0 ; m ! Offset subtracted from station
4583 SSCALE_ROAD_DZ(1)  1 ! Scale factor divided into (station - SSTART_ROAD_DZ)
4584 L_START_ROAD_DZ(1) 0 ; m ! Offset subtracted from lateral position
4585 L_SCALE_ROAD_DZ(1) 1 ! Scale factor divided into (lateral position -
4586 ! L_START_ROAD_DZ)
4587

```

Ln 4566, Col 1 Insert Sel: Normal DOS File size: 278954

Figure 23. Part of an Echo file showing a variable width dataset.

In each table, the column variable is a simple integer that identifies the column. The first table of numbers (after the 2D keyword, ROAD_DZ_CARPET (1) , but before the keyword ENDDATA) has values of the output variable vs. values of the row variable and the column ID. The second table (after the separator keyword ENDDATA but before the keyword ENDCOLS) has values of the column variable (lateral position L for the example) as a function of the row variable and column ID.

Variable-width Configurable Functions are used in VehicleSim products to describe road elevation (as in Figure 23) and road friction. They are also described in the document *VehicleSim 3D Ground and Roads*.

Summary of Configurable Function Properties

Table 8 summarizes the calculation methods available in Configurable Functions to obtain a value F from one or two independent variables. The keywords for the many options were listed earlier in Table 5 (page 43) and Table 7 (page 47).

Table 8. Summary of calculation methods for Configurable Functions

Type	Command	Argument	Interpolation	Extrapolation
Constant	<i>func</i> _CONSTANT	Value	—	Flat
Coefficient	<i>func</i> _COEFFICIENT	Value	—	Linear
1D Table	<i>func</i> _TABLE	Keyword	Linear	Linear, Flat, Loop
			Spline	Linear, Flat, Loop
			Step	Flat, Loop
2D Table	<i>func</i> _CARPET	Keyword	Linear	Linear, loop, From Zero
			Step	Flat
			Spline	Linear
			Variable-width linear	Linear
			Variable-width step	Flat
Equation	<i>func</i> _EQUATION	Formula	—	—

For a given function name *func* (e.g., FD, ROAD_DZ), Table 8 shows that there may be up to five commands that configure the function to use a type of calculation. The commands for a constant or a coefficient set the function up with no table interpolation. The command for an equation sets the function up to evaluate a formulation involving a row variable, possibly a column variable, and possibly any other variables of interest that are in the model. The table options involve both interpolation within the range of tabular data provided as input, and extrapolation beyond the range.

As mentioned earlier, examples for each type of calculation method are shown in the document *VS Browser (GUI and Database)*.

Table 8 shows one extrapolation option that has not been mentioned. This is a legacy 2D table option used for symmetric tire data. When selected, the tire model automatically adds a row for

zero slip, and a column for zero vertical load. The only Configurable Functions that support this option are for tire forces and aligning moment.

The commands with the `_COMBINE` suffix (e.g., `ROAD_DZ_COMBINE`) do not immediately change the type of calculation. The option to calculate F by combining two 1D methods is selected by using any of the commands in Table 8 other than `_CARPET` or `_EQUATION`. The combination mode is disabled by a `_CARPET` or `_EQUATION` command.

Transforming Configurable Functions

Up to six parameters are available in some Configurable Functions to provide scaling and offsets for the calculated value and for the independent variable(s) (Table 9). These parameters are all visible in the Echo file excerpts shown in Figure 21, Figure 22, and Figure 23.

Table 9. Configurable function modifiers and keywords.

Keyword / Suffix	Example: ROAD_DZ	Description
<code>_GAIN</code>	<code>ROAD_DZ_GAIN 1</code>	Gain and offset for calculated value
<code>_OFFSET</code>	<code>ROAD_DZ_OFFSET 0</code>	
<code>scale_xrow</code>	<code>SSCALE_ROAD_DZ 1</code>	Scale and offset for primary independent variable
<code>start_xrow</code>	<code>SSTART_ROAD_DZ 0</code>	
<code>scale_xcol</code>	<code>L_SCALE_ROAD_DZ 1</code>	Scale and offset for secondary independent variable
<code>start_xcol</code>	<code>L_START_ROAD_DZ 0</code>	

The `_GAIN` and `_OFFSET` parameters are always applied to the function value regardless of how it was calculated; the value used in the VS Math Model is:

$$F_m = F \cdot \text{gain} + \text{offset}$$

where F is the output value of the Configurable Function as calculated from X and $Xcol$ according to the equations given in the previous subsections. X and $Xcol$ are calculated from variables in the VS Math Model together with parameters of the Configurable Function:

$$X = (x_{\text{row_model}} - \text{start_xrow}) / \text{scale_xrow}$$

and

$$Xcol = (x_{\text{col_model}} - \text{start_xcol}) / \text{scale_xcol}$$

where $x_{\text{row_model}}$ is the row variable from the VS Math Model (e.g., station along the road reference line) and $x_{\text{col_model}}$ is the column variable from the VS Math Model (e.g., lateral position relative to the road reference line).

In all Configurable Functions, the default values of the `_GAIN` and `SCALE` parameters are unity; the default values of the `_OFFSET` and `START` parameters are zero. In most cases the default values are used; these parameters are provided for advanced applications.

As an example of an advanced application, Figure 24 shows a normalized version of the sine-with-dwell waveform used in several regulatory tests. The nonlinear function of time covers a steering wheel angle range of $\pm 1^\circ$. In a specific application, this waveform is typically rescaled using the *gain* parameter for the function.

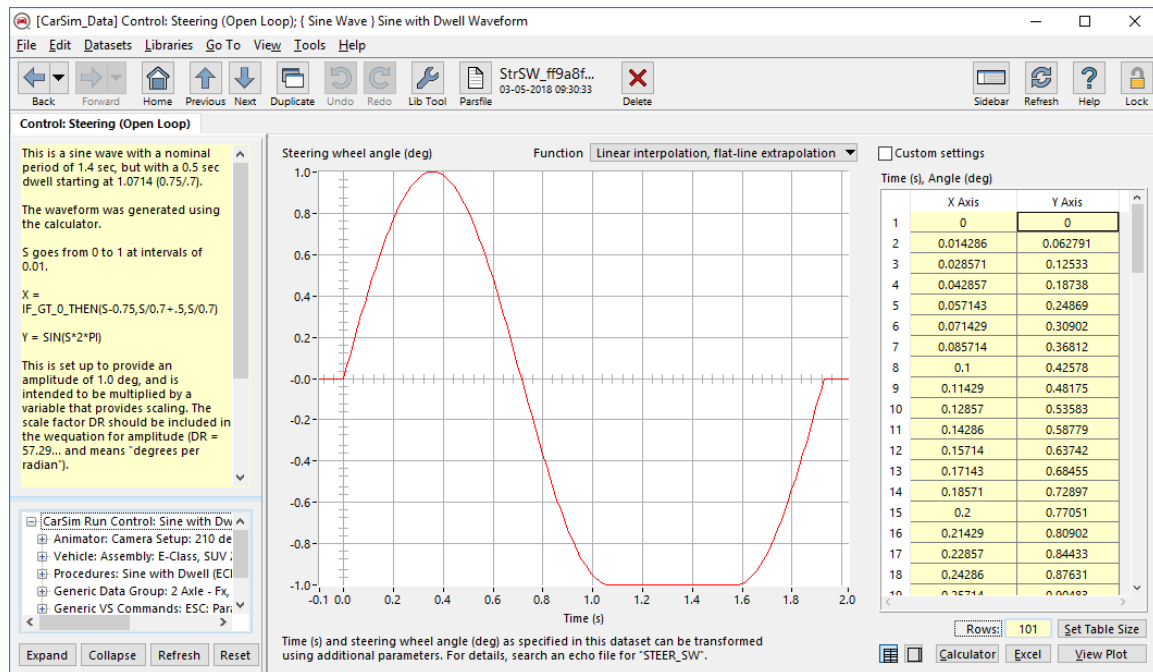


Figure 24. Normalized sine-with-dwell waveform.

The note at the bottom of the screen in the figure identifies the root keyword is STEER_SW. Search an Echo file to find the data for this function. For example, Figure 25 shows part of an Echo file made at the end of a run that was interrupted during a sequence of sine with dwell tests. When the run was stopped, the gain for the amplitude was -120.781125 (keyword = STEER_SW_GAIN).

```

ConTEXT - [Z:\Current\CarSim_2018.1\CarSim_Data\Results\Run_d8fe68ee-6e6b-4a26-9ba1-5c4...
File Edit View Format Project Tools Options Window Help

! STEER_SW: Open loop steering wheel angle as a function of time. Steering wheel
! angle is a function of time (CONSTANT, COEFFICIENT, or TABLE). Alternatively, a
! custom equation can be defined at runtime. Steering wheel angle from the
! calculation can be adjusted with STEER_SW_GAIN and STEER_SW_OFFSET. Time used in
! the calculation can be adjusted with TSCALE_STEER and TSTART_STEER.

! 1D table: col 1 = time (s), col 2 = steering wheel angle (deg)
STEER_SW_TABLE LINEAR_FLAT ! linear interpolation, flat-line extrapolation
0, 0
0.014286, 0.062791
0.028571, 0.12533
0.042857, 0.18738
0.057143, 0.24869

1.8857, -0.18738
1.9, -0.12533
1.9143, -0.062791
1.9286, 0
ENDTABLE
STEER_SW_GAIN -120.781125 ! Gain multiplied with calculated value to get steering
! wheel angle
STEER_SW_OFFSET 0 ; deg ! Offset added (after gain) to get steering wheel angle
TSTART_STEER 158.025 ; s ! Offset subtracted from time
TSCALE_STEER 1 ! Scale factor divided into (time - TSTART_STEER)

Ln 4886, Col 77 Insert Sel: Normal Modified DDS File size: 303810

```

Figure 25. Sine with dwell data taken from the middle of a run.

The row variable (e.g., time) can be transformed before it is used to calculate the steering wheel angle with the parameters *scale_xrow* and *start_xrow*.

In this example, the parameter `TSTART_STEER` is typically set to the time when the steering is supposed to start. In Figure 25, the start time is 158.025s (keyword = `TSTART_STEER`).

Initial Conditions of State Variables

At any instant, the state of a VS Math Model is completely defined by the values of the state variables, along with the values of the static parameters and Configurable Functions. Many of the variables are calculated as the run proceeds by numerically integrating differential equations; however, some are calculated by other means. For example, the current force in a hysteretic spring model is needed to fully define the state of the model and is therefore included as a state variable. Regardless of how they are calculated, the concept of state variables implies that the state of the VS Math Model is not completely determined unless values are known for all of the state variables.

Any of the state variables in a VS Math Model can be given an initial value from the Parsfile using the same syntax as used for non-indexed parameters:

keyword [=] *expression* [; [[UNITS] [=] *units*] [; [*description*]]]

where *keyword* is the name of the state variable, *expression* is either a number or an algebraic expression (see *VS Commands Manual* for details about symbolic expressions), *units* is an optional keyword used to specify the units associated with the parameter or variable specified by *keyword*, and *description* is a new text description of the parameter or variable.

For example, the statement

```
SV_YO -100
```

sets the value of the y-coordinate of the sprung mass origin to -100 meters.

As shown in Chapter 2, a list of state variables can be obtained with the **View** button from the **Run Control** screen after selecting “State variables in math model (text).” This option can be used even if the simulation has not yet been run.

In general, setting initial values of state variables is not recommended. The VS Math Models perform a set of initialization calculations to put the vehicle in a state close to equilibrium. (For details of how this is done in CarSim and TruckSim, please see the Technical Memo: *Initialization in CarSim and TruckSim*.) Setting values for state variables involving suspension deflection or speeds of any kind are likely to put the model in an unstable state that will cause an initial transient response that might take a while to settle out.

4. Extending the Math Models

Each vehicle VS Math Model contains highly optimized equations for the physics of the vehicle, plus driver controls and environmental interactions (3D road, wind, etc.). By itself, the VS Math Model covers the complete system-level dynamic behavior of the vehicle as controlled by a driver or rider, with closed-loop control options sufficient to mimic driver/rider behavior in basic maneuvers that have traditionally been used by manufacturers to characterize vehicle behavior.

VS Math Models support a variety of methods for extending the existing internal model, using both built-in capabilities and external tools (Table 10).

Table 10. Options for extending a VS Math Model at runtime.

Method	Built-In Support	External Tool	Description
Built-in options	Math models include common control and configuration options	none	Use built-in options for setting controls, tire models, powertrain options, etc.
Optional modules	Commands for adding optional features with multiple variables	none	Commands such as <code>DEFINE_PATHS</code> add features to the model
VS Commands	VS Math Models recognize VS Commands	none	Add variables and equations to extend model
Embedded Python	Interface to Python	Python	Apply Python functions from within the VS Math Model
Import/Export variables	Import and export arrays activated at runtime	Simulink, LabVIEW, ASCET, FMI, Custom programs	Several hundred built-in variables can be replaced or modified with values imported from other software.
		Real-time systems with hardware in the loop (HIL)	Live measures from HIL are imported to the VS Math Model
VS API	VS Math Models have documented API	Programming languages such as C/C++, Python, VB, MATLAB, etc.	External program can control the simulation, access many variables in the model, and add complex calculations

The first row in Table 10 is a reference to many built-in options that can be activated or disabled based on settings. For example, CarSim supports powertrains with rear-wheel drive, front-wheel drive, and all-wheel drive, with more options for transmission type, electric power, hybrid, etc. Every VS Math Model for BikeSim, CarSim, and TruckSim includes many built-in options that are mutually exclusive. Changing an option can drastically modify the capabilities of the model by eliminating or adding features. Built-in options are described by Echo files written by the VS Math Models, as described earlier in the section *Echo and End Files* (page 16). Most built-in options are also described in the documentation available from the **Help** menu (powertrain options are

documented in the **Powertrain** help document; tire options are documented in the **Tire Models** help document, etc.)

VS Math Models also support optional features that are installed with `INSTALL_` and `DEFINE_` commands, such as `DEFINE_PAYLOADS`, `DEFINE_PATHS`, etc. These extend the VS Math Model to include more features, which usually involve modules with associated parameters, variables, and Configurable Functions that are defined by the command. For example, an ADAS scenario might involve 20 traffic vehicles, 12 paths, and 3 ADAS sensors, with hundreds of parameters and output variables that would not exist when using the same VS Math Model for a basic vehicle dynamics test.

VS Math Models support generic VS Commands for adding variables and equations. VS Events are used to monitor output variables to determine if a defined condition occurs; when the condition occurs the VS Event is triggered, the VS Math Model will either stop the simulation or read a specified Parsfile that will change any number of model parameters. Options for extending the model with VS Commands are described in detail in *VS Commands Manual*.

When running on Windows or non-RT Linux, VS Math Models can load Python and run embedded Python code from within the model, in a similar manner to VS Commands. For more information, please see the Technical Memo *Extending a Model with Embedded Python*.

The options for using Import and Export variables to communicate with external software (Simulink, LabVIEW, etc.) involve making a model in the external software, such as an advanced controller, and connecting it to the VS Math Model using appropriate Import and Export variables. Details for using external software are beyond the scope of this manual; however, the setting up of arrays of Import and Export variables involve just a few commands that are included in the Parsfile read by the VS Math Model, as described in the remainder of this section.

Options for extending the model with external programs that run the VS Math Model using the VS application program interface (API) functions are described in the *VehicleSim API Manual*.

Communicating with Import and Export Arrays

VS Math Models for Windows can be launched and controlled by the VS Browser or a VS Solver Wrapper program, such that no other software is needed. This mode of simulation has been assumed in most of the material presented up to this point. Alternatively, VS Math Models for all platforms may be launched and controlled from other simulation environments such as MATLAB, Simulink, LabVIEW, ASCET, FMI, and custom software.

To work with external software such as Simulink, the VS Math Model is run from a wrapper program or library that connects the VS Solver library to the other environment. The wrapper communicates with the calling environment in a way that is standard for that environment and communicates with the VS Solver library using the VS API. For example, Figure 26 shows the relationships between a Simulink model, the CarSim S-Function library, and a VS Solver library.

In the example, the VS Browser launches a Simulink model, prepares input files needed by the VS Math Model, and sends COM commands to Simulink. During the run, there is close communication between Simulink and the S-Function functions, and between the S-Function functions and the VS Solver functions.

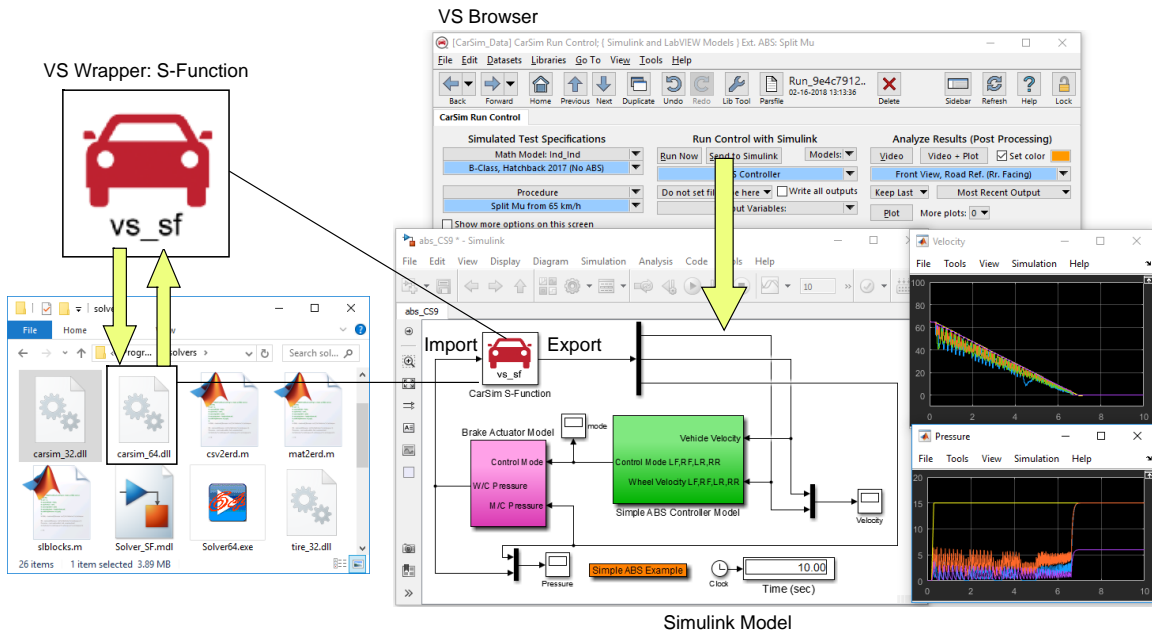


Figure 26. Running a VS Math Model with Simulink.

Simulation programs such as Simulink have standard methods for communicating with program modules using arrays of Import variables and arrays of Export variables.

Import Variables

The VS Math Models allow other simulation environments, such as Simulink, LabVIEW, ASCET, or custom software, to set the values of selected *Import variables*.

Each VS Math Model includes a set of hundreds of variables that can be imported. As shown earlier (Chapter 2), the list of variables that are available for import can be viewed from a VS Browser by using the **View** button on the lower-right corner of the **Run Control** screen after selecting **Imports into math model (text)** or **Imports into math model (Excel)** from the drop-down list.

By default, all potential Import variables are ignored unless they are activated from the Parsfile. Within the Parsfile, the syntax for activating an Import variable is:

```
[IMPORT] keyword [mode [initial_value]] [; [UNITS] [=] units] [;]
```

For example, the statement

```
IMPORT IMP_FD_L1 replace
```

activates the variable to import the left front damper force, replacing the internally calculated value.

Keyword is the name of the Import variable. *Mode* is a keyword that specifies a mode for using the Import variable. *Initial_value* is an expression that used to help with initialization. *Units* can be used to change the units of the variable as defined in the external software. For example, if a variable has units of degrees in Simulink, each time the variable is imported it is converted to SI units of radians for internal use in the VS Math Model.

Note that all keywords except the name of the variable (*keyword*) are optional.

Note Prior to version 2018.1, the array of import variables was not available during initialization, and the optional numerical expression *initial_value* could be used to improve the initialization. In newer versions, the expression *initial_value* usually has no effect.

Mode determines how the imported variable is combined with a native value that is calculated in the VS model and would be used if there were no Import. Table 11 lists the three valid *mode* names that activate variables that will be imported from external software. For example, if the imported variable is the brake control and the mode is `REPLACE`, then the brake control within the VS Math Model is replaced with the value of the Import variable. If the VS Math Model is running as part of a Simulink model, the brake control is included in the array of variables that are transferred each time step from Simulink to the VS Math Model. If the mode is `ADD`, then the brake control is also included in the array of variables transferred from Simulink; however, in this case, the imported value is added to the internal value. If the mode is `MULTIPLY`, then the internal value is multiplied by the imported value. If not specified, the default mode is `ADD`.

Table 11. Modes for activating variables for import from external software.

Mode Keyword	Description
<code>ADD</code>	Add Import variable to built-in variable.
<code>MULTIPLY</code>	Multiply Import variable with built-in variable. The Import variable is set to be dimensionless, regardless of the units associated with the variable.
<code>REPLACE</code>	Replace value of built-in variable with value of Import variable.

Not all modes can be used for all Import variables. In some cases, there is no built-in internal value. The availability of an internal value is indicated in the machine-generated text file that lists Import variables (Figure 13, page 27) with the text "[V]" at the end of the description. If the description included [V], then there is an internal value that can be combined with an Import based on the mode. If not, then *mode* has no effect.

When activating import variables from the VS Browser screen **I/O Channels: Import**, the pull-down control used to select *mode* is not active for variables that do not specify [V] in the machine-generated documentation.

Note VS Math Models also support modes that can be used with VS Commands and are independent of the interactions with Simulink or other external software. These commands — `VS_ADD`, `VS_MULTIPLY`, and `VS_REPLACE` — are described in *VS Commands Manual*.

The default units for the Import variables are listed in the two documentation files accessed with the **View** button on the **Run Control** screen (see Table 3 on page 24). If different units are used in the external model, then they should be specified using the *units* keyword. (Alternatively, the VS Command `SET_UNITS` can be used to change the units.) However, if the mode for the variable is

set to `MULTIPLY`, then the Import variable is always treated as a dimensionless ratio and any specifications for units are ignored.

Any potential Import variables that are not explicitly activated are ignored during the run.

If the optional `IMPORT` command is used and *keyword* is not recognized as a valid potential Import variable, then an error message is generated identifying the name that is not recognized. On the other hand, if the line in the Parsfile begins with *keyword* and it is not recognized, no error message is generated. In most applications, failure to recognize *keyword* will cause problems with the external model (e.g., Simulink), so aggressive error handling is preferred. For this reason, the optional `IMPORT` command *keyword* is recommended.

New Import variables can be defined at runtime using the VS Command `DEFINE_IMPORT` (see *VS Commands Manual*). These variables are also listed in the documentation files accessed with the **View** button on the **Run Control** screen and are activated in the same manner as the predefined Import variables.

VS Math Models support two options for controlling the timing of the communication with outside software, specified with the system parameter `OPT_IO_SYNC_FM`, described in a later section (page 64).

Export and Output Variables

Each VS Math Model includes several potential output variables that can be written to file for later plotting and animation. They can also be streamed for live animation. The list of potential output variables can be viewed from the **Run Control** screen of a VS Browser using the **View** button after selecting **Outputs from math model (text)** or **Outputs from math model (Excel)** from the drop-down list (see Table 3 on page 24), or by using a VS Browser library screen available for specifying output variables interactively.

The same output variables can also be exported to other simulation environments such as Simulink, LabVIEW, ASCET, FMI, or custom software.

Variables that are (1) exported, (2) written to file, or (3) sent to a live animator all have the user display units indicated. For example, yaw angle will have already been converted from internal SI units of radians to the user units of degrees. All three possible copies of the output variable have the same units, which can be modified using the `SET_UNITS` VS Command as described in *VS Commands Manual*.

Exporting Variables to External Software

Within the Parsfile, two options can be used to specify that the output should be exported to other software such as Simulink. One is the command `EXPORT`, and the other is a custom command made with the prefix `EXP_`. Both of the following lines of input have the same effect if *name* is an existing output variable:

```
EXPORT name
EXP_name
```

For example, to export the variable Yaw, add the line:

```
EXPORT YAW
```

If the output variable does not exist, the VS Math Model will generate an error message for the `EXPORT` command, indicating that the variable name is not recognized. On the other hand, if the `EXP_` prefix is used with a name that is not recognized, no error message is generated. In most applications, failure to recognize *keyword* will cause problems with the external model (e.g., Simulink), so aggressive error handling is preferred. For this reason, the `EXPORT` command option is recommended.

VS Math Models support two options for controlling the timing of the communication with outside software, specified with the system parameter `OPT_IO_SYNC_FM`, described in a later section (page 66).

Broadcasting Variables for Live Video

Similar options are available to specify that the output should be broadcast during interactive simulation to a live video (animated), as in the case of a driving simulator. One is the command `ANIMATE`, and the other is a custom command made with the prefix `ANI_`. Both of the following lines of input have exactly the same effect if *name* is an existing output variable:

```
ANIMATE name
ANI_name
```

For example, to export the variable Yaw, add the line:

```
ANIMATE YAW
```

Writing Variables to File for Post Processing

Similar options can be used to specify that an output variable should be written to the BIN file for plotting and animation. One is the command `WRITE`, and the other is a custom command made with the prefix `WRT_`. Both of the following lines of input have exactly the same effect if *name* is an existing output variable:

```
WRITE name
WRT_name
```

For example, to specify that the output variable Yaw should be written to file, add the line:

```
WRITE YAW
```

or

```
WRT_YAW
```

In normal use with control from the VS Browser, commands are automatically written to activate output variables for plotting and animation based on links to plot and animation datasets. When a run is made, the VS Browser scans all linked plot setups and automatically uses `WRT_` commands to activate output variables for writing to the VS/ERD files. The VS Browser also scans all animation datasets to add `WRT_` commands for post-processing animation, and `ANI_` commands for live animation. Therefore, you do not normally need to be concerned with these commands. (However, they might be needed if running the VS Math Models from other software.)

The browser uses the `WRT_` prefix rather than the `WRITE` command because there are often many references to variables that are in some VS Math Models but not all (e.g., variables related to trailer motions). If the variables do not exist, then the `WRT_` version will not generate an error message.

Multiple Ports

Simulink and Functional Mockup Interface (FMI) allow import and export variables to be grouped into Ports, where each port has at least one signal, and possibly more. Figure 27 shows part of a model with four import Ports to a CarSim S-Function, and three output Ports.

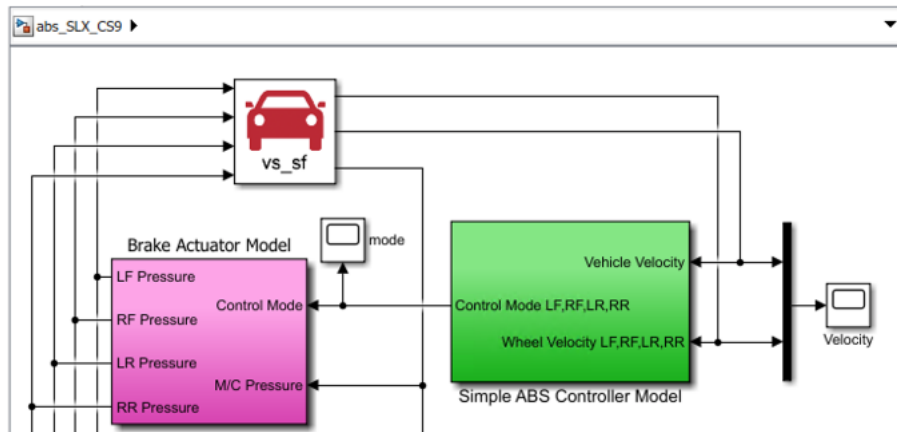


Figure 27. Part of a Simulink model using multiple Ports with a CarSim S-Function.

The VS Math Model does not have groups of import and export variables. However, the S-Function for Simulink and the VS FMU for FMI models both support Ports.

The organization of I/O variables into ports is defined by two commands that are contained in the Simfile: `PORTS_IMP` and `PORTS_EXP`. The syntax for each is:

command *nports* , *nv1* , [*nv2* [, *nv2* [...]]]

where *command* is `PORTS_IMP` or `PORTS_EXP`, *nports* is the number of ports, *nv1* is the number of variables for port 1, *nv2* is the number of variables for port 2, and so on.

The two commands used for the example Simulink model in Figure 27 are:

```
PORTS_IMP 4,1,1,1,1
PORTS_EXP 3,4,1,1
```

In this example, there is a total of four import variables and six export variables.

5. Time Steps During Simulation

A VS Math Model runs a simulation by performing calculations at closely spaced intervals of time, using the VS Math Model equations. As mentioned earlier, variables that are needed to define the state of a VS Math Model and which normally change during the simulation run are called *state variables*. Together with the model parameters and Configurable Functions, the state variables are used to calculate the VS Math Model output variables.

VS Numerical Integration Methods

Many of the state variables are defined within the VS Math Model with ordinary differential equations (ODEs). For example, a basic CarSim vehicle model has about 100 ODEs. These state variables are then calculated from their derivatives using numerical integration.

In theory, each ODE state variable is a continuous function of time: $q(t)$. However, in a simulation run, calculations are made at discrete intervals of time. Figure 28 shows the history for a state variable q at discrete values of time separated by a constant step ΔT , where ΔT is the time-step specified with the parameter `TSTEP`. At each step, identified with a count i , there is a corresponding value of q_i and its derivative $\dot{q}(t)$, indicated as q'_i in the figure.



Figure 28. Discrete samples of a state variable q and its derivative q' .

The VS Math Model computes the values of each output variable at time t , using current values of the state variables and possibly their derivatives, where each derivative is calculated from an ODE in the VS Math Model. The ODE state variables are in turn calculated using numerical integration.

VS Math Models support six methods for numerical integration, specified with the parameter `OPT_INT_METHOD`. Two of the methods involve calculations at `TSTEP`, and the other four include a second set of calculations at a half step `TSTEP/2`.

Table 12 lists the numerical integration methods available in VS Math Models. In the table, AM indicates an Adams-Moulton method, AB is Adams-Bashforth, and RK is Runge-Kutta. “Method” indicates the value of the system parameter `OPT_INT_METHOD`. “Order” is the theoretical relationship between the time-step and numerical error for smooth equations, e.g., the error varies as ΔT^2 for a second-order method. “Half-Step” indicates whether calculations are made at the half-step $\Delta T/2$.

Table 12. Numerical integration methods available in VS Math Models.

Name	Method	Order	Half-Step	When to Use
Euler	-1	1	No	Research and diagnostic applications.
AB-2	0	2	No	Whenever possible, with $T_{STEP} = 0.0005s$.
AM-2	2	2	Yes	When linking to external models that cannot run with a $0.0005s$ step; use $T_{STEP} = 0.001s$ (half-step = $0.0005s$).
AM-3	3	3	Yes	Scenarios with continuous conditions.
AM-4	4	4	Yes	Scenarios with exceptionally continuous conditions.
RK-2	1	2	Yes	Scenarios with discontinuities that are bad for AM-2.

The recommended options are:

1. The AB-2 method with time step of $0.0005s$ ($2000Hz$) is recommended for all setups except when using external software where this $2000Hz$ frequency is not possible.
2. The AM-2 method with time step of $0.001s$ ($1000Hz$) is recommended when using external software when the external software cannot update at $2000Hz$. Be sure the parameter `OPT_IO_UPDATE` is set to zero in this case.

The integration method is set at the beginning of a run and cannot be changed during the run.

Note If an AM or RK half-step method is chosen, a system parameter `OPT_IO_UPDATE` can be used to communicate with external software at intervals of $T_{STEP}/2$. This legacy option is not recommended for routine use; if variables are to be exchanged rapidly, then the AB-2 method is preferred.

For more information about these methods, please see the tech memo *Numerical Integration Methods in VS Math Models*.

Other State Variables

The term *equation of motion* is sometimes used to describe an equation used to calculate derivatives for ODEs.

VS Math Models also include state variables that are not defined with ODEs. These are needed to define conditions of interest in the model, such as whether a clutch is locked or slipping, or the current value of a force that is affected by friction.

State variables that are calculated by numerically integrating their derivatives are identified in machined-generated Echo files and text documentation with “ODE” in the description; if the description does not include “ODE,” then the state variable is used to define some other state in the model, such as a friction force.

The VS Math Model also includes hundreds (or thousands) of output variables for use in post-processing, such as plotting and animation. These output variables are all available for exporting to other software, such as Simulink.

To summarize, whenever the VS Math Model needs derivatives for the ODEs, values are calculated for three groups of variables:

1. ODE State variables are calculated by numerical integration using the values from the most recent full time step plus values of their derivatives as calculated at one or more previous time steps, including half steps.
2. Other state variables are calculated directly using the current values of (potentially) all model parameters and all state variables.
3. Output variables are calculated directly using the current values of (potentially) all model parameters and all state variables.

A distinction is that state variables calculated by numerical integration (group 1) are calculated differently at the half step than is the case at the full time step. On the other hand, values of variables from groups 2 and 3 are calculated the same way at the half step as they are at the full step.

The AM and RK methods can be efficient if results at the half step are not of interest, especially when working with external software (e.g., Simulink) that does not need updating at the half-step interval. On the other hand, if tight communication with external software is needed, then Mechanical Simulation recommends using a single-step method (typically AB-2) with a smaller time step (e.g., 0.0005s).

Simulation Variables for Time

Time is the one truly independent variable in a time-domain VS Math Model; it is not calculated by the model but is instead provided by the simulation environment. The simulation time in a VS Math Model starts with a specified value `TSTART` and increases incrementally as the simulation proceeds.

As mentioned earlier, the outputs from the VS Math Model are usually not saved every time step. If the calculation time step `TSTEP` is 0.0005 s and the output is 0.025 s, results are only written to file at multiples of 50, specified with the parameter `IPRINT = 50`. However, it is possible to write to file every time the calculations are made. For the AM and RK integration options, outputs may be written at both the full step and the half step. To do this, set `IPRINT` to 0.

Time in the Math Model and the Output File

The VS Math Model for a vehicle or other dynamic system integrates internal ODEs with respect to simulation time `T`. When viewing results by video or plotting, the main independent variable is also time. In this case, the concept of time is based on the recording. The VS Math Models allow output to be enabled and disabled, so the duration of the recording can be less than the duration of the simulation.

Table 13 lists four names for time. Two (`T` and `T_Stamp`) apply to the simulation time. Both are associated with the same internal variable, but only `T_Stamp` is used to identify a channel in the

output file. The other two (Time and T_Record) apply to the recorded time. The variable Time is calculated by post-processing software such as VS Visualizer using the start time T_START, the (calculated) time interval for output, TSTEP_WRITE, and the sample number from the file (1, 2, 3...). The variable T_Record is calculated internally the same way within the VS Math Model. This is done to support plotting and analyses using CSV output files that do not include time step information.

Table 13. Variables for simulation time and recording time.

Variable	Location	Definition
T	VS Math Model	Simulation time calculated by numerical integrator or provided by external software (e.g., Simulink)
Time	VS/ERD File	Time from sample number in an output file
T_Record	Both the model and file	Alternate name for Time that is also written to output file
T_Stamp		Alternate name for T that is also written to output file

Three of the variables are recognized within the VS Math Model: T, T_Record, and T_Stamp, and may therefore be referenced in VS Commands. However, they are protected and cannot be modified. Three are available for setting up plots with VS Visualizer: Time, T_Record, and T_Stamp.

Note Multiple names are used for the two variables to support legacy setups when different variables were used internally and written to file.

Variables Representing Real Time

There are times when it is useful to track the “real time,” sometimes called clock time or wall-clock. For example, if a simulation covers 60 simulated seconds, it might run in 4 seconds of real time. Clock precision can vary by platform, but the math model will attempt to use highest precision available to provide elapsed real time. Two variables are provided for the real time needed each time step. One — T_Real_Step — is used only by the VS Math Model, from the start of the calculations to the finish. The other — T_Real_Last — goes from the finish of the previous time step to the finish of the last time step. This includes time outside the calculations of the VS Math Model that is due to external software.

Table 14. Variables based on real clock time.

Variable	Definition
T_Real_Elapsed	Elapsed real time since the start of the simulation
T_Real_Last	Total real time used by the Math Model plus the calling program for the last time step
T_Real_Step	Real time used by the Math Model for the last time step

Timing for Import and Export Variables

The built-in equations of motion for a VS Math Model are organized roughly into two groups:

1. Kinematical equations are used to calculate variables involving position and velocity, including the derivatives of position state variables (translational coordinates and rotation angles).
2. Dynamical equations are used to calculate forces and moments, derivatives of speed state variables (translational and rotational accelerations), and miscellaneous output variables involving forces, moments, and accelerations.

When running under the control of external software and using Import and Export arrays, each time step the VS Math Model copies all values from the Import array for use in model calculations. Any calculation that can potentially make use of an Import variable will do so if the Import variable was activated as described earlier (page 57).

As the VS Math Model performs calculations (kinematical and dynamical), new values are calculated for all output variables.

If output variables are activated for export (page 59), the updated values are copied to the Export array before control is returned to the external software.

VS Math Models support two options for controlling the timing of the communication with outside software, specified with the system parameter `OPT_IO_SYNC_FM`.

Note The parameter `OPT_IO_SYNC_FM` is available only when the numerical integration method is Euler or AB-2 (these are set with `OPT_INT_METHOD` set to `-1` or `0`, respectively) and the simulation includes both import and export variables.

Conventional sequence

When `OPT_IO_SYNC_FM` is zero, all model calculations are made for the specified simulation time T . At the start of the time step, all state variables that were calculated by numerical integration have known values. The kinematical and dynamical equations are applied, and the output variables that are activated for export are scaled to user units and copied to the Export array.

Before returning control, the solver applies numerical integration to calculate all ODE state variables for the next time step ($T + T_{STEP}$ or $T + T_{STEP}/2$, depending on the integration method).

The conventional sequence works perfectly if the variables imported into the VS Math Model block are based on factors external to the vehicle model, such as clock time.

Option for synchronizing imported forces and exported kinematics

If the variables being imported are calculated based on variables that are exported, the imported variables have a time lag. This type of connection occurs when parts of the multibody model are defined externally, such as tires, springs, powertrain, etc. Deflections and speeds exported from the VS model are used to calculate forces and moments that are provided as imports to the VS model at the next time step.

An alternative timing sequence is used internally when the system parameter `OPT_IO_SYNC_FM` is assigned a nonzero value. When this parameter is nonzero, forces and moments calculated externally are synchronized with the rest of the model by using a different calculation sequence for time T :

1. Skip the kinematical equations; the values for this time T were calculated in the previous time step.
2. Apply the dynamical equations.
3. At this point, all derivatives of ODE state variables have been calculated. Numerically integrate the ODEs to obtain the state variables for time $T + TSTEP$.
4. Apply the kinematical equations for the new time $T + TSTEP$.
5. Scale all output variables activated for export and copy the values into the Export array.

With this approach, the variables calculated in the kinematical functions apply for time $T + TSTEP$, while all other variables (forces, accelerations, miscellaneous outputs) have values calculated for time T . The next time step, forces and moments imported from the external model will be synchronized with the new time $T + TSTEP$.

For example, Figure 29 shows simulation results from a CarSim example that uses Simulink to model brake hydraulics and an ABS controller that duplicates internal features of the CarSim VS Math Model. Four simulations are compared:

1. Simulink model with preview (blue line and the label “Simulink, Preview On”),
2. Simulink model without preview (red line, “Simulink, Preview Off”).
3. built-in model (green line, “Built-In”), and
4. built-in model with 16 kHz updates (gray line, “Built-In, 16k”).

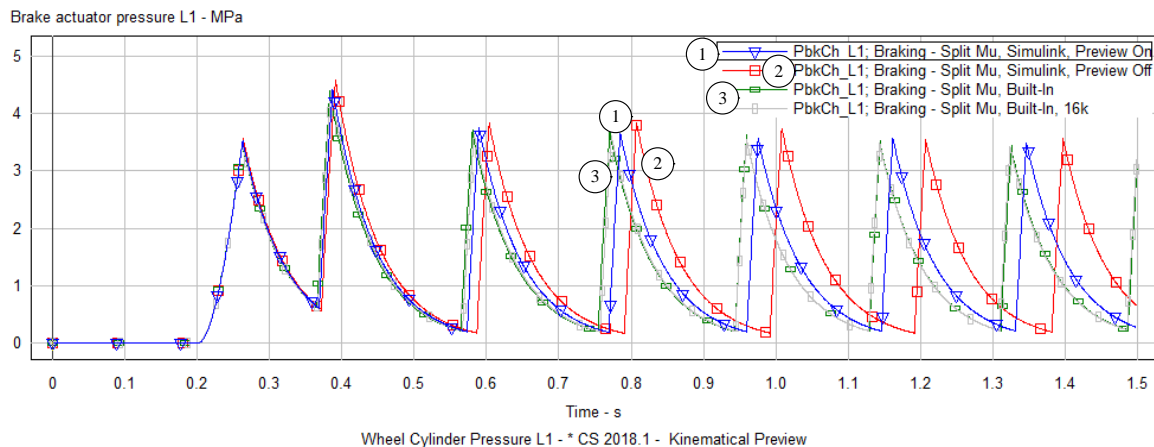


Figure 29. Brake actuator cycling in ABS with and without synchronization.

The figure shows close agreement between the built-in controller with normal time step (2 kHz) and high frequency (16 kHz) ③. Both Simulink plots show some lag that slowly grows, with the case of Preview Off ② being more drastic than the case of Preview On ①.

Even with the Preview On, there is some lag due to the connection: Simulink uses a variable-step integration and its own initializations. However, the Preview reduces the lag significantly.

Closer agreement between the Simulink and built-in ABS models is obtained when the less accurate Euler integration method is used (Figure 30). In this example, Euler integration is used for two simulations made in Simulink (with and without preview), and the built-in version. In this case, very close agreement is seen between the built-in version and Simulink with preview (① and ③). However, both differ from the reference made using the built-in version with a higher frequency (16 kHz) update (④). In terms of matching the reference, the results using the AB-2 integration method are closer, as was shown in Figure 29.

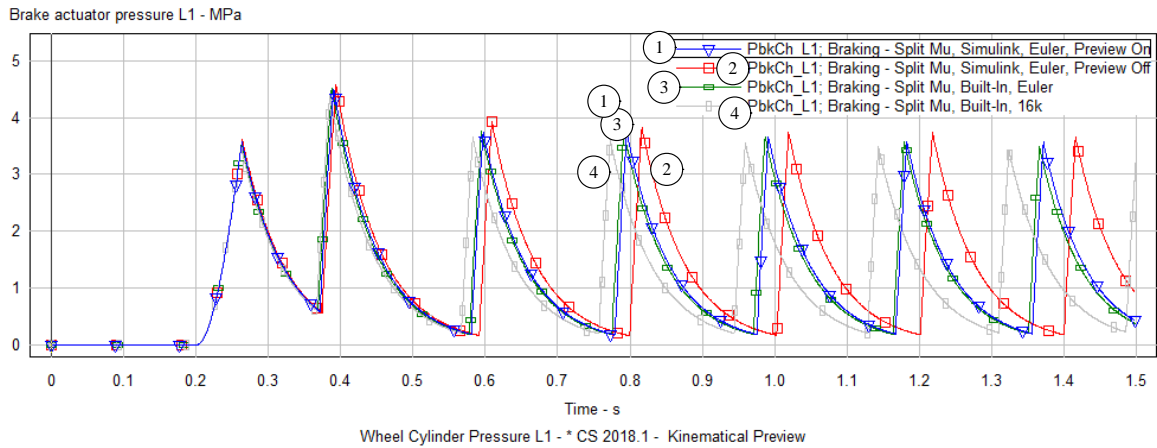


Figure 30. Example with ABS cycling using Euler integration, with and without synchronization.

The preview feature works well when the imported variables are dynamical (forces and moments, and in this case, actuator pressures) calculated externally using exported kinematical variables (e.g., wheel speeds). However, a time lag cannot be avoided if the imported variables include both kinematical and dynamical variables.

6. The Simulation Process

A VS simulation run on Windows occurs in five major steps:

1. Load the selected VS Solver library into memory.
2. Construct the VS Math Model and initialize.
3. Run the simulated test.
4. Terminate the run.
5. Unload the VS Solver library.

The full process has many steps that involve interaction between various factors:

- how the vehicle is configured (trailers, suspension types, etc.),
- how variables are calculated using equations built into the solver,
- how information is obtained from input files,
- how variables are exchanged with external software such as Simulink,
- how equations are processed that are added at runtime with VS Commands, and
- how externally defined functions that might be installed using the VS API are applied.

The *VehicleSim API Manual* has the definitive description of the simulation process that includes all the above factors; *VS Commands Manual* has a description that excludes the API interactions.

This chapter provides a simplified description of the process for basic use of the VS Math Model, without considering VS Commands or extensions made with the VS API.

The following sections describe the behavior of the VS Math Model after it has been constructed, up until the run is completed and the solver library can be unloaded from Windows.

Initialize

Before a simulation run can be started, a minimal VS Math Model performs these steps.

1. Read a Simfile to obtain names of input and output files; allocate memory, create an internal database, and define some core system variables.
2. Begin reading the input Parsfile and obtain the model layout. From that information, activate modules based on the layout, extend the database with model parameters and variables, and continue to read from input Parsfiles. After the Parsfiles have been read, determine how many variables will be imported and exported, and allocate the Import and Export arrays accordingly. At this point, the VS Math Model has been constructed for the simulation requirements.
3. Perform initialization calculations that depend on the values read in step 2.
4. Calculate initial conditions for built-in variables.

5. Possibly apply all model equations to calculate all output variables, and determine forces and moments that might affect compliances in the vehicle model. (This step can be skipped by setting the system parameter `OPT_SKIP_INIT_DYN` to a non-zero value. However, this is usually not recommended.)
6. Write an Echo file that lists all of the model parameters and Configurable Functions. The values are labeled and converted from internal SI units to user units.
7. Prepare to record output variables during the simulation run. For ERD/VS files, write the header. Open the file that will contain written variables, or, if running on an RT system, allocate memory for a buffer for storing the variables until the run finishes.

After the above steps, the simulation begins to run.

Run

The simulation is made by incrementing time T by a small step `TSTEP` and performing calculations to update the state variables and output variables for the new time $T + \text{TSTEP}$. Depending on system settings, three possible sequences are performed each time step.

Single-Step Integration, Conventional Sequence

The Euler and AB-2 integration methods calculate model equations at the time step specified with the parameter `TSTEP`. When the parameter `OPT_IO_SYNC_FM` is zero, the following actions are taken that all apply for the current simulation time T :

1. Copy variables from an Import array that is shared with an external workspace such as Simulink. Convert the imported variables from user units to internal SI units.
2. Calculate values for all variables in the model: built-in output variables, derivatives of state variables, and extra state variables (not defined with ODEs). Convert all potential outputs from internal SI units to user units.
3. Possibly write output variables to a file for later viewing in post-processing. For some real-time systems, the variables might be saved in memory for later writing to file. Outputs are usually not written every time step but are written at multiples based on the parameter `IPRINT`.
4. Copy activated export variables from the internal model to the Export array if there is one.
5. Numerically integrate the derivatives of the state variables that are defined with ODEs to obtain values of state variables for the next time step, at time $T + \text{TSTEP}$.

Single-Step Integration, Synchronized Imports and Exports

When the Euler or AB-2 integration method is selected, and there are both import and export variables activated, and the parameter `OPT_IO_SYNC_FM` is nonzero, then the following actions are taken that all apply for the current simulation time T as defined in the external simulation workspace (e.g., Simulink):

1. Copy variables from the Import array that is shared with an external workspace such as Simulink. Convert the imported variables from user units to internal SI units.
2. Calculate values for all dynamical variables in the model (skip the kinematical equations): apply the dynamical equations for time T . These include derivatives of state variables, extra state variables (not defined with ODEs), and output variables that are not calculated with the kinematical equations. Convert all potential outputs from internal SI units to user units.
3. Possibly write output variables to a file for later viewing in post-processing. For some real-time systems, the variables might be saved in memory for later writing to file. Outputs are usually not written every time step but are written at multiples based on the parameter `IPRINT`.
4. Numerically integrate the derivatives of the state variables that are defined with ODEs to obtain values of state variables for the next time step, at time $T + TSTEP$.
5. Calculate values for all kinematical variables in the model for time $T + TSTEP$: apply the kinematical equations to calculate derivatives of state variables, extra state variables (not defined with ODEs), and output variables that are part of the kinematical equations. Convert all potential outputs from internal SI units to user units.
6. Copy activated export variables from the internal model to the Export array.

With this approach, the variables calculated in the kinematical functions apply for time $T + TSTEP$, while all other variables (forces, accelerations, miscellaneous outputs) still have values calculated for time T . The next time step, forces and moments imported from the external model will be synchronized with the new time $T + TSTEP$.

Half-Step Integration Methods

As noted in Chapter 5, the AM and RK numerical integration methods break the calculations into two parts: the first is a set of calculations applied at multiples of the specified time step `TSTEP`, and the second is a set of calculations made at the half step.

When used with external simulation software (e.g., Simulink), the system parameter `OPT_IO_UPDATE` is used to specify one of two options for communicating with the VS Math Model block. When the parameter `OPT_IO_UPDATE` is zero, the calling program communicates with the VS Math Model at the main time step; when given a nonzero value, the calling program communicates twice as frequently, with the time interval $TSTEP/2$.

Actions at the full step TSTEP

The VS Math Model performs the following actions at each full time-step (multiples of the system parameter `TSTEP`):

1. Copy variables from an Import array (if there is one) that is shared with an external workspace such as Simulink. Convert the imported variables from user units to internal SI units.
2. Calculate values for all variables in the model: built-in output variables, derivatives of state variables, and extra state variables (not defined with ODEs). Convert all potential outputs from internal SI units to user units.

3. Possibly write output variables to a file for later viewing in post-processing. For some real-time systems, the variables might be saved in memory for later writing to file. Outputs are usually not written every time step, but are written at multiples based on the parameter
4. Copy activated export variables from the internal model to the Export array if there is one.
5. Numerically integrate the derivatives of the state variables that are defined with ODEs to obtain values of state variables for the next time step, at time $T + TSTEP/2$.

Actions at the half step

The VS Math Model performs the following actions at each half step. The time T is now at a half-step value.

1. If the parameter `OPT_IO_UPDATE` is nonzero, copy variables from an Import array (if there is one) that is shared with an external workspace such as Simulink. Convert the imported variables from user units to internal SI units.
2. Calculate values for all variables in the model: built-in output variables, derivatives of state variables, and extra state variables (not defined with ODEs). Convert all potential outputs from internal SI units to user units.
3. Write output variables to a file for later viewing in post-processing only in the special case where `IPRINT` was set to 0.
4. If the parameter `OPT_IO_UPDATE` is nonzero, copy activated export variables from the internal model to the Export array if there is one.
5. Numerically integrate the derivatives of the state variables that are defined with ODEs to obtain values of state variables for the next time step, which will be a full step.

Terminate

When the run ends, the VS Math Model cleans up and shuts down.

1. Set parameter values for the possibility that a future run will continue from where this one stopped. For example, set the start time to the current time, and turn off some initialization options to allow the VS Math Model state variables to be left intact.
2. Finish the writing of outputs to a binary file that goes along with the VS or ERD header file:
3. Write to the binary file if variables were saved in memory (e.g., for real time operation).
4. Close the binary file.
5. Write an End Parsfile that can be read as input to continue the run later.
6. Free memory that was explicitly allocated during initialization.

7. More Information

This document has described the general design and operation of a VS Math Model, with the intent of providing information to advanced users who wish to integrate the models with other software.

It described the basic operation, including the contents of input and output files, with emphasis on the use of an input Parsfile to specify properties of the VS Math Model with parameters, Configurable Functions, and arrays. The overall operation of the VS Math Model to solve equations of motion and perform the simulation was then presented.

The **Help** menu links to other documents with useful information about VS Math Models:

- *System Parameters in VS Math Models* describes system-level parameters that are available in all VS Math Models, such as time step, starting and stopping conditions, etc.
- *VS Commands* is intended for users who want to extend a VS Math Model by adding new variables and equations. A few simple equations can often provide extensive capability for simulating advanced controls and complex test conditions. This manual also describes the capabilities of the VS Math Models for handling symbolic expressions.
- *Extending a Model with Embedded Python* describes how to activate Python and use Python functions within the VS Math Model via VS Commands.
- *VS API: Accessing and Extending VehicleSim Math Models* describes how VS Math Models can be constructed and run under other programs written in MATLAB, C, Visual Basic, and other programming languages.
- *VS Visualizer Reference Manual* describes the interactive use of *VS Visualizer* for viewing simulation results through video and plotting.
- *Numerical Integration Methods in VS Math Models* is a tech memo that describes the numerical integration methods in detail, and discusses some advanced topics related to numerical integration.
- *VS Solver Wrapper* is a tool for running constructing and running VS Math Models from the Windows Command Prompt. The Solver Wrapper is a useful for cases running simulations without the VS Browser. The same tool is available on various Linux operating systems.

Mechanical Simulation



755 Phoenix Drive, Ann Arbor MI, 48108, USA

Phone: 734 668-2930 • Fax: 734 668-2877 • Email: info@carsim.com

carsim.com